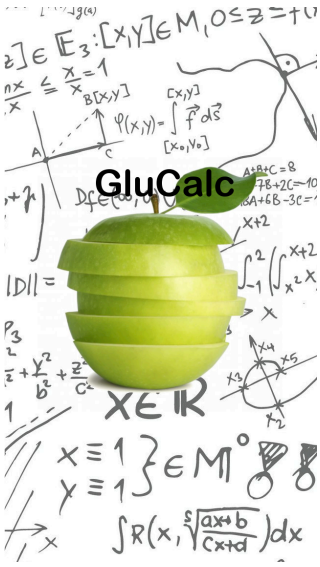
h e g
Haute école de gestion de Genève
Geneva School of Business Administration

GluCalc

Application iOS pour les diabétiques



Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES

par :

Alberto CLOP

Conseiller au travail de Bachelor :

Peter DAEHNE

Professeur HES

Genève, août 2013

Haute École de Gestion de Genève (HEG-GE)

Filière Informaticien de gestion

Déclaration

Ce travail de Bachelor est réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre de Bachelor en informatique de gestion. L'étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de Bachelor, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de Bachelor, du juré et de la HEG.

« J'atteste avoir réalisé seul le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie. »

Fait à Genève, le 23 août 2013

Alberto Clop

Remerciements

Premièrement, je remercie M. Louis Boyer qui est le créateur de ce projet. Merci de m'avoir offert l'opportunité de réaliser ce projet qui vous tient tant à cœur et merci pour votre professionnalisme car vous m'avez fourni une bonne documentation technique me permettant de me concentrer sur le développement et la proposition d'améliorations.

Je remercie le CHUV qui nous a un peu financé, M. Michael Hauschild qui a validé et simplifié les calculs.

Je remercie l'entreprise SENTE spécialisée dans le développement mobile qui nous a fourni un support initial pour bien commencer le projet.

Je remercie M. Paul Hegarty qui a mis à disposition gratuitement ses cours dans iTunes que j'ai suivis durant ma première semaine de thèse. A partir des cours de Java et Android de M. Peter Daehne, j'ai pu aisément suivre ce cours sur internet.

Résumé

En 2008, mon client appris par le médecin que son enfant âgé de seulement cinq ans souffrait du diabète de type 1. Dès lors sa famille dut se soumettre aux calculs des glucides lors des repas. Ces calculs sont extrêmement importants, car ils permettent de déterminer la quantité d'insuline à injecter à leur enfant. Leurs résultats doivent être transmis au médecin pour permettre un suivi médical.

Un jour, mon client eut l'idée de créer un formulaire Excel permettant d'automatiser ces calculs rébarbatifs, ce qui permit à la famille de gérer plus facilement les données et de profiter plus simplement des repas. Après cinq années d'utilisation et d'amélioration du formulaire, il décida de créer une application mobile qui s'appellera GluCalc.

GluCalc fournira à un patient la possibilité d'enregistrer les types de repas avec les données fournies par le médecin ; cible alimentaire, glycémie cible, facteur de sensibilité et bolus. Il pourra enregistrer ses aliments et renseigner le ratio entre glucides et quantité. A partir d'un type de repas et d'aliments, il pourra enregistrer ses repas ; la quantité d'insuline a administré lui sera indiqué avec la possibilité de l'ajuster. Les repas seront historisés pour permettre la consultation mais aussi l'export pour les transmettre a son médecin. Ceci n'est qu'une partie des fonctionnalités.

Je vous invite à découvrir GluCalc dans mon mémoire ou sur l'Apple Store.

Table des matières

1. Introduction

Cette thèse englobe la phase de développement du projet GluCalc (www.glucalc.ch). Celui-ci offre un logiciel mobile qui a pour but d'alléger la tâche du patient diabétique dans les calculs à effectuer au moment du repas. Il permet non seulement de calculer les hydrates de carbone (glucides) en fonction des aliments sélectionnés, mais aussi de déterminer la quantité d'insuline nécessaire selon le schéma prescrit par le médecin.

En tant que développeur, j'ai donc la responsabilité de discuter avec mon client des possibilités techniques permettant de réaliser son projet. Nous discutons de la faisabilité des fonctionnalités et des possibilités offertes par l'environnement iOS. A partir des schémas d'écrans fournis par mon client, je propose une ergonomie pour optimiser l'utilisabilité et le développement de l'application.

J'étudierai les différents outils et composants offerts par le SDK d'Apple et en discuterai avec des experts (Sente) pour offrir à mon client une application réactive, stable et pérenne. Puis, je développerai l'application.

A la fin de ma thèse, je livrerai l'application à mon client et ferai un transfert de connaissances à l'entreprise qui sera responsable de sa maintenance pour permettre une publication rapide dans l'Apple Store.

2. Contexte

M. Boyer, mon client, a découvert en 2008 que son enfant de cinq ans souffrait du diabète de type 1. Dès lors sa famille s'est adaptée aux calculs manuels des glucides lors des repas.

Le traitement du diabète est très prenant ; il demande au patient de calculer les hydrates de carbone (ou glucides) qu'il ingère. C'est à partir de ce calcul que l'on peut déterminer la quantité d'insuline à injecter pour lui permettre de métaboliser ce qui a été ingéré.

Comme mon client le dit, et c'est son principal objectif :

« Chaque seconde épargnée sur les calculs permet de garder la motivation pour atteindre l'équilibre et les objectifs thérapeutiques. Ce gain de temps préserve la convivialité des repas et permet de conserver une alimentation riche et variée. »

Etant chef de projets informatiques, M. Boyer a choisi d'automatiser l'ensemble des calculs nécessaires à la bonne gestion d'un repas dans un tableur Excel.

Cela fait maintenant cinq ans qu'il l'utilise et l'améliore. Son fils commence à être autonome, il est donc nécessaire de faire évoluer l'outil pour qu'il soit transportable.

3. La solution

Souhaitant offrir à son fils un outil transportable, facile d'utilisation mais aussi avec une volonté de le partager avec d'autres patients, M. Boyer s'est orienté vers une solution mobile. Après avoir analysé les différents supports mobiles et la possibilité d'utiliser des outils qui génèrent des applications pour plusieurs plateformes (Android, iOS, BlackBerry, etc..), il s'est orienté vers une solution native iOS.

L'application sera donc utilisable sur les iPhone et les iPod touch.

4. Fonctionnalités prévues

4.1 Gestion des catégories d'aliments

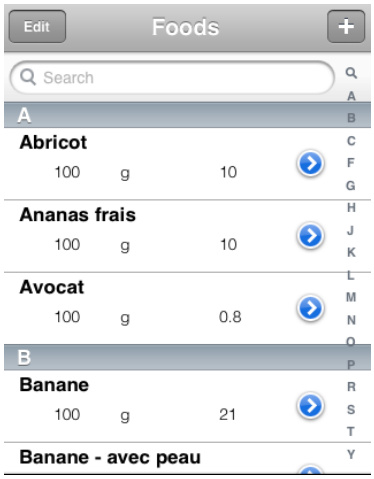
Dans cette application tous les aliments sont catégorisés. Ces catégories sont très utilisées par les nutritionnistes. Nous avons donc créé une vue permettant de créer, modifier et supprimer des catégories. La suppression d'une catégorie entraîne la suppression de tous les aliments appartenant à cette même catégorie. Bien entendu, l'utilisateur est prévenu par une pop-up d'information.

4.2 Gestion des aliments

La gestion des aliments est la fonctionnalité de base de cette application, elle permet la création, la modification et la suppression.
La création d'un aliment nécessite la sélection de sa catégorie et de renseigner la valeur en glucides pour une quantité donnée.

Ici nous avons un Abricot qui se trouve dans la catégorie Fruits contenant 10 grammes de glucides pour 100 grammes. Le ratio entre glucides et quantité sera utilisé lors de la saisie d'un repas. Il suffira à l'utilisateur de choisir un aliment puis de renseigner soit la quantité ingérée (poids, volume), soit la quantité de glucides que cela représente pour que l'autre valeur soit calculée automatiquement.

Comme l'application référencera un grand nombre d'aliments, les listes d'aliments contiennent un champ de recherche ainsi qu'une liste d'index permettant l'accès rapide à un aliment donné.



4.3 Gestion des types de repas

Un repas contient les données suivantes permettant les différents calculs lors d'un repas :

- Cible alimentaire : Quantité de glucides recommandée pour un repas
- Glycémie cible : valeur de glycémie souhaitée après le repas
- FSI – Facteur de sensibilité
- Bolus : En médecine et en pharmacie, le terme bolus désigne une dose de médicament que l'on doit administrer au complet d'un seul coup, généralement par injection.

Ces valeurs sont fournies par le médecin, il est possible de les protéger par un mot de passe pour éviter tout changement par des utilisateurs de l'appareil mobile.

4.4 Gestion des aliments favoris

Pour rendre plus rapide la saisie des repas, il est possible de renseigner des aliments favoris par repas. Comme dans la saisie du repas, il est possible de modifier la quantité d'aliment ou la quantité de glucides pour chaque aliment.

Par exemple, pour le petit-déjeuner, je suis habitué à manger des bananes et boire du jus d'oranges ; je peux donc les mettre en favoris de ce type de repas. Et donc, chaque jour, lors de la création d'un nouveau repas de ce type, je pourrais par une simple case à cocher indiquer si je veux le pré-remplir avec les aliments favoris.

4.5 Création d'un repas

Il s'agit de la fonctionnalité principale de ce logiciel ; elle permet de renseigner les aliments mangés et d'obtenir le calcul du bolus que l'on doit s'administrer. Ceci se passe en trois étapes :

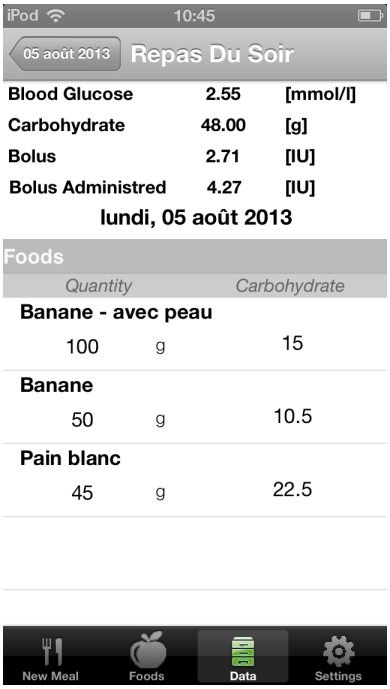
- Choix du type de repas, renseignement du taux de glycémie avant le repas puis choix de l'ajout des aliments favoris.
- Ajout des aliments du repas, en indiquant leur quantité. Il est possible de renseigner uniquement la quantité ou la quantité de glucides d'un aliment, l'autre valeur sera automatiquement calculée à partir des valeurs de base.
  A chaque modification, le total de glucides et la valeur du bolus sont mises à jour.
- A partir du bolus calculé, l'utilisateur peut ajuster la valeur pour indiquer ce qui sera réellement administré. Ceci donne une flexibilité à l'utilisateur pour prendre en compte des facteurs qui ne sont pas dans le calcul, comme les activités de la journée (sport).

A la fin, la totalité de ces valeurs sont archivées, y compris la valeur ajustée du bolus.

4.6 Consultation du journal des repas

L'utilisateur peut consulter tous ses repas organisés par date, puis listés par heure.

Pour chaque repas, la liste des tous aliments est disponible avec leur quantité et les glucides ingérés, ainsi que toutes les valeurs du repas (Bolus, Bolus administré, total des glucides, date, etc.).



4.7 *Import*

Un fichier csv d'aliments avec leur catégorie existe dans les sources ; il permet de définir des valeurs de base au premier lancement ; une option dans les réglages permettra de ré exécuter cet import. Ce fichier existe en français et anglais ; la version française sera importée si le système est en français sinon ce sera la version en anglais. Une version allemande est en cours de préparation.

Le logiciel permet l'import d'aliments provenant d'un fichier csv externe, par exemple provenant d'un mail. Le fichier devra avoir l'extension « .glucalc » et contenir « Foods.csv » dans son nom.

Le nom de l'aliment étant employé comme identifiant dans la base de donnée, si un aliment de même nom existe déjà, ses informations seront mises à jour.

4.8 Export

Dans cette application, nous exportons deux types de données : les aliments et l'historique des repas.

L'export des aliments s'effectue dans le format csv, ce qui permet de les sauvegarder, de les échanger avec d'autres utilisateurs ou même de les éditer sous Excel (ou un autre éditeur) pour les réimporter.

L'export de l'historique des repas peut se faire en csv ou sous format texte. Il permet d'effectuer des analyses sur les données, ce qui est très pratique pour transmettre rapidement et simplement nos données à notre médecin. Car outre les visites prévues chez le médecin, en cas de malaise, il est nécessaire d'envoyer ces informations pour permettre aux médecins de nous aider au plus vite.

4.9 Autres fonctionnalités

- Des conditions d'utilisations sont à accepter.
- Une vue permettant d'activer la protection des données par un mot de passe.
- Les configurations de base sont affichées au premier lancement, puis accessible depuis les réglages.
- Une vue permet de consulter les ratios de contrôle, exprimant la quantité l'insuline utilisée quotidiennement par rapport au poids corporel du patient.

- Une vue offre des informations pour le calcul du bolus.
- Le logiciel étant multilingue, si le système est en français ou en allemand les textes seront traduits. Dans tous les autres cas le logiciel s'affiche en anglais.

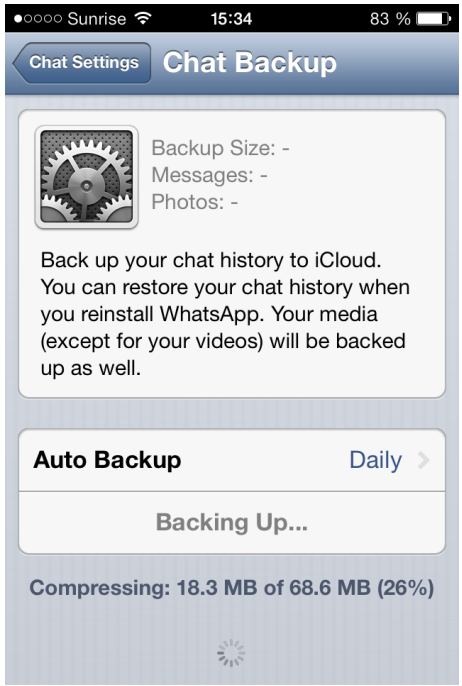
5. Améliorations possibles

5.1 Choix d'une langue

Actuellement, le logiciel se base sur la langue du système. Un réglage pourrait permettre d'en sélectionner une autre.

5.2 iCloud

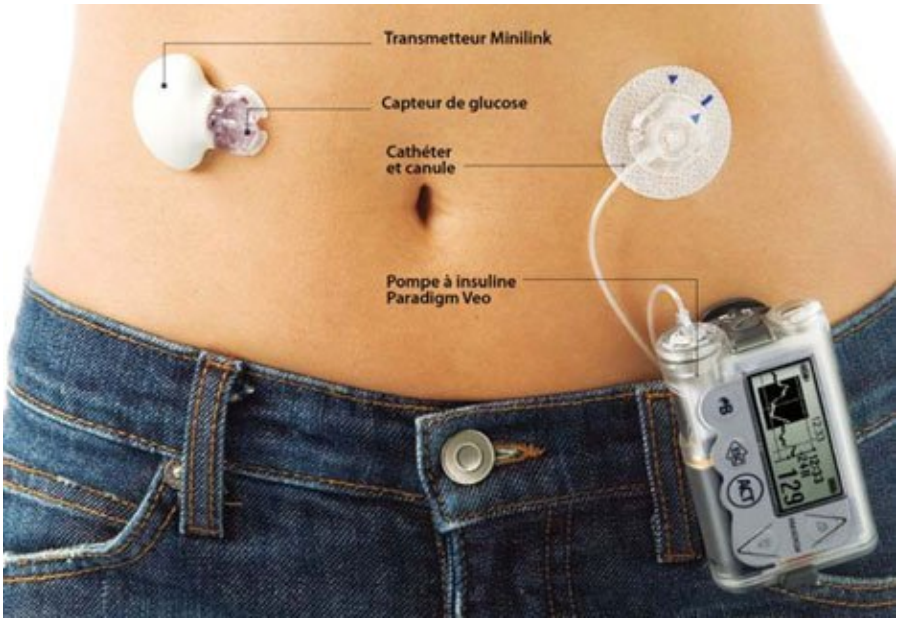
iCloud est le service de stockage en ligne d'Apple, il permet de sauvegarder des données allant d'une simple photo à la sauvegarde complète de votre appareil.
Etant un service fiable et sécurisé, de plus en plus d'applications s'y connectent pour permettre à l'utilisateur de retrouver ses données facilement après un vol ou un crash du système. Ci-dessous par exemple, l'utilisation de ce service par l'application WhatsApp (logiciel de chat) :



Dans notre application, nous regroupons les données dans une base de données locale ainsi que dans les UserDefaults. Il nous serait donc facile d'extraire toutes les données pour les sauvegarder automatiquement dans l'iCloud.

5.3 Connexion à d'autres appareils

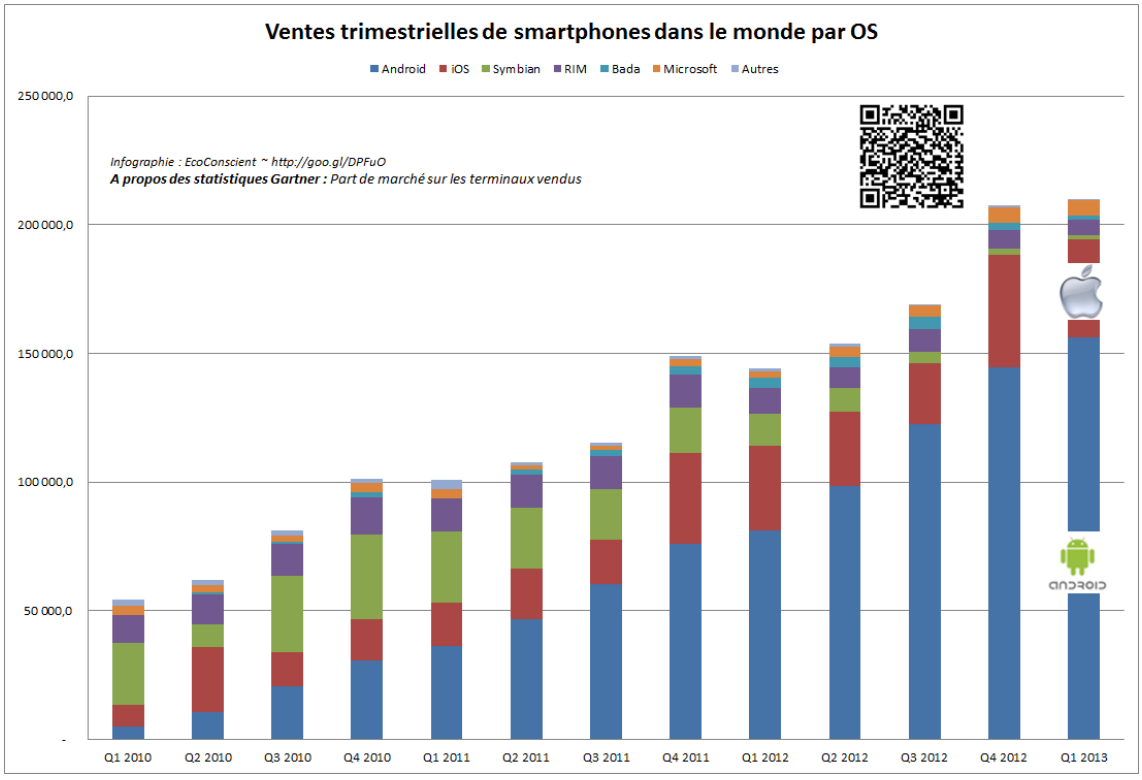
L'application pourrait recevoir ou envoyer des données à un appareil de mesure.
Les pompes à insuline, mises au point dans les années 80, ne cessent de s'améliorer. Elles permettent d'injecter l'insuline, de mesurer la glycémie, voire même d'alarmer le patient en cas d'anomalies majeures. On pourrait envisager un appareil qui permettra de communiquer en Bluetooth. L'application mobile, jumelée à la pompe, recevrait la mesure de glycémie et transmettrait la quantité d'insuline à injecter. Le patient pourrait s'affranchir du besoin de sortir sa pompe pour y lire des informations et la régler.



(http://www.doctissimo.fr/html/dossiers/diabete/articles/14668-nouveaux-progres-pompes-insuline.htm)

5.4 Création d'une version Android

Pour permettre de toucher plus d'utilisateurs, une application doit exister sur plusieurs OS. Selon un article du 14 août 2013 de la société de conseil Gartner, Android reste l'OS le plus répandu. Il devance de loin tous ses concurrents.



Après cette application sous iOS, il est logique de s'intéresser à Android. A eux deux 83,2% des utilisateurs pourraient accéder à cette application.

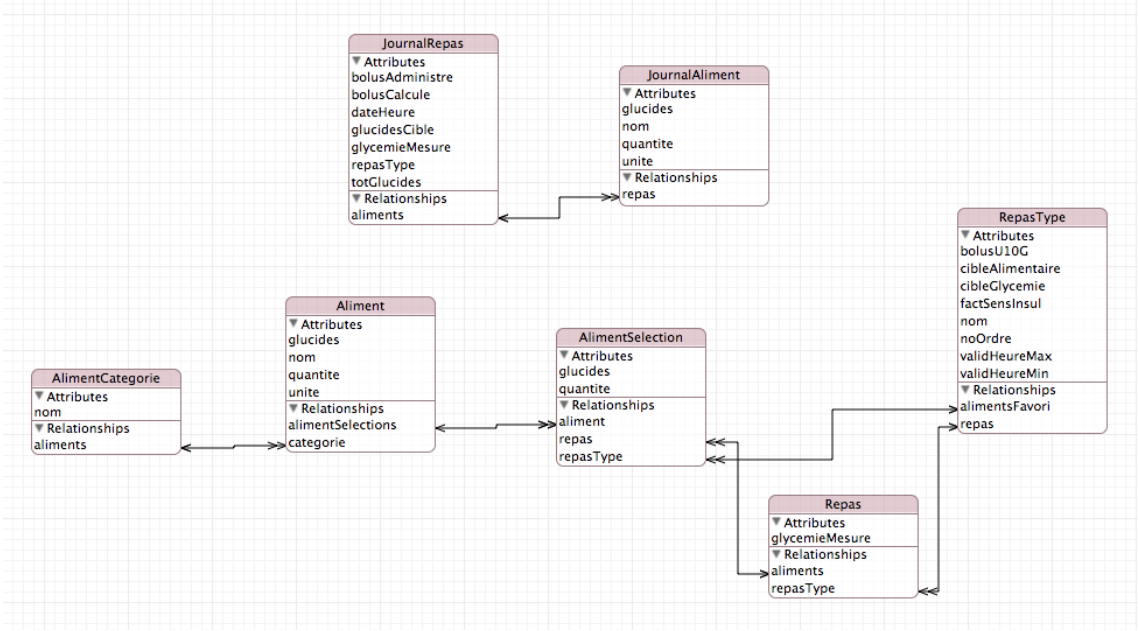
6. Associations et entreprises intéressées

Le CHUV suit le projet et a même débloqué un petit budget pour le support envers Sente. Ils sont donc très intéressés par cette application qu'ils recommanderaient à leurs patients. N'ayant pas encore une vision précise sur leur engagement dans les logiciels mobiles, ils ne pensent pas prendre en charge le logiciel pour le moment.

L'application sera donc vendue à bas prix pour payer la maintenance, voire les évolutions selon le succès.

7. Base de données

A partir du schéma créé par mon client dans son cahier des charges, j'ai créé la base de données ci-dessous.



On remarque dans la partie supérieure les tables permettant de journaliser les repas et leurs aliments tel un Data warehouse, tandis que la partie inférieure représente le jeu de données modifiable.

Les aliments sont catégorisés, méthode utilisée par les nutritionnistes. Ils contiennent en plus le ratio entre quantité et glucides. Ce qui facilite la saisie lors de leur sélection, car à partir de l'un nous pouvons déduire l'autre. Ces aliments peuvent être sélectionnés en tant que favoris d'un type de repas ainsi que lors de la création d'un repas.

Un repas est lié à un type de repas. Ces types, fournis par le médecin, contiennent les valeurs de mesures permettant d'effectuer le calcul du bolus ainsi que de contrôler les quantités ingérées et injectées par rapport à ses conseils.

8. Architecture des sources

Le projet contient la structure de dossiers ci-dessous :

```
Glucalc
        Utils

        View
                AbstractView
                Cell
                ConfigurationsStoryBoardViews
                TableView
                Utils

        Model

        Supporting Files
```

Le répertoire « Utils » du premier niveau contient les classes utilitaires pouvant être utilisées par toutes les classes.

Le répertoire « View » contient toutes les classes décrivant les vues. Il contient dans « AbstractView » toutes les classes mères ; dans « Cell » les classes décrivant une cellule de tableau ; dans « TableView » celles décrivant une table (liste) ; dans « Utils », les utilitaires de vues puis dans « ConfigurationsStoryBoardViews », les vues spécifique au StoryBoard « Configuration ».

Le répertoire « Model » contient toutes les classes décrivant les tables (ou entités) de la base de données.

Le répertoire « Supporting Files » contient que les fichiers sources tel que les images, les traductions, etc.

Dans le répertoire racine, se trouve le fichier de base de données, les storyBoard ainsi que la classe de démarrage.

9. Architecture des classes

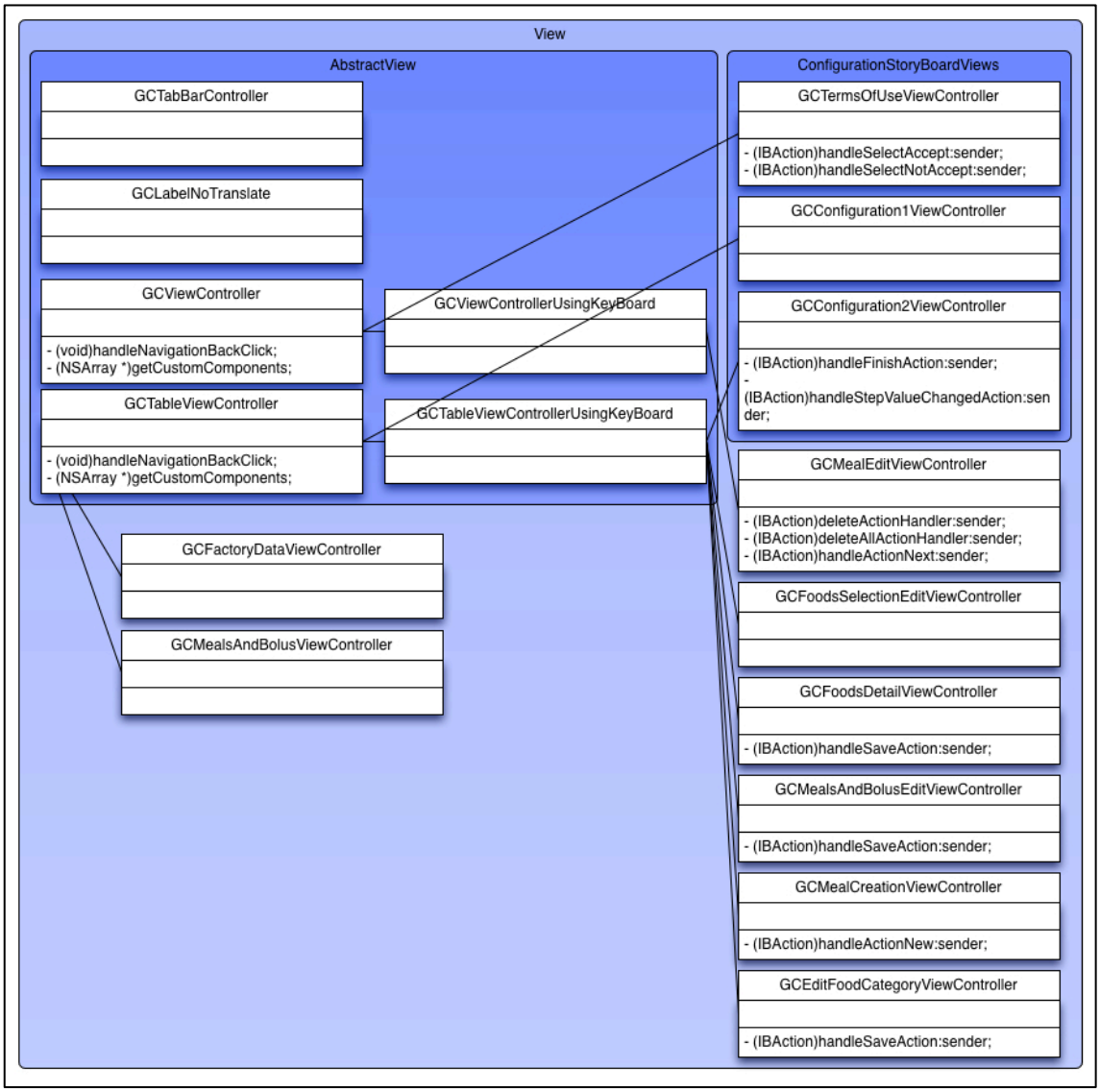
Toutes les classes contiennent une description dans leur en-tête, je vais donc présenter deux exemples d'architecture.

9.1 Classes mères

J'ai créé quatre classes dont toutes les classes représentant des vues doivent hériter.

GCViewController et GCTableViewController qui héritent des classes de bases UIViewController et UITableViewController et ajoutent les fonctionnalités de traductions de tous les textes ainsi que l'ajout automatique d'un bouton « Retour » en haut à gauche de la vue. Elles utilisent la classe GCViewControllerUtils contenant des méthodes de traduction.

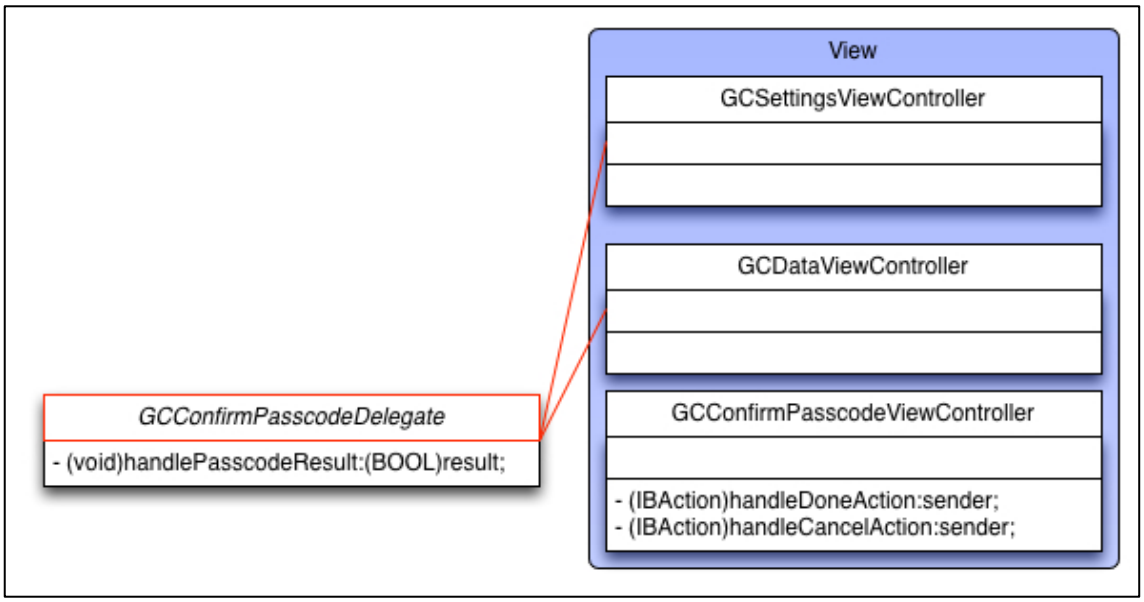
GCViewControllerUsingKeyBoard et GCTableViewControllerUsingKeyBoard héritent de GCViewController et GCTableViewController pour ajouter la fonctionnalité de gestion de la vue avec et sans clavier lors de la saisie dans des champs textes. Elles utilisent la classe GCViewControllerUsingKeyboardUtils qui contient les méthodes permettant la gestion de cette fonctionnalité ainsi que UITextFieldDelegate qui permet de gérer les champs textes. Ci dessous un diagramme affichant une partie des classes.



9.2 Retourner des données

Lorsqu'une vue en lance une autre, il est aisé d'envoyer des données à l'aide d'une variable. Il est plus difficile de retourner des informations à la vue parente. Pour ce faire, j'utilise une interface contenant une ou plusieurs méthodes. Lorsque qu'une vue en appelle une seconde, celle-ci doit implémenter cette interface et s'enregistrer dans une variable de la seconde vue. Cette variable est du type de sa propre interface et est généralement nommée delegate.

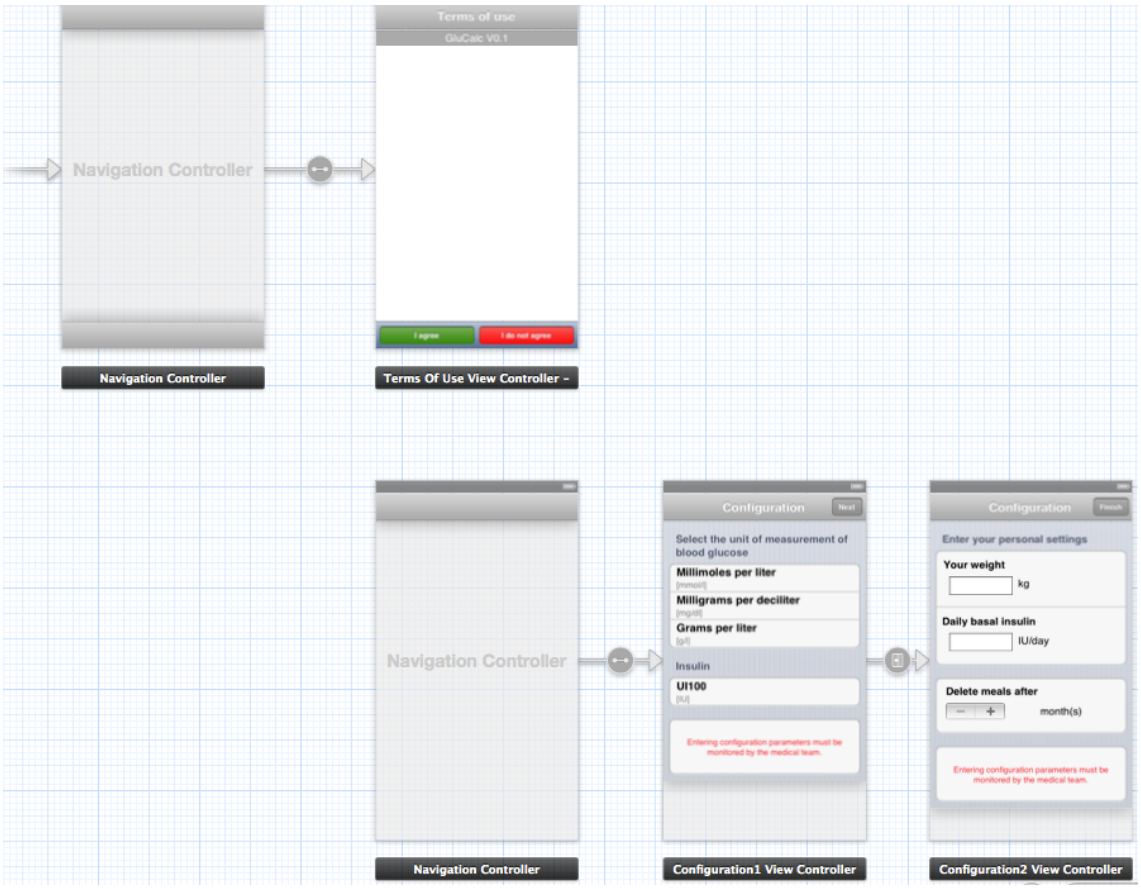
Ci-dessous, un exemple avec la vue GCConfirmPasscodeViewController qui permet à un utilisateur de renseigner le mot de passe pour que celui-ci soit vérifié. Pour informer la vue parente du résultat, cette classe fournit une interface nommée GCConfirmPasscodeDelegate. Les deux voulant afficher cette vue doivent donc implémenter son interface.



10. Les vues

Ce chapitre explique les vues développées dans les storyboard.

10.1 Configuration



Ces vues sont les seules se trouvant dans un storyboard séparé ne dépendant pas du reste de l'application. Elles sont affichées au premier lancement de l'application, puis mises à disposition dans la vue réglages. Elles permettent de changer les paramètres de l'utilisateur.

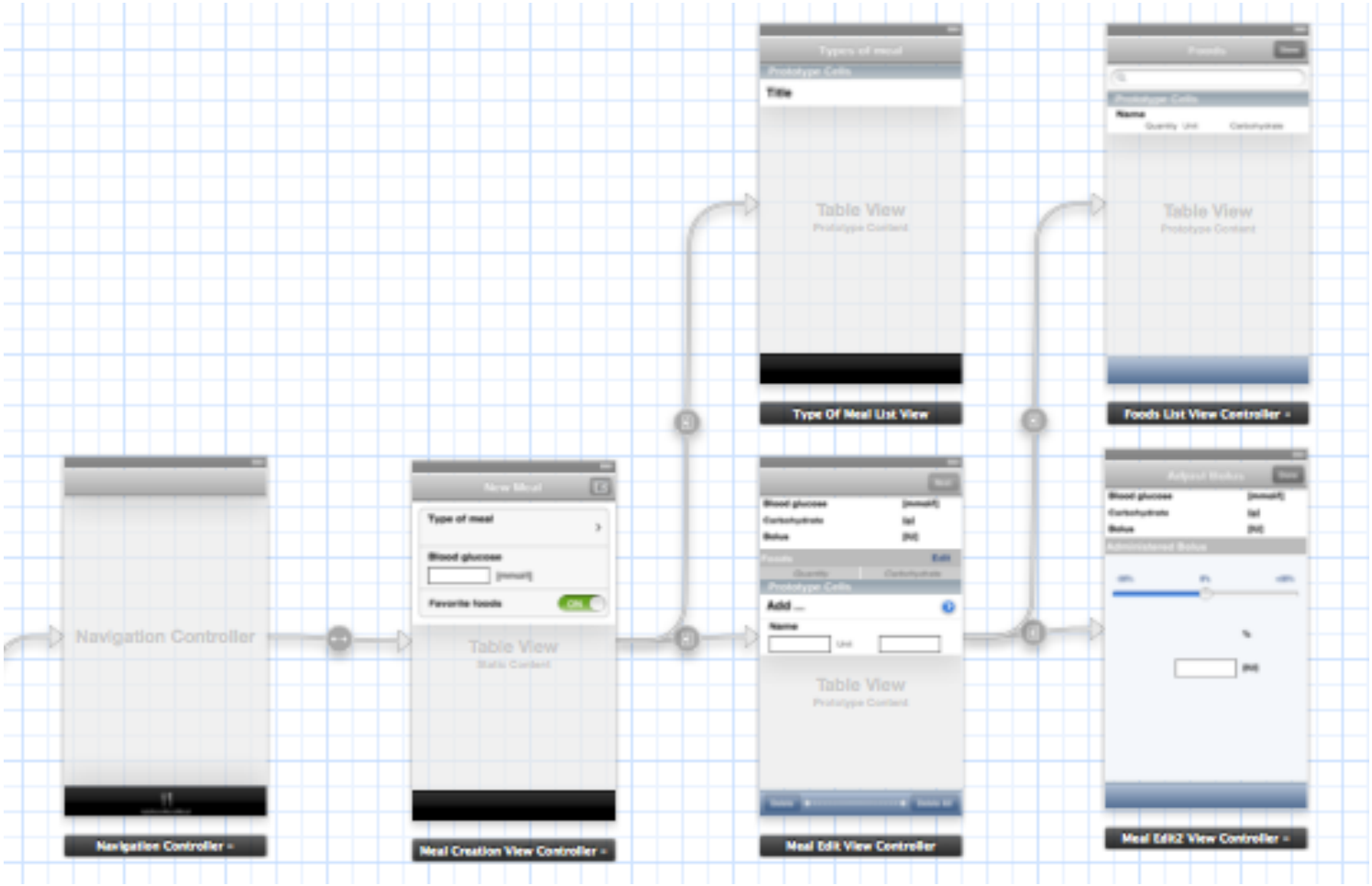
Les vues de configuration sont protégées par un mot de passe si celui-ci est activé.

10.2 Tab Bar Controller

Composant de base de l'application ; il permet de grouper toutes les vues de l'application en quatre groupes : Nouveau repas, aliments, données et paramètres.

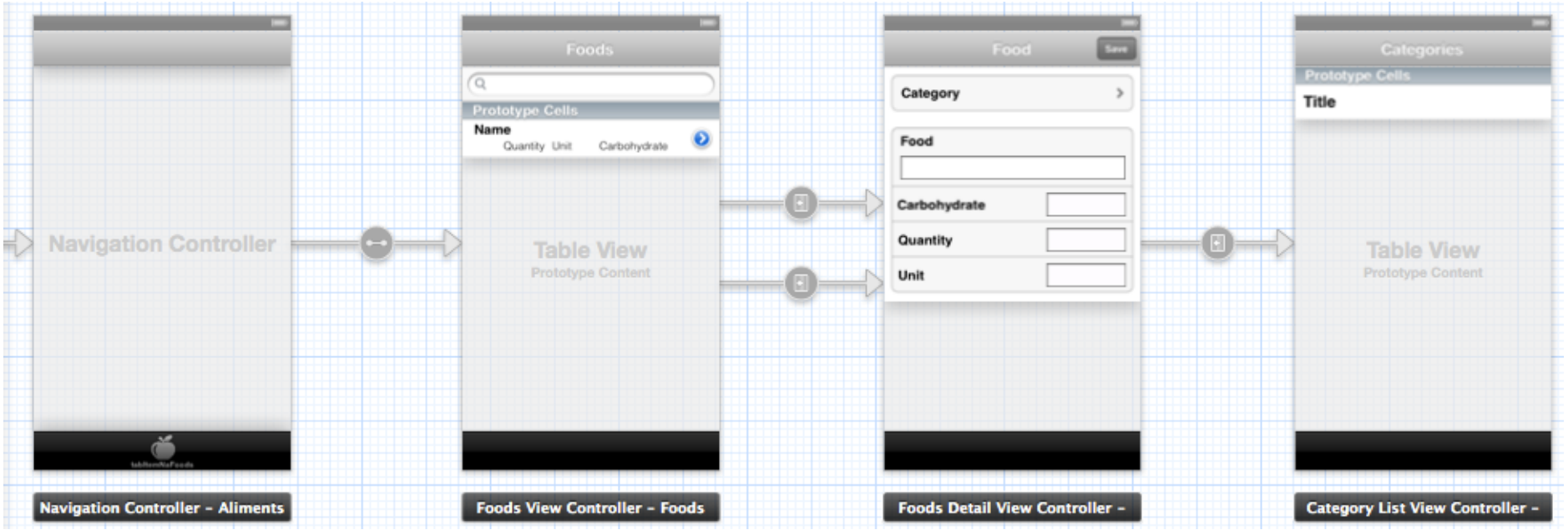
10.3 Nouveau repas

Vues les plus utilisées de l'application, elles permettent de renseigner les repas pris au jour le jour.



10.4 Aliments

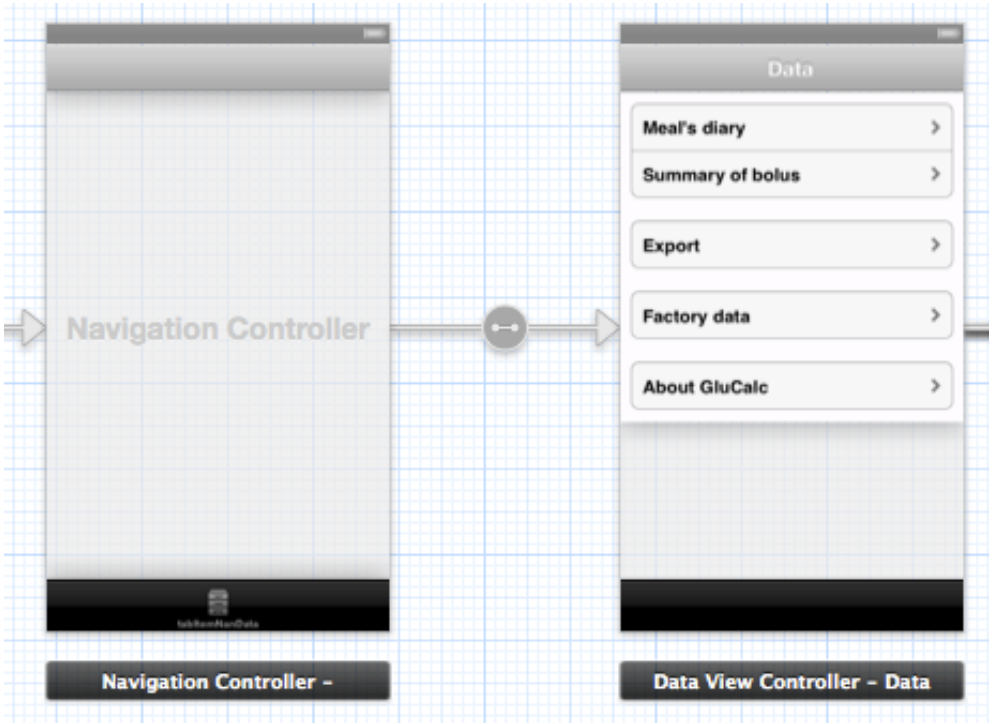
Vues permettant la gestion des aliments ; création, modification et suppression.



10.5 Données

10.5.1 Menu

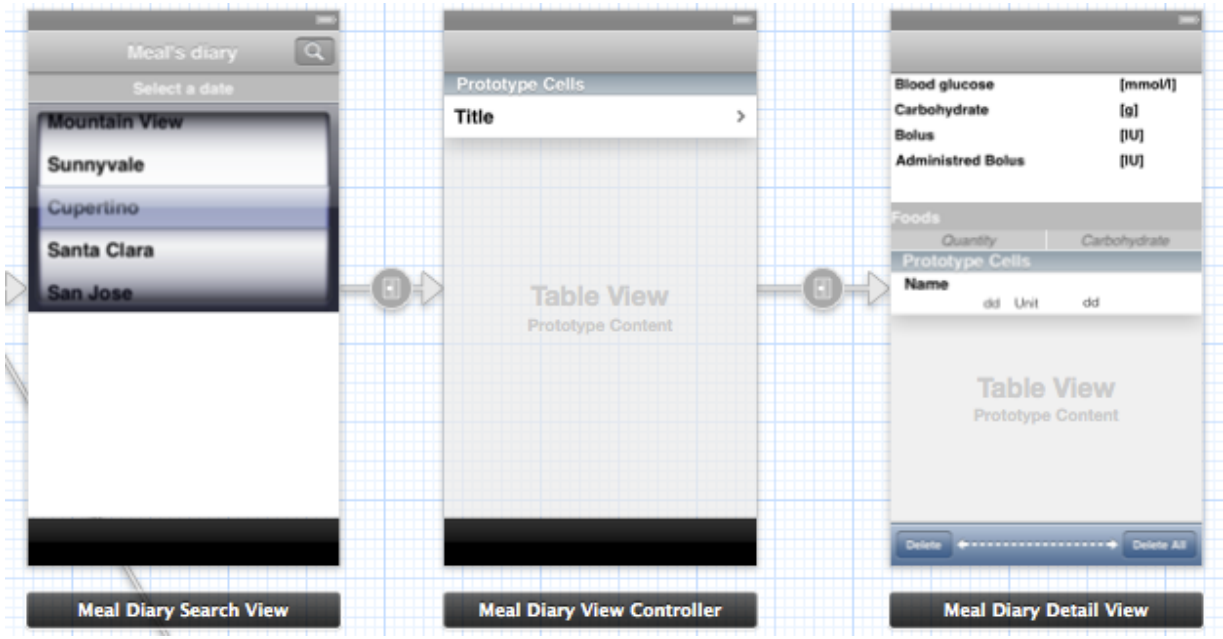
Vue offrant l'accès à toutes les vues du groupe données.



10.5.2 Journal des repas

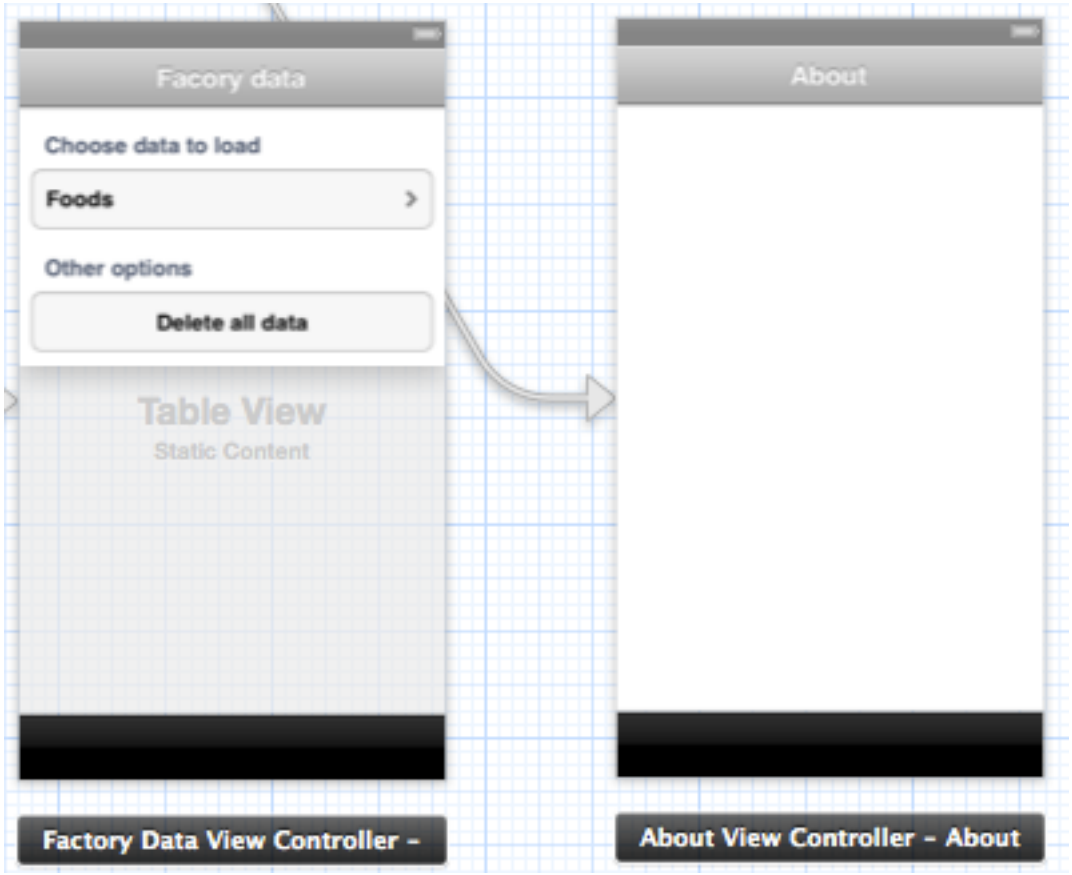
Ces vues offrent l'accès à l'historique des repas renseignés. Les repas sont regroupés par date, puis à la sélection d'une d'entre elles, une liste des repas avec l'heure s'affiche. Finalement, à la sélection d'un repas, la troisième vue s'affiche pour afficher toutes les informations de celui-ci.

Aucune suppression n'a été implémentée dans ces vues, car à chaque démarrage leur date est comparée au nombre de mois du paramètre « Historique des repas » ; tous les repas trop anciens sont supprimés.



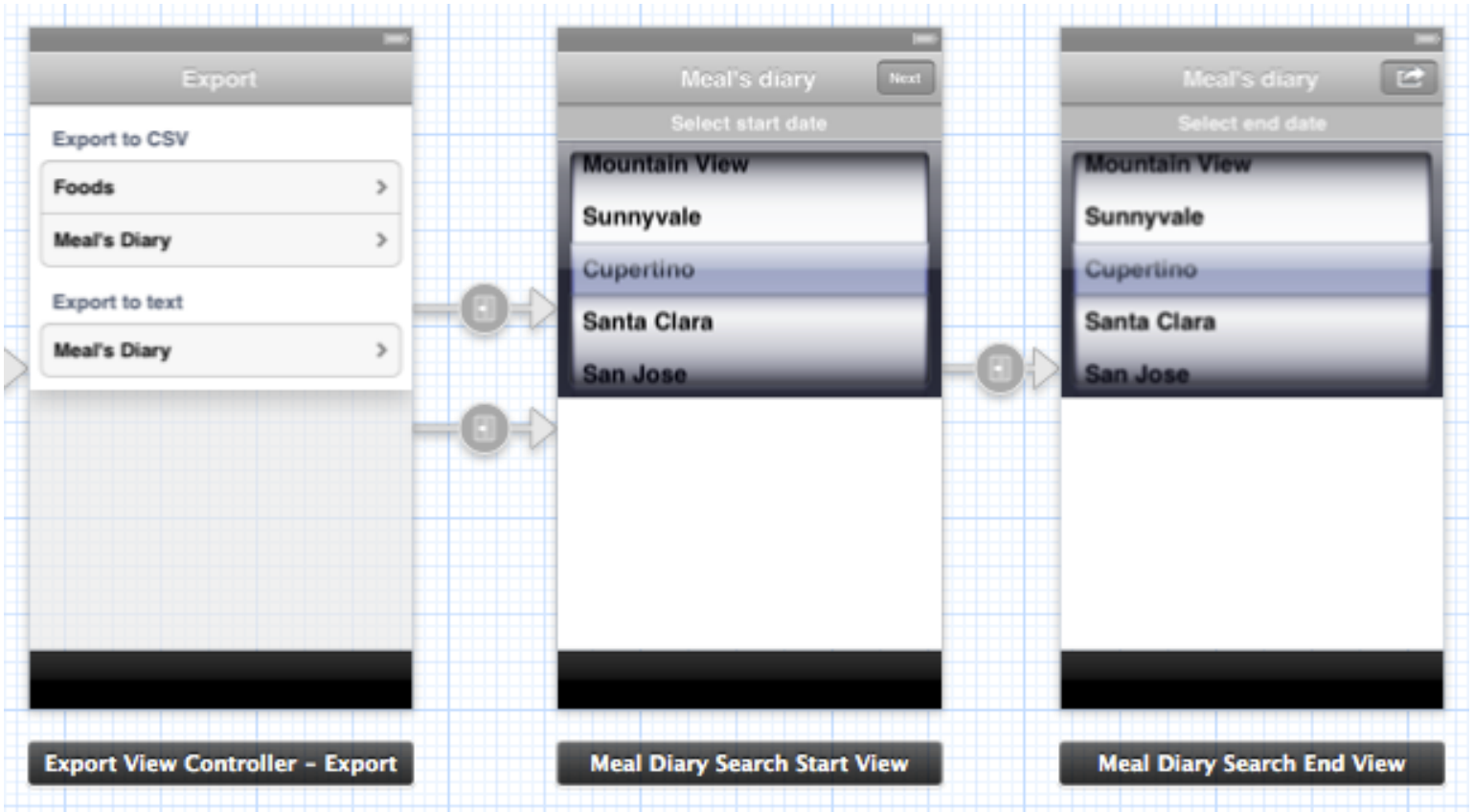
10.5.3 Données d'usine et A propos

La vue « Données d'usines » offre la possibilité de réimporter les aliments d'usine avec une option permettant de supprimer ou non tous les aliments avant l'import, mais offre aussi la possibilité de supprimer toutes les données. Cette vue est protégée par un mot de passe si celui-ci est activé.



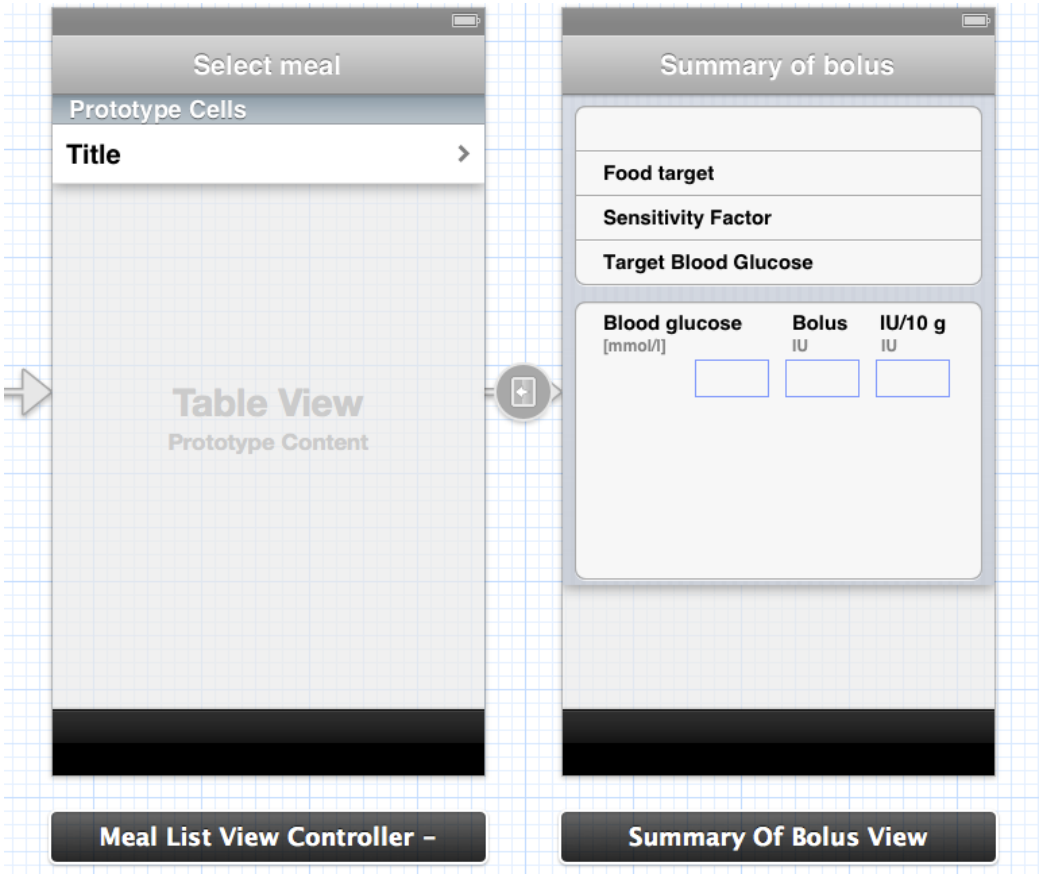
10.5.4 Export

Les deux formats d'export sont représentés par les deux groupes de cellules de la première vue. Lors de la sélection d'une des deux cellules « Journal de repas », une liste apparait affichant toutes les dates disponibles puis à la sélection d'une d'entre elle la seconde liste s'affiche avec les dates qui suivent celle qui à été sélectionnée. Finalement à la sélection de la seconde date, l'export se lance avec les repas se trouvant entre les deux dates.



10.5.5 Sommaire des bolus

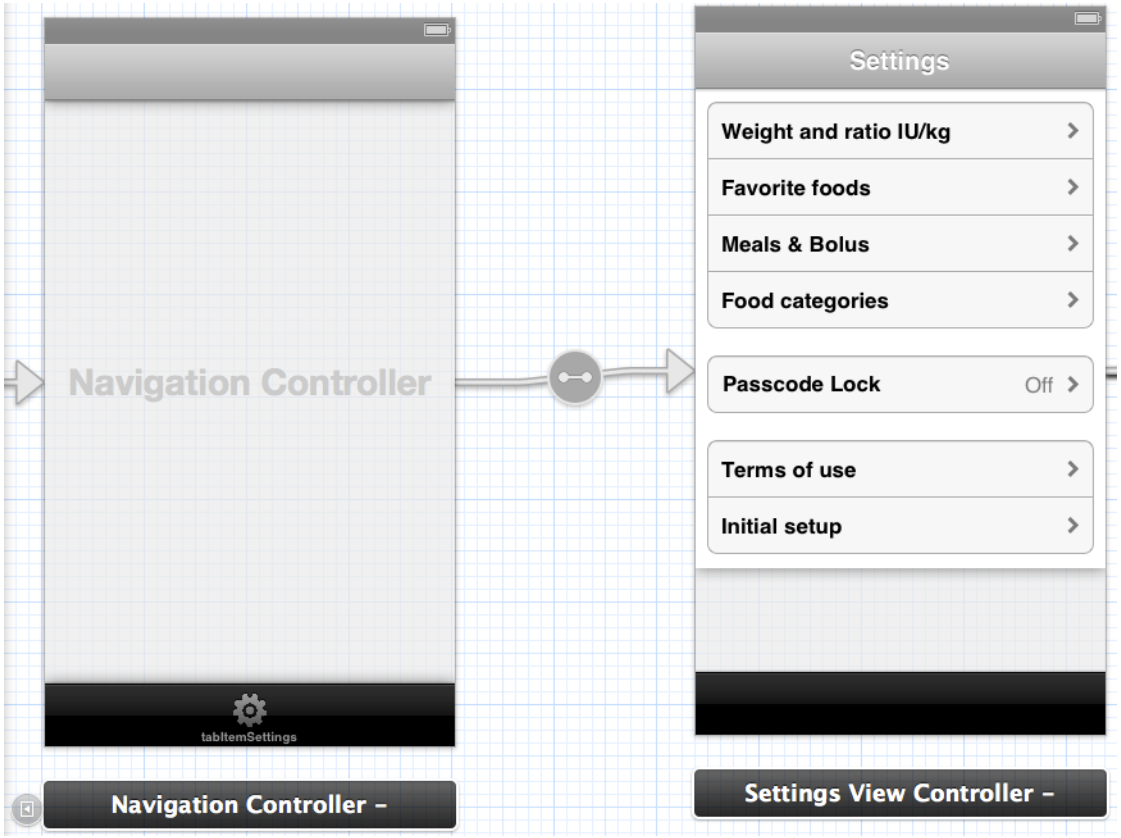
Ces vues offrent des informations sur les paramètres d'un repas ainsi qu'une grille de valeurs des bolus selon les taux de glycémie : 0, 3, 6, 9, 12 et 15. Les patients ont l'habitude de travailler avec ce type de grilles.



10.6 Paramètres

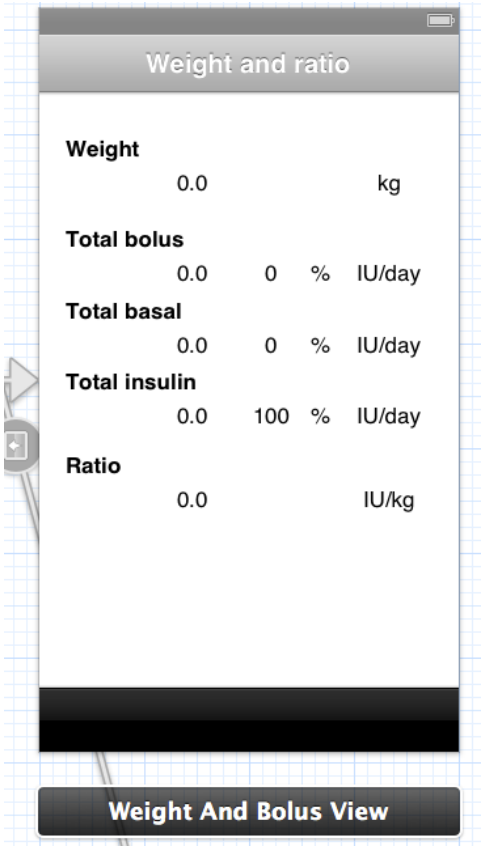
10.6.1 Menu

Vue offrant l'accès à toutes les vues du groupe paramètres.



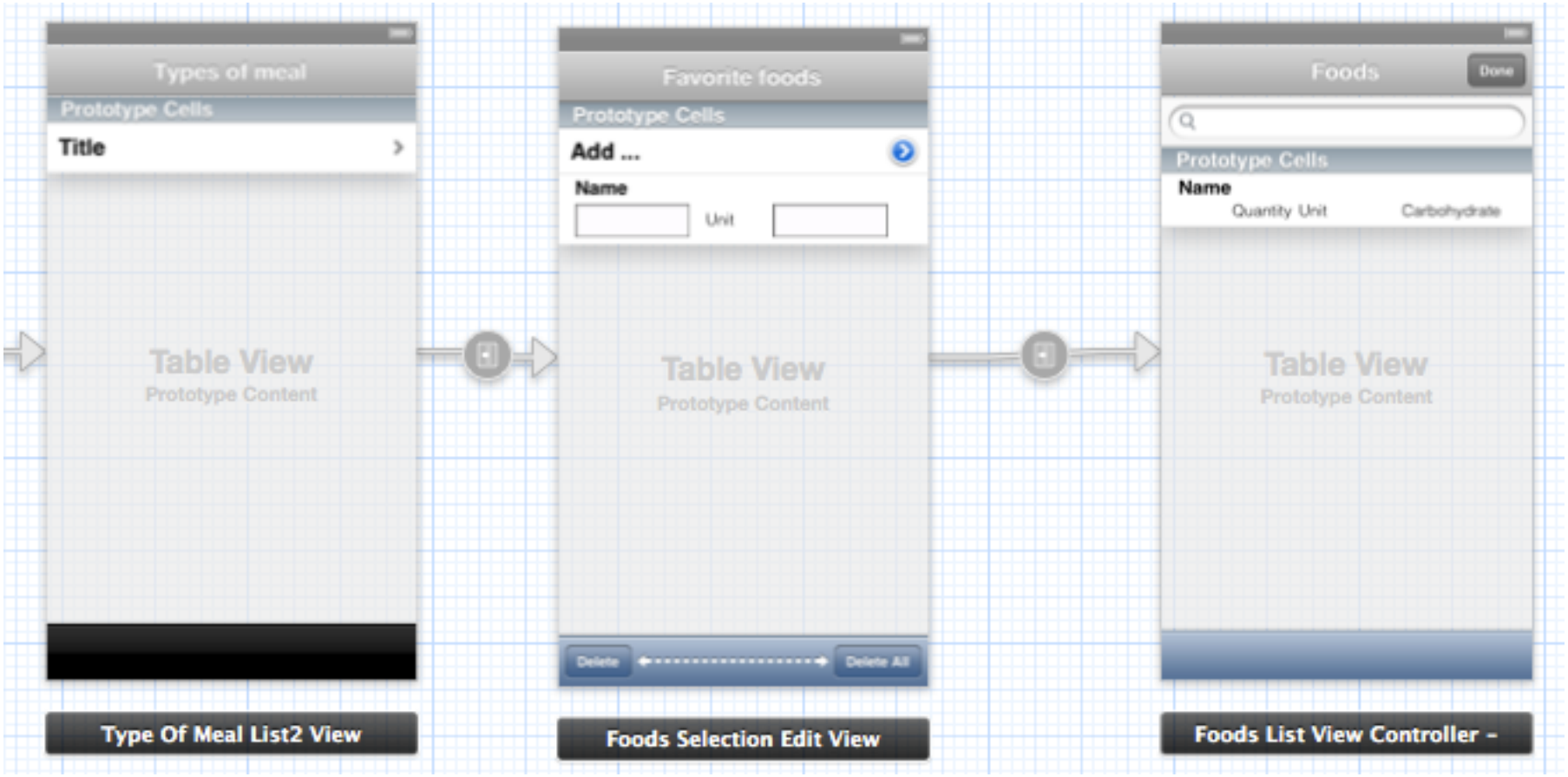
10.6.2 Poids et ratio

Cette vue offre un résumé bolus et basal pour tous les types de repas enregistré.



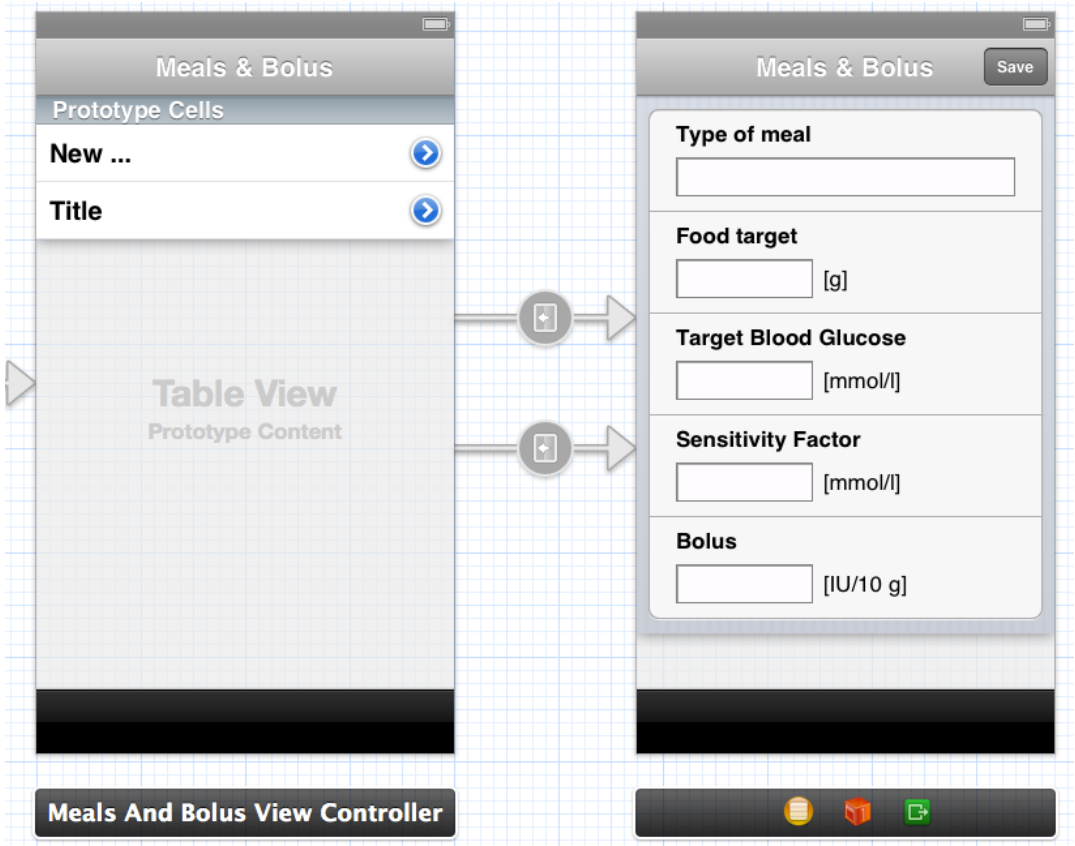
10.6.3 Aliments favoris

Ces vues permettent pour chaque repas de sélectionner ses aliments favoris. Une fois sélectionnés, il est possible de renseigner les valeurs par défaut pour chaque aliment.



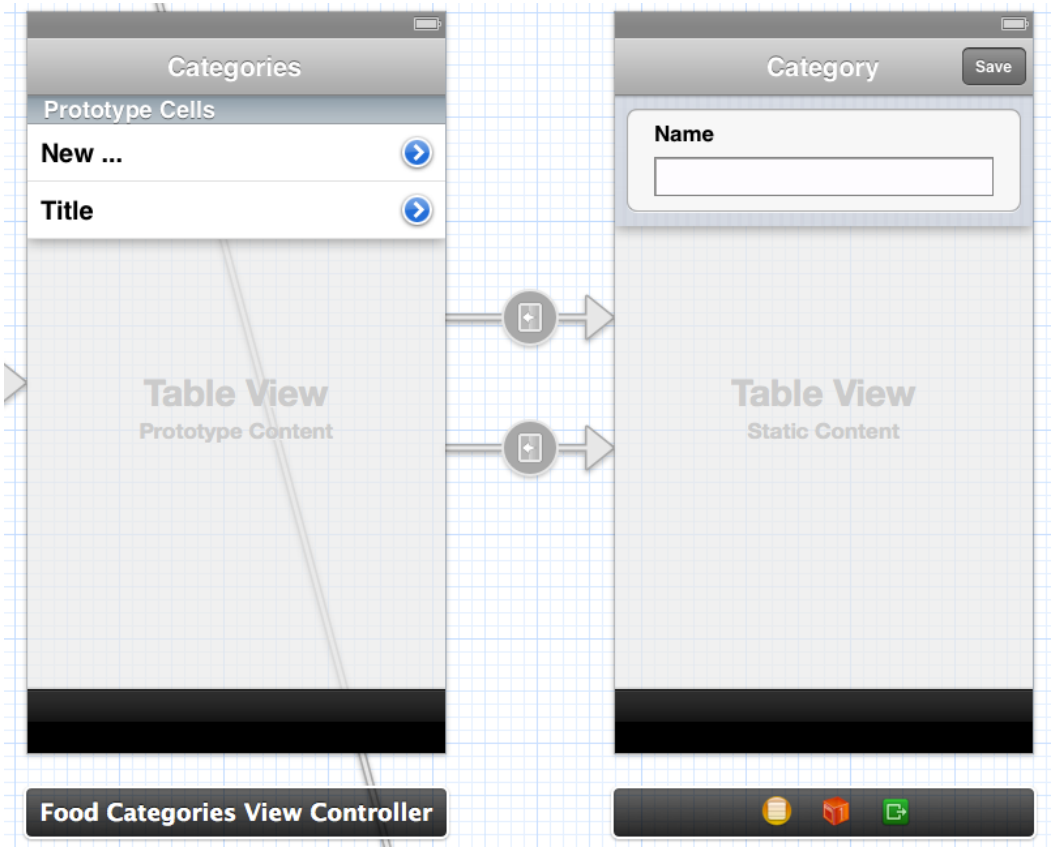
10.6.4 Repas et Bolus

Ces vues permettent la gestion des types de repas ; création, modification et suppression.



10.6.5 Catégories d'aliments

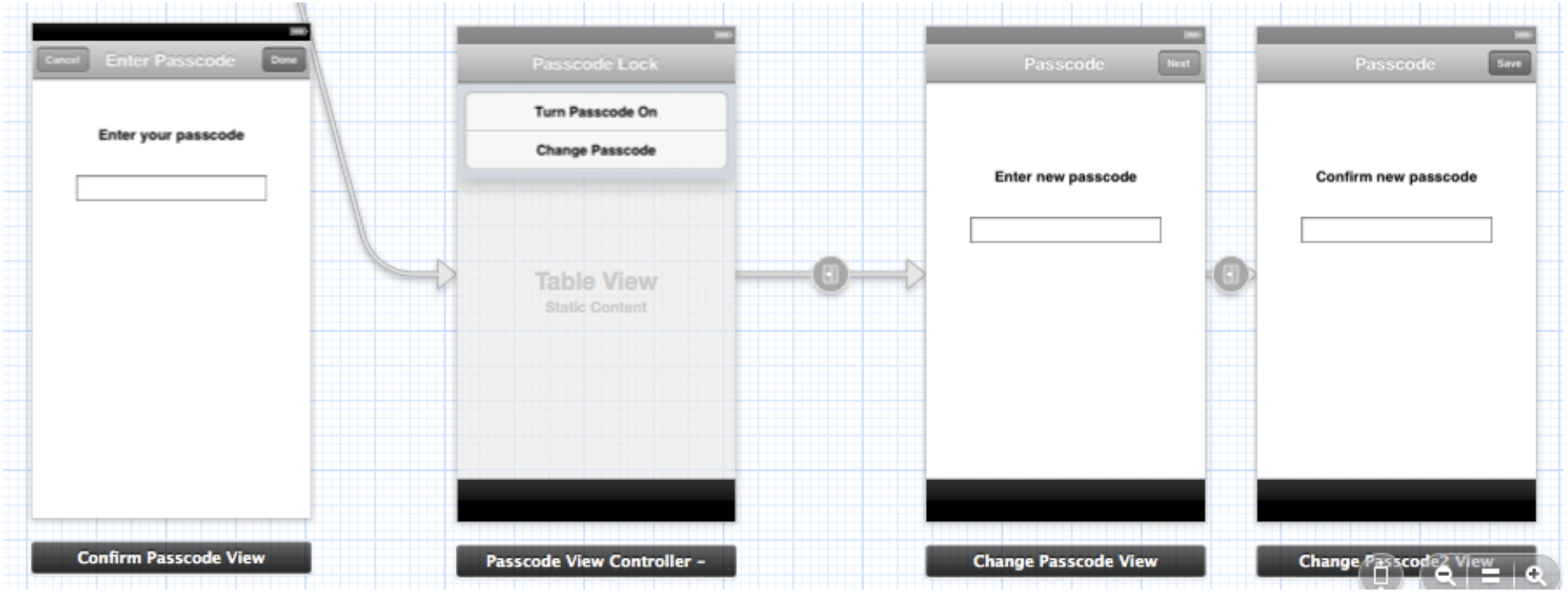
Ces vues permettent la gestion des catégories; création, modification et suppression.



10.6.6 Mot de passe

La première vue peut être appelée par n'importe quel vue pour obliger l'utilisateur à renseigner le mot de passe ; elle retourne l'information si le mot de passe renseigné est correct.

Les autres vues permettent la gestion du mot de passe ; activation, désactivation et modification.



11. Difficultés rencontrés

11.1 Tests sur des appareils

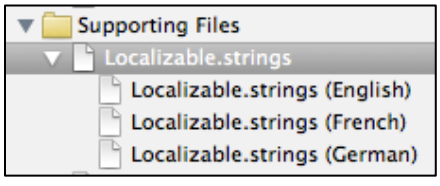
De base, Apple offre un excellent simulateur d'appareils. Néanmoins, il est toujours préférable d'utiliser un vrai appareil pour tester les gestes, la mémoire, comment l'application réagit dans un environnement réel contenant de nombreuses données et d'autres applications.

A l'inverse d'Android qui permet de lancer l'application sur un appareil uniquement en le connectant par USB, Apple demande de certifier chaque appareil et de l'enregistrer avec votre compte de développeur Apple. Il faut donc débourser un abonnement annuel de développeur et suivre une marche à suivre un peu complexe pour un débutant.

11.2 Application multilingue

Pour permettre de gérer plusieurs langues dans une application, Apple offre deux fonctionnalités. Celles-ci peuvent être combinées.

La première utilise les textes se trouvant dans le code. Par exemple, dans le code permettant l'affichage d'une pop-up d'erreur, nous écrivons le texte du titre, du message et éventuellement des boutons. Pour traduire un texte, le SDK fournit la méthode NSLocalizedString qui va automatiquement chercher une correspondance dans le fichier Localizable.string de la langue en cours. On se retrouve avec un fichier de correspondance, en texte plat, par langue.



Ces fichiers peuvent être générés automatiquement à l'aide d'une commande dans le terminal qui va parcourir les fichiers (code) pour trouver tous les NSLocalizedString. Ci-dessous l'appel de la commande pour générer un fichier dans le répertoire Anglais.

```
find . -name \*.m | xargs genstrings -o en.lproj
```

Dans le fichier généré, nous retrouvons, pour chaque traduction, le commentaire (second paramètre de la méthode NSLocalizedString) que j'utilise pour indiquer la phrase en anglais et le mot clé. Il ne nous reste plus qu'à compléter les égalités pour indiquer la traduction selon la langue.

```
/* This food's name already exist, please use another */
"alertErrorFoodNameAlreadyExist" = "Ce nom d'aliment est déjà utilisé";

/* Please select a value */
"alertErrorSelectAValueMessage" = "Veuillez sélectionner une valeur";

/* Please select foods to continue */
"alertErrorSelectFoods" = "Veuillez sélectionner des aliments pour
continuer";

/* You must accept the terms of use in order to use this software */
"alertErrorTermOfuseMessage"   =   "Vous   devez   accepter   les   conditions
d'utilisation afin de pouvoir utiliser ce logiciel";
```

La seconde fonctionnalité utilise les textes se trouvant dans les vues. Lorsque l'on dessine une vue, quand on glisse un label, le texte affiché est celui à traduire. Quand cette fonctionnalité est activée dans l'IDE XCode, il crée pour chaque vue ou StoryBoard autant de fichiers que de langues à gérer. Le fichier de langue peut être en format texte, les id des composants sont donc utilisés comme clé de traduction, ou sous format de conception (design), ce qui permet de dessiner des vues différentes avec des champs plus ou moins grands.

La première fonctionnalité est automatiquement à utiliser dès lors qu'on a du texte dans le code.

La seconde quant à elle n'est pas fonctionnelle, j'ai fait de multiples tentatives, mais dès lors que l'on change une vue après avoir indiqué qu'elle est multilingue, celle-ci ne fonctionne plus. Pour contourner ce problème, la plupart des développeurs traduisent tous les champs à la main avec la méthode NSLocalizedString. Le problème est que cela génère beaucoup de code.

Ma solution est de créer une classe mère dont toutes les vues à traduire vont hériter. Cette classe mère va parcourir tous ses enfants et réagir selon le composant pour récupérer le texte à afficher et l'envoyer dans NSLocalizedString. Si une traduction existe, elle est affichée, sinon NSLocalizedString renvoie le message entré en paramètre.

Ci-dessous l'appel de la traduction selon le composant que j'ai écrit dans ma méthode « translateView »:

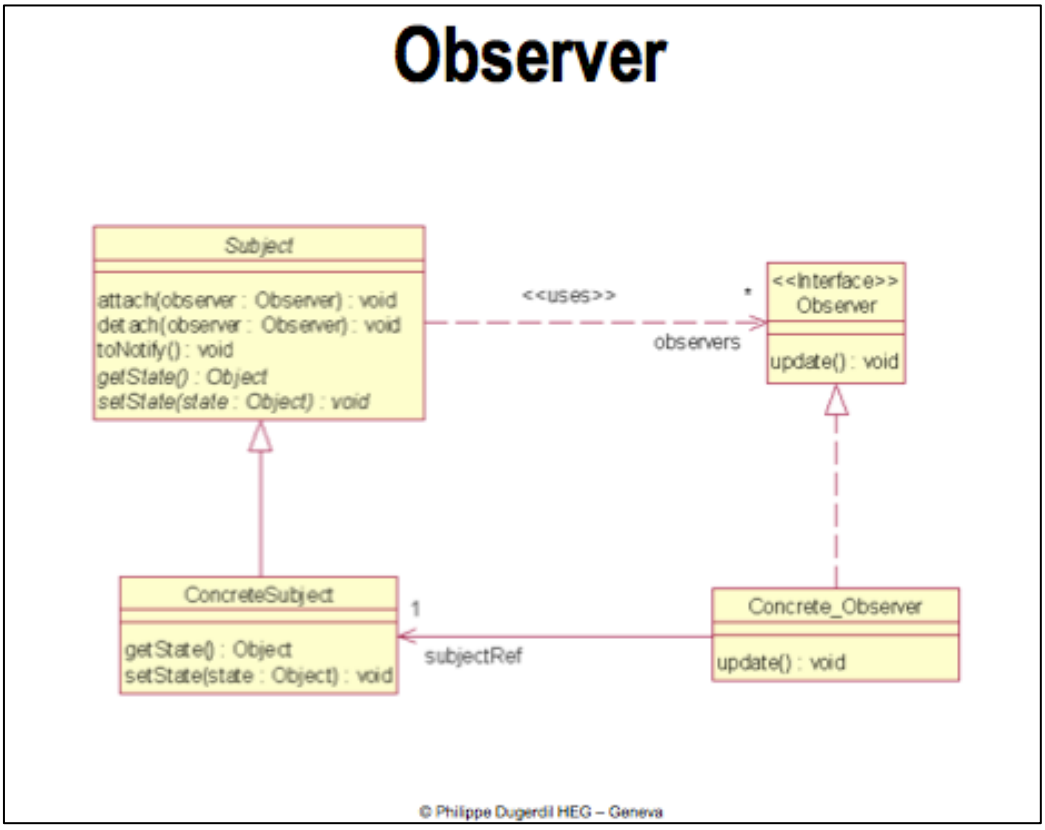
```
if ([view isKindOfClass:[UILabel class]]) {
   NSLog(NSLocalizedString(((UILabel *)view).text, nil));
   ((UILabel *)view).text = NSLocalizedString(((UILabel *)view).text, nil);

}else if ([view isKindOfClass:[UIBarButtonItem class]])
{
   NSLog(NSLocalizedString(((UIBarButtonItem *)view).title, nil));
   ((UIBarButtonItem *)view).title = NSLocalizedString(((UIBarButtonItem
*)view).title, nil);
```

Ma solution permet donc de centraliser toutes les traductions dans les fichiers Localizable.string, ce qui facilite la maintenabilité.

12. Génie Logiciel - Design Pattern utilisé

Le pattern Observer, permet à des classes (observateurs) d'être informées de chaque mise à jour d'attributs d'une classe qui doit être observable.



Dans iOS, ce pattern est automatiquement intégré pour chaque attribut d'une classe. Il est donc très aisé de s'enregistrer en tant qu'observateur d'un attribut et donc être

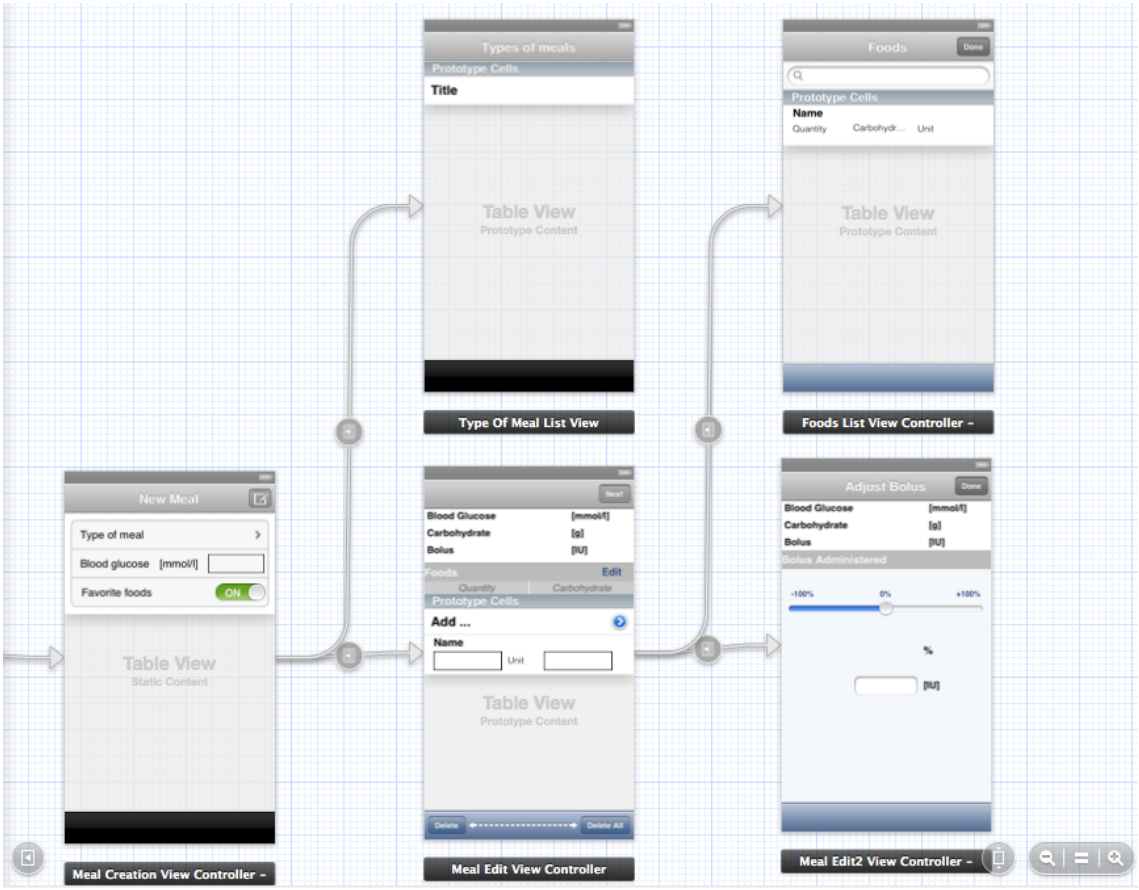
informé à chaque changement de valeur. Ceci permet de réagir à ce changement en mettant à jour une vue ou en lançant une sauvegarde, etc.

Dans cette application, ce pattern est utilisé lors de la sélection des aliments. Chaque cellule (aliment) possède un champ de saisie pour la quantité et un pour les glucides. En dessus de cette liste d'aliments se trouvent des valeurs calculées en se basant sur les quantités et les glucides. A chaque changement, les calculs sont rejoués. J'utilise donc le pattern Observer. Vous trouverez plus de détails dans le chapitre « Gestion des observateurs ».

13. Bonnes pratiques utilisées

13.1 StoryBoard

Depuis XCode 4, il est possible de créer toutes vos vues ou un groupe de vues dans un même fichier. Ceci permet d'avoir une vue globale, de lier facilement des objets ou des actions à un code, et son principal point fort est de permettre de créer les différents enchaînements de vues. Exemple ; un clic sur un bouton affiche une vue tandis qu'un clic sur une ligne d'une liste en affiche une autre.



Dès lors, les transitions sont automatiques. Pour pouvoir intercepter la transition pour par exemple envoyer des données à la prochaine vue, il suffit de surcharger la méthode prepareForSegue et de réagir selon l'identifiant. Ci-dessous un exemple d'envoi de l'élément sélectionné.

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    if ([[segue identifier] isEqualToString:@"DetailSegue"]) {

        // Fetch Item
        Aliment *item = [self.foods objectAtIndex:[indexPath row]];

        // Send item to GCFoodsDetailViewController
        [[segue destinationViewController] setFood:item];
    }
}
```

13.2 Fichiers utils

Pour améliorer la lisibilité, mais surtout la maintenabilité, j'ai utilisé des fichiers utilitaires contenant du code générique réutilisable.

Par exemple, j'ai créé une méthode permettant de retirer tous les accesseurs (icône à droite de chaque ligne) d'une liste. L'un des exemples les plus utilisés est la création d'une liste à cocher. La « coche» qui apparaît à la sélection d'une ligne est un accesseur nommé UITableViewCellAccessoryCheckmark. Ma méthode permet de remettre à zéro les coches avant d'afficher la nouvelle sélection.

Ci-dessous le code de cette méthode :

```
+ (void) uncheckAllCellsIn:(UITableView *)tableView
From:(UITableViewController *) delegate
{
    for (int i=0; i<[tableView numberOfSections];i++) {
        for (int j=0; j<[tableView numberOfRowsInSection:i];j++) {
            UITableViewCell *cell = [delegate tableView:tableView
cellForRowAtIndexPath:[NSIndexPath indexPathForItem:j inSection:i]];

            cell.accessoryType = UITableViewCellAccessoryNone;
        }
    }
}
```

13.3 Base de données

Dans cette application nous avons décidé de stocker les données en local. Deux choix sont disponibles : utiliser directement SQLite et donc écrire des requêtes SQL ou utiliser Core Data.

Plus récent, Core Data permet de concevoir le modèle à l'aide d'un designer puis d'extraire automatiquement des classes correspondant aux différentes entités (tables). Vous pouvez donc gérer les données uniquement en Objective C sans ne jamais avoir besoin de coder en SQL, le SDK le fait pour vous. Pour cette raison, Core Data fut mon choix.

13.4 Paramètres

Pour sauvegarder des données telles que des paramètres de l'application ou des valeurs techniques, deux méthodes existent. La plus fréquemment utilisée est UserDefaults qui permet un accès facile à des données typées et consomme très peu de lignes de code. L'autre méthode est d'utiliser la base de données ; dans le cas où on en utilise déjà une cela ne prend pas beaucoup plus de lignes de code ; le défaut est que la valeur ne sera pas typée. Je m'explique, si vous voulez créer une table contenant un nom de paramètre et une valeur vous ne pourrez choisir qu'un seul type pour la valeur.

Je me suis donc orienté vers le UserDefaults. Néanmoins, il me restait une problématique à résoudre. L'utilisation du UserDefaults, nécessite de connaître le nom du paramètre et son type. Je ne suis pas fan d'avoir des strings dans le code il est très facile de faire une erreur de frappe et la maintenance est difficile. Quant au type, pour setter une valeur on peut appeler setBool :ForKey ou setFloat :ForKey, etc.. Et pareil pour le getter boolForKey ou floatForKey. J'ai donc choisi de me créer une classe statique qui contient des getter et setter. Je délègue donc la responsabilité de connaître le string exact et le type correspondant pour chaque paramètre. Ce qui allège encore plus mon code car je n'ai plus qu'à appeler une méthode.

Exemple pour un booléen :

```
+(NSUserDefaults *)getUserDefaults{
    return [NSUserDefaults standardUserDefaults];
}
+(void)saveUserDefaults{
    [[self getUserDefaults] synchronize];
}
+(void)setBlnFoodDataFirstPush:(BOOL)blnFoodDataFirstPush{
    [[self getUserDefaults] setBool:blnFoodDataFirstPush
forKey:@"blnFoodDataFirstPush"];
    [self saveUserDefaults];
}
+(BOOL)blnFoodDataFirstPush{
    return [[self getUserDefaults] boolForKey:@"blnFoodDataFirstPush"];
}
```

13.5 Gestion des observateurs

Dans mon application, plus précisément dans la sélection des aliments d'un nouveau repas, je me retrouve avec une liste contenant des champs éditables que la vue doit observer. A chaque changement, les calculs du total de glucides et du bolus sont lancés.

Une liste ou tableView fonctionne de telle sorte à utiliser le minimum de mémoire possible. Elle va donc demander à l'affichage de chaque nouvelle ligne de fournir une instance de cellule (ligne) ; nous devons fournir celle-ci en récupérant des instances qui ne sont plus visibles à l'aide de la méthode dequeueReusableCellWithIdentifier. C'est donc dans cette méthode que nous enregistrons notre vue en tant qu'observateur.

Le problème maintenant est de trouver quand retirer l'observateur, car quand un événement est lancé et non récupéré cela déclenche des alertes dans les logs et crée des bugs. Nous pouvons savoir quand une cellule est réutilisée ou instanciée, mais il n'y aucun moyen de savoir quand elle est dés allouée de la mémoire par la vue. Ma première hypothèse fut de supprimer l'observateur dans toutes les cellules quand la vue serait désallouée, mais cela n'a pas suffi car des cellules peuvent être désallouées à tout moment. C'est donc la responsabilité de la cellule d'informer la vue (liste) qu'elle va être désallouée. Dès lors, l'observation de cette cellule est supprimée.

13.6 Regroupement de clés à observer

Par défaut, il est possible d'observer chaque attribut contenant un getter et setter avec comme clé le nom de l'attribut.

Dans mon application, je me retrouve dans un cas où je souhaite observer tous les changements d'une cellule quel que soit l'attribut modifié. Pour éviter de nommer à chaque fois tous les attributs à observer, et risquer d'en oublier un, j'ai surchargé la méthode keyPathsForValuesAffectingValueForKey. Je peux donc utiliser une clé d'observation quelconque qui répondrait aux changements de tous les attributs :

```
+ (NSSet *)keyPathsForValuesAffectingValueForKey:(NSString *)key
{
    NSSet *keyPaths = [super keyPathsForValuesAffectingValueForKey:key];

    //The observable key "observableKeyForCellDidUpdate" contain key
carbohydrate and quantity
    if ([key isEqualToString:observableKeyForCellDidUpdate])
    {
        NSSet *affectingKeys = [NSSet setWithObjects:@"carbohydrate",
@"quantity", nil];
        keyPaths = [keyPaths setByAddingObjectsFromSet:affectingKeys];
    }
    return keyPaths;
}
```

13.7 Versioning des sources

Dans tout projet de développement, il est nécessaire de versioner ses sources. Ceci permet de retrouver des lignes supprimées ou de remonter dans le temps pour trouver la source d'un bug, mais surtout de sauvegarder le projet.

Parmi les serveurs de versioning les plus utilisés, je nommerai SVN et CSV. Mais ceux-ci sont payants et lourds à installer pour un projet avec un seul développeur. Apple a son propre outil de versioning nommé GIT. Il est parfaitement intégré à l'IDE XCode, c'est donc lui que j'utilise. Ceci permet de versioner, mais comme je n'utilise pas de serveur externe, il me reste la vulnérabilité du crash de mon disque dur.

J'utilise donc un autre outil d'Apple nommé Time Machine. Celui-ci permet d'effectuer des sauvegardes automatiques de votre ordinateur tout au long de la journée. Avec cet outil, je me retrouve donc avec une copie de mes sources sur un disque dur externe.

14. Conseils d'un débutant

14.1 Base de données

Utiliser Core Data plutôt que de tout faire à la main. Ceci permet d'utiliser uniquement des objets en Objective C.

Attention: Cette option se sélectionne à la création du projet.

Autre remarque: Si vous modifiez la structure d'entité ou d'objet déjà existante cela peut entrainer un planter au démarrage de votre application. Ceci est normal, car l'application ne peut pas faire le la correspondance avec le nouveau format. Pour corriger ce problème, il suffit de désinstaller l'application de l'appareil (Iphone simulateur, iPhone, IPod) et la relancer depuis XCode, ce qui installera la nouvelle version. Malheureusement, la désinstallation d'une application implique la perte de ses données.

14.2 Créer une vue

Deux contrôleurs de vue (UIViewController et UITableViewController) sont disponibles, l'un contenant une vue et l'autre une tableau (tableview).

Quand l'on choisit de créer une nouvelle vue avec des champs, on serait tenté d'utiliser le contrôleur de vue, qui nous permettrait de déposer les composant où l'on veut. Etonnamment, ce n'est pas la plus simple des solutions. La plupart du temps, c'est un contrôleur de tableau qui est utilisé. Celui-ci est à de nombreuses options qui permettent de changer son look pour qu'il ressemble plus à une vue qu'une simple liste, par exemple la vue groupée des cellules.

Je vous conseille de tester les différentes opportunités offertes par ce contrôleur, c'est comme ces petites choses qu'on ne remarque pas, mais dès qu'on les voit une fois on les voit partout.

Beaucoup d'applications se basent principalement dessus ; vous le trouverez dans vos appareils mobiles par exemple dans l'application "Réglages".

14.3 Import

Il est très intéressant pour un utilisateur de pouvoir importer des valeurs par défaut. Je vous conseille, tout comme dans mon application, d'avoir un fichier inclus dans l'application. En cas de multilingue, n'hésitez pas à en avoir autant que de langues gérées.

Pour permettre l'échange de données, ou l'import de données plus récentes que celles par défaut, il est intéressant d'offrir un import de fichiers externes. Dans mon cas, j'ai utilisé l'import de fichiers csv qui peuvent être transféré depuis les mails ou autre application stockant des fichiers.

L'autre méthode qui est recommandé par les internautes est d'utiliser le partage de fichier d'iTunes. Chaque application a un répertoire qu'elle peut rendre visible à iTunes, dès lors dans iTunes un explorateur de fichiers apparaît. Ceci permet donc de déposer ou même retirer des fichiers. Au lancement votre application peut lire le répertoire et importer les nouveaux fichiers.

14.4 Export

Pour permettre l'échange ou la sauvegarde des données, l'export est un service à considérer.

Il est possible de configurer chaque application avec laquelle vous souhaitez communiquer ; moi je vous conseille une solution plus pérenne qui est d'utiliser le Sharing. Son utilisation vous permet de générer un fichier et il déterminera seul quelles sont les applications sur le mobile pouvant le lire. Il est très utilisé pour l'envoi de mail, la sauvegarde dans des systèmes de fichiers comme DropBox, ou le réseautage (Facebook, Tweeter).

L'autre méthode qui est recommandée par les internautes est d'utiliser le partage de fichier d'iTunes, comme chaque application à son propre système de répertoires, il peut afficher les fichiers générés.

Une méthode qui peut être utilisée plutôt comme backup est la sauvegarde directe dans l'iCloud.

14.5 Versionning

Comme je l'ai déjà indiqué dans un chapitre précédent, toujours versionner vos fichiers pour permettre de revenir en arrière car le « contrôle + z » a des limites.

15. Publication du logiciel

Dans un premier temps, nous avions prévu de publier le logiciel dans l'Apple Store nous même, ce qui est une chose facile quand l'on veut distribuer gratuitement l'application. Durant le projet, le choix de vendre l'application à bas coût a été pris par mon client. En consultant Sente, nous avons compris que pour permettre à Apple de transférer les gains, nous devions remplir des documents et éventuellement créer un compte. Nous avons donc décidé de transférer cette charge à Sente qui a l'habitude d'effectuer ce genre d'opérations.

Conclusion

Lors du choix de mon mémoire, je souhaitais trouver un sujet qui pouvait allier ma facilité dans le développement et mon intérêt à produire une solution utile qui aidera des gens. Dans le marché, nous n'avons pas souvent l'occasion de produire des solutions qui ont pour premier objectif d'aider les gens, c'est pourquoi quand je suis tombé sur l'offre de mon client, je n'ai pas hésité une seule seconde. Consacrer deux mois, sans mon travail, à produire une solution permettant de faciliter la vie des diabétiques était pour moi une belle forme de bénévolat. De plus, pour ajouter du piment à ce sujet qui se trouve être très sérieux, je me suis trouvé face à un nouveau langage ; Objective C.

Ayant déjà produit une solution mobile sous forme de site web et ayant suivi des cours de développement Android, j'étais à même de concevoir les problématiques posées par un logiciel sur une plateforme mobile. Mes années d'expériences en Action Script et Java, couplées avec mes cours de développement, m'ont rapidement permis de comprendre le langage Objective C. J'ai été agréablement surpris par l'efficacité de ce langage ainsi que celle de l'IDE XCode. Mon seul regret aura été de ne pas avoir un meilleur débuggeur, du niveau de celui d'Eclipse et NetBeans.

Ne connaissant pas de diabétiques, je ne pouvais pas concevoir leurs difficultés au jour le jour. Ce mémoire m'a permis de les découvrir et je suis reconnaissant envers mon client qui m'a offert l'opportunité de réaliser cette application d'une grande importance pour lui.

Après mon rendu de mémoire, je livrerai une dernière version à mon client et ferai un passage de connaissance avec l'entreprise Sente.

Je suis fier de pouvoir dire que notre application sera publiée d'ici la fin de l'année, en espérant qu'elle puisse aider un maximum de personnes.

Bibliographie

Site de l'application, http://www.glucalc.ch

Article « Android vs iOS vs Windows vs RIM au Q2 2013 – (Gartner) », http://www.eco-conscient.com/art-547-vente-de-mobiles-android-croit-de-600-au-premier-trimestre-2010.html

Image Pompe à insuline, http://www.doctissimo.fr/html/dossiers/diabete/articles/14668-nouveaux-progres-pompes-insuline.htm