Rapport de fin d'études

Développeur Informatique

2020 – 2022

Projet en Entreprise :

Scan&Shop

budgetbox

Tuteur : M. JUE Kevin

Pilotes : M. MACHU Brice

M. LIERVILLE Fabien

Auteur du document :

M. FICHOU Kevin

ÉCOLE SUPÉRIEURE DE L'ALTERNANCE

FICHE DE CONFIDENTIALITE
DES RAPPORTS, MEMOIRES, THESES ET SOUTENANCES PROFESSIONNELS

Formation/qualification préparée : Développeur Informatique

Nom-Prénom du stagiaire : FICHOU KEVIN

Titre du dossier professionnel : .....

Date de la soutenance : 06/09/2022 - 9h30

Nom de l'entreprise : BudgetBox

Nom et qualité du représentant de l'entreprise : M. JUE Kevin, Product-Owner

Noms, entreprises et fonctions des membres de jury : .....

Nom-Prénom	Entreprise	Fonction

Mode de diffusion autorisé
(Cocher la case correspondante)

☐ **Diffusion libre**
Le dossier est conservé en archives au CESI, il peut être librement consulté et reproduit. Il peut être utilisé par les destinataires, les études peuvent faire l'objet de publication....

☑ **Diffusion limitée au CESI**
Les membres du jury rendent leur exemplaire au stagiaire à la fin de la soutenance. Le stagiaire est responsable de cette restitution. Un exemplaire est conservé en archives au CESI. Le dossier peut être consulté pour exemple ou illustration par les stagiaires des promotions suivantes mais il ne peut être ni sorti du CESI, ni reproduit, sauf autorisation expresse de l'auteur et de son entreprise. La mention « Diffusion limitée au CESI, reproduction interdite » doit figurer sur la page de garde.

☐ **Diffusion interdite**
Les membres du jury rendent leur exemplaire au stagiaire à la fin de la soutenance. Le stagiaire est responsable de cette restitution. Un exemplaire est conservé au CESI, à titre de preuve dans le dossier pédagogique du stagiaire. Le dossier ne peut être ni consulté, ni sorti du CESI, ni reproduit, sauf autorisation expresse de l'auteur et de son entreprise. La mention « Diffusion et reproduction interdites » doit figurer sur la page de garde.

Signatures :

Pour l'entreprise	Le stagiaire	Le CESI
LYES KIMOUCHE	FICHOU Kevin	

Remerciements **5**
In Memoriam **6**
Introduction **7**
I – Budgetbox , l'entreprise et ses équipes **8**
L'entreprise Budgetbox 8
1.1 Un bref historique 8
1.2 Son Chiffre d'Affaires 8
1.3 Ses locaux 8
1.4 Sa clientèle 9
Les produits et solutions de Budgetbox 9
2.1 Scaneo & Scaneo-Cloud 9
2.2 Son business model 10
3) Le pôle Connected-Shopper 10
3.1 L'équipe 10
3.2 et moi dans tout ça ? 10
4) Les autres pôles du site rouennais 11
4.1 L'Infra 11
4.2 Media-data 11
4.3 La Recette 12
II- Le projet Scan&Shop, contexte et objectifs **13**
1) Présentations du projet 13
1.1 Scan&Shop, mais qu'est-ce donc ? 13
1.2 Objectifs 13
1.3 Le calendrier 14
1.4 Le cahier des charges 14
1.5 Les difficultés annoncées au lancement du projet 14
2) Déroulement du projet 15
2.1 Trame principale 15
2.2 SCRUM process 17
2.3 L'apprentissage « sur le tas » 18
2.4 Mes relations avec l'équipe 19
3) L'environnement de développement 19
3.1 Xcode 20
3.2 Le matériel 20
3.3 SWIFT, le jeune langage d'Apple 20

a. Définition 21
b. Caractéristiques 21
c Quels sont ses avantages ? 21
d Quelles sont ses limites ? 22
e Comparativement à Objective-C 22
f Représente-t-il l'avenir du développement iOS ? 23
4) La chasse aux files d'attente en caisse 23
4.1 Principe 23
5) Les maquettes graphiques 24
5.1 De mes ébauches… 24
5.2… Aux maquettes de Paris 24
III La réalisation du projet 25
1) La mise en place de l'architecture. 25
2) Récupération et décompression 25
2.1 La décompression 26
2.2 L'extraction 27
2.3 Le parse des données 27
3) La Base de Données 28
3.1. Core Data 28
3.2. Quelles différences avec les NSUserDefaults. 29
4)- Une animation au lancement. 30
4.1 Reconfiguration du storyboard 31
4.2 La configuration des éléments 31
4.3 L'animation 31
5) Le menu principal 32
5.1- La mise en place du menu principal 32
5.2- Choisir le magasin et entrer sa carte de fidélité 34
a. Le choix du magasin 35
b. La vue de sélection de magasin 35
c. La cellule d'ajout de carte de fidélité 36
d. La cellule du code barre 36
5.3. Choisissons notre magasin 37
a. Mes Interfaces 37
b. Mes Dépôts 38
c. Le contrôleur de vue mon Magasin 38

d. La collection de vues de sélection de la distance....38
e. La collection de vues de magasins....39
6). La collection de vues des coupons de réduction....40
6.1. La cellule de coupon....41
6.2. Le modèle de coupon....41
6.3. La vue de coupon....41
6.4. La cellule de coupon déconnecté....41
6.5. La vue de coupon déconnecté....42
7). La collection de vues des recettes et les cellules de types de recettes....42
7.1. La cellule de type de recette....42
7.2. La vue de sorte de recette....43
7.3. La collection de vues de recettes....43
a. Le Modèle de la vue de recette....43
b. La cellule de recette....43
c. La vue de recette....44
7.4. Fonctionnement du filtrage d'une recette....44
8). Le panier, au cœur de *Scan&Shop*....45
8.1. Le contrôleur de panier....45
8.2. La collection de vue du panier....45
8.3. Le modèle d'item du panier....46
8.4. Les données d'item du panier....47
9). Les tests unitaires....48
Conclusion....49
Table des illustrations....51
Annexe 1....52
Annexe 2....53
Annexe 3....54
Annexe 4....55
Annexe 5 :....56

Remerciements

Avant d'aller plus loin dans ce rapport de fin d'études, je souhaiterais tout d'abord remercier M.JUE Kevin, mon tuteur, pour m'avoir recruté et formé au métier de développeur iOS et aidé à faire mes premiers pas dans le monde du développement informatique.

Je remercie tous les membres du jury pour m'offrir du temps dédier à la lecture de ce rapport ainsi qu'à son évaluation. J'espère que vous apprécierez votre lecture, autant que moi j'en eu à rédiger ce rapport.

Je souhaite également remercier MM. MACHU Brice et LIERVILLE Fabien, tous deux pilotes de la formation, pour avoir été à mon écoute et avoir toujours cherché à m'aider sur la voie de l'apprentissage et les meilleures conditions de travail au sein du CESI. Et plus spécifiquement à M. MACHU Brice qui m'a permis de rejoindre la formation de Développeur Informatique millésime 2020.

Je remercie M. LEGRAS Pierre, collègue et voisin de bureau dans les locaux de Rouen pour tous ses bons et précieux conseils sur le développement iOS qui vinrent compléter à merveille tout ce que m'apportait mon tuteur.

Un grand remerciement à M. KETTANI HALABI Zakaria pour ses précieux conseils et sa grande expérience concernant l'architecture du code, ainsi que pour tous nos débats philosophiques et théologiques qui ont parsemé ces deux années.

Mais je remercie aussi M. RICHARD Arthur, qui, par sa fraîcheur et ses connaissances m'a beaucoup apporté et aidé sur la voie de l'apprentissage, et qui n'a pas hésité à me prêter ses bras pour porter des cartons.

Plus généralement, je remercie tous les membres des équipes de Budgetbox pour tout ce qu'ils ont pu m'apporter, tant techniquement, qu'humainement, alors que je découvrais le monde civil. Également pour leurs retours sur le présent rapport et les précieux conseils qu'ils m'ont donnés.

Un grand merci à M. BILLAUX Matthieu, ancien compagnon d'armes, versé dans la cyberdéfense et qui m'a aidé à faire mes premiers pas dans ce monde très particulier qu'est l'informatique avant mon entrée au CESI.

Je remercie grandement les véritables moteurs de mon apprentissage, mon épouse, Mme FICHOU Laura, ainsi que mes enfants Louise et Antoine, qui sont véritablement ma raison d'être et de me battre.

In Memoriam

Ce rapport, ainsi que mes deux années passées sont dédiés à ma mère, Isabelle MAUGE, chef de projet WEB à la mairie de Dieppe, qui nous a quitté le 12 juillet 2021.

Jusqu'au bout, elle fut une mère et une grand-mère dévouée à ses enfants et petits-enfants. Sans elle, rien de tout ceci n'aurait été possible pour moi. Elle fut la première à me parler du CESI et des opportunités de reconversions qu'il offrait. Mais elle fut également un véritable soutien dans ma reconversion professionnelle et mon passage du monde militaire au monde civil.

Je te remercie m'man et, comme gens de mer que nous sommes, je te souhaite bon vent et bonne mer.

Introduction

Alors que viennent de s'achever ces deux années de reconversion professionnelle, je viens, au travers de ce rapport, vous présenter le projet réalisé au sein de l'entreprise Budgetbox : Scan&Shop.

Après une première expérience professionnelle, de dix ans, passée dans les Forces Armées, comme officier-marinier spécialisé dans la détection surface en sous-marin nucléaire d'attaque et frégates, je décidais de changer tout à fait de voie professionnelle.

Souhaitant au préalable rejoindre les rangs de la cyberdéfense au sein de la DGSE, il m'apparut rapidement qu'il me manquait les bases nécessaires à ce métier. Accepté au CESI pour suivre la formation de Développeur Informatique, millésime 2020-2022, je débutais cette reconversion sans la moindre connaissance en informatique. Ce fut dans les locaux de Budgetbox, à Saint-Etienne-du-Rouvray, que j'appris la programmation logicielle.

Arrivé dans les équipes de développement de l'entreprise, on me présenta plus précisément les missions de l'entreprise, ses produits et services auprès de ses clients. Puis vinrent mes premiers pas dans la programmation informatique, avec notamment des interventions dans l'application de démonstration, puis les produits iPhone. Destiné à devenir le prochain expert des technologies Apple de l'entreprise, succédant à mon tuteur. Après quelques mois de découvertes et d'un premier travail, débuta pleinement le projet autour duquel devait s'articuler toute ma reconversion : Scan&Shop.

Véritable vitrine du savoir-faire de l'entreprise et de ses équipes, l'application de démonstration était utilisée lors des forums et conventions par les équipes commerciales pour démarcher de nouveaux clients auprès de la grande distribution. Cependant, cette application ne répond plus aux attentes et ses graphismes ne sont plus en phase avec ce qu'il se fait de nos jours. De même, codée en Objective-C car utilisé uniquement sur iPhone, ce langage est destiné à ne plus être suivi par Apple dans les années qui viennent pour être remplacé par SWIFT. Dès lors, le projet Scan&Shop se doit de pallier ces manquements, tout en étant l'occasion d'apporter des améliorations, de la nouveauté et de la fraîcheur dans ses graphismes. L'entreprise disposant d'un pôle de design dans ses locaux parisiens, celui-ci fut chargé de nous remettre des maquettes graphiques, formant, avec les spécifications techniques, la base sur laquelle m'appuyer pour créer, depuis un nouveau projet, l'application de démonstration de nouvelle génération.

Dans le présent rapport, et au travers de plusieurs grands thèmes, je vous présenterai la solution que j'ai apporté, ainsi que les différentes étapes lors de sa réalisation. Ainsi, dans un premier temps, je vous présenterai l'entreprise Budgetbox, ses équipes, son modèle d'affaire, ses produits. Puis, j'introduirais le contexte dans lequel j'eus à développer cette application de démonstration, le langage utilisé, le matériel à disposition. Enfin, dans une troisième partie, je vous apporterais le déroulé du développement, étape par étape, de ladite application, les difficultés auxquelles je fis face, mais également les solutions originales trouvées pour pallier à celles-ci. Aussi, permettez-moi de vous introduire l'entreprise Budgetbox.

I – Budgetbox, l'entreprise et ses équipes

L'entreprise Budgetbox

1.1 Un bref historique

L'histoire de l'entreprise Budgetbox débute en 2007 et souhaite accompagner marques et enseignes de la grande distribution dans leur stratégie de *retail-media*. Pour se faire, l'entreprise va proposer un logiciel qui doit optimiser les achats de la clientèle de ces enseignes.

Figure 1: Logo de l'entreprise

Créée par messieurs Pierre LEBEL et Gautier DHAUSSY, dans le cadre d'un programme universitaire « Projet Entreprendre », la jeune société va tourner son concept autour du principe suivant : Apporter de la valeur ajoutée pour le consommateur, ce dernier recevant en temps réel des informations personnalisées, tandis qu'il déambule dans le supermarché.

Ce concept fondateur va permettre à l'entreprise de s'implémenter durablement et gagner les enseignes de la grande distribution, avec l'apparition des scannettes dans leurs magasins, puis, plus tard, des solutions mobiles pour smartphones et iPhones.

La société fut rachetée en 2020 par SOGEC, elle-même filiale du groupe La Poste. Déjà actionnaire depuis 2012, ce rachat vient renforcer SOGEC dans son offre auprès de la Grande Distribution, permettant de couvrir un ensemble de mécaniques d'activation d'achats (les bons de réductions, offres de remboursement différées, etc.) sur l'ensemble du parcours de courses.

1.2 Son Chiffre d'Affaires

Pour des raisons de confidentialité, le chiffre d'affaires annuel de l'entreprise ne peut être communiqué à toute personne ne faisant pas partie de l'entreprise.

Je vous remercie pour votre compréhension.

1.3 Ses locaux

Budgetbox dispose actuellement de locaux à Rouen et à Paris.

Les locaux de Rouen, anciennement siège de la société, se voient abriter les équipes de développements, recette et infrastructure, avec une partie décisionnelle et commerciale. C'est ici que nous retrouvons les équipes MEDIA, CONNECTED-SHOPPER, INFRASTRUCTURE et RECETTE. Lesdites équipes ayant à charge du bon déroulement de créations, tests et de déploiement des solutions développées.

Tandis qu'à Paris, l'accent est plutôt porté sur la partie commerciale, avec la recherche de nouveaux clients, ainsi que la fidélisation des clients existants. Le studio de design se voit en charge des créations des coupons de réduction, en adéquation avec les clients, pour les solutions dématérialisées embarquées dans les scannettes et les applications Android et Apple.

1.4 Sa clientèle

La clientèle cible de Budgetbox est principalement constituée de deux types :

- La Grande Distribution
- Les Grandes Enseignes

Auprès des Grandes Enseignes, Budgetbox propose un service de distribution des promotions. En effet, profitant de sa place avantageuse sur le marché de la distribution des coupons de réductions numériques, Budgetbox travaille sur les souhaits de ses clients en créant lesdits coupons et de les distribuer au travers de ses produits numériques déployés en magasin, telles les scannettes ou encore les applications pour smartphone.

Facturant ses solutions personnalisées à chacun de ses clients, Budgetbox est également rétribuée pour chaque coupon numérique affiché à l'écran de l'utilisateur, et plus encore lors de son utilisation. Également présente avec des solutions pour le *drive*, l'entreprise propose un service de création de site internet dédié cet autre parcours de courses.

Dans le cas de la Grande Distribution, Budgetbox déploie et maintien des solutions numériques, via des murs de scannettes en libre-service dans les magasins, comme les supermarchés, ou bien des solutions développées et personnalisées à chaque enseigne de la Grande Distribution.

Les produits et solutions de Budgetbox

2.1 Scaneo & Scaneo-cloud

Les solutions *Scaneo* et *Scaneo-cloud* sont les principales solutions numériques que propose l'entreprise Budgetbox à ses clients.

La première consiste en le déploiement d'une infrastructure en local, une par magasin, comprenant un serveur, un ou plusieurs murs de scannettes et les appareils afférents. La maintenance applicative est assurée par les équipes de Budgetbox. Toutefois, les évolutions ne peuvent être réalisées que sur demande expresse du client, puisque le matériel et la solution étant en configuration locale, les équipes de développement ne peuvent agir en autonomie.

De plus, si cette solution permet à un client d'enregistrer sa carte de fidélité dans un magasin proche de son domicile, il est dans l'obligation, lorsqu'il se rend dans un autre magasin d'une enseigne identique, de s'enregistrer à nouveau, du fait de la configuration locale des informations en base de données.

À la différence de *Scaneo-Cloud*, qui propose un service équivalent, mais dont l'hébergement est réalisé par Budgetbox. Dès lors, les équipes de développement peuvent aisément apporter améliorations, maintenances correctives et applicatives.

Dorénavant, notre même usager peut se rendre dans n'importe quel magasin d'une même enseigne, partout sur le territoire, les informations contenues en base de données étant centralisées et partagées, il ne lui est plus nécessaire de s'enregistrer à nouveau.

2.2 Son business model

Commercialisant une solution permettant de scanner soi-même ses articles auprès des enseignes de la Grande Distribution, Budgetbox propose donc deux modèles de sa solution :

- Un hébergement au sein d'un centre de données (Data Center)
- Un hébergement en espace ouvert (Cloud)

Avec la première solution, l'enseigne héberge la solution et paie une licence d'exploitation, complétée par un abonnement pour la tenue de la maintenance évolutive et corrective de la solution déployée.

Tandis qu'avec la seconde solution, c'est l'entreprise Budgetbox qui vient héberger la solution, facturant à l'enseigne un abonnement comprenant hébergement, exploitation, maintenance évolutive et corrective.

L'entreprise complète ces solutions en fournissant la possibilité auprès des annonceurs de commercialiser des campagnes publicitaires et leur diffusion auprès des acteurs de la Grande Distribution via ses différents canaux dématérialisés.

3) Le pôle Connected-Shopper

3.1 L'équipe

Le pôle *Connected-Shopper* est décomposé en deux équipes plus petites : Scaneo et Scaneo-Cloud. Composé de quatorze membres, développeurs, Product Owner et Prox Product Owner, elle a sa charge le maintien des solutions liées aux scannettes et la gestion de bases de données déployées chez les clients de Budgetbox.

3.2 et moi dans tout ça ?

D'abord intégré dans les rangs de l'équipe *Connected-Shopper*, mes premières tâches furent consacrées à la prise en main des outils de développement et à me familiariser avec mon nouvel environnement professionnel. Au travers du maintien à niveau de l'application de démonstration, j'eus à découvrir le codage au travers du langage d'Apple Objective-C.

Au bout de quinze mois, je fus reversé dans le pôle *Scaneo-Cloud*, afin de pouvoir simplifier les échanges avec le *Product Owner* du projet de développement de la nouvelle application de démonstration.

4) Les autres pôles du site rouennais

Les équipes de Budgetbox à Rouen ne sont pas uniquement constituées des membres de *Connected-Shopper*. Bien plus nombreux, les bureaux normands comptent bien d'autres collaborateurs aux missions variées.

4.1 L'Infra

L'équipe Infra est constituée de cinq membres dont les rôles sont multiples. À eux la charge de veiller sur la bonne tenue des réseaux, notamment ceux reliant l'infrastructure de l'entreprise à celles de ses clients, via des tunnels VPN. À ce titre, ils sont le support pour lesdits clients, au niveau des serveurs, et interviennent, à distance, en cas de problème.

Cependant, ils occupent également un rôle d'accueil, lorsqu'un nouveau collaborateur intègre les rangs de l'entreprise, il leur revient la tâche de préparer et fournir le matériel nécessaire à notre nouveau collègue, afin que celui-ci puisse travailler dans les meilleures conditions.

Enfin, ces hommes gèrent le montage de tunnel VPN interne, afin que les équipes de développement et de test puissent assurer leur service en tous lieux. De plus, une partie d'entre eux ont à charge de veiller à la bonne tenue de la cybersécurité au sein de l'entreprise, veillant à ce que toutes les mises à jour soient faites, entre autres choses.

4.2 Media-data

Le pôle Media-Data se subdivise en deux parties : la partie Media et la partie Data.

L'objectif de ce pôle est de permettre la maintenance et l'utilisation d'un logiciel développé en interne et à destination de la clientèle de Budgetbox, *Evercampaign*.

En effet, cette application Web sert à la création et au suivi des campagnes de promotions des marques, offrant également une capacité d'analyse, ainsi que des comptes-rendus rédigés sous forme de rapports et à destination desdites marques afin qu'elles puissent en interpréter les résultats.

Si l'équipe Media gère la partie de maintenance et l'amélioration de l'application Web, l'équipe Data est plus en contact avec les clients, car elle va créer les rapports, effectuer une première analyse, à partir des données des parcours achats clients, puis rendre ses conclusions. Ce travail est mené avec l'application fournie par Media. Dès lors, Data va pouvoir assurer la bonne gestion des coupons de réduction dématérialisés durant les campagnes promotionnelles lancées depuis *Evercampaign*.

Si, via les bureaux de Paris, un service commercial de Budgetbox vient démarcher des clients, la place de chef de file dans le domaine de la promotion, de la conception de coupons de réduction dématérialisés et de leur diffusion, offre à l'entreprise la possibilité d'être diligenté par ses clients directement et à leur propre initiative.

4.3 La Recette

Le pôle Recette est le dernier maillon de la chaîne de production. En effet, les membres de cette équipe interviennent au terme du processus de codage par la rédaction de cahiers de tests et leur exécution.

Leur but est avant tout de satisfaire le client, en livrant à ce dernier des produits de grande qualité, conformes à ses attentes et testés en amont, notamment en vérifiant qu'aucune anomalie ne s'y trouve, ou bien encore que toutes les fonctionnalités attendues soient présentes. Le tout, dans le respect des délais de livraison.

Cependant, forts de cette expérience des tests, les membres de cette équipe tiennent à jour une base de connaissances de tests, servant à la non-régression des solutions estampillées Budgetbox. De plus, en lien avec le client, l'équipe va apporter une vision plus fonctionnelle et moins technique qui puisse parler avec ledit client.

Afin de s'améliorer et d'assurer un plus grand nombre de tests, ces derniers sont en cours d'automatisation, augmentant drastiquement le rendement de tests effectués, ainsi que leur qualité.

II- Le projet Scan&Shop, contexte et objectifs

1) Présentations du projet

1.1 Scan&Shop, mais qu'est-ce donc ?

Scan&Shop est une solution applicative pour iPhone aidant au suivi de sa session de courses et le parfait compagnon pour réussir à maîtriser ses achats et à s'affranchir des longues files d'attente lors du passage en caisse.

1.2 Objectifs

Forte d'une douzaine d'années d'expérience, l'entreprise démarche lors des salons et conventions de nouveaux marchés et clients grâce à son application de démonstration, développée pour iPhone. Cependant, cette application n'est plus représentative de ce que les équipes de développement sont capables de faire, tout en ayant un design ne correspondant plus aux standards actuels.

Souhaitant apporter de la fraîcheur et remettre à niveaux, par l'apport du retour d'expériences de ces dernières années, l'entreprise cherchait alors à mettre en chantier un nouveau projet d'application native de démonstration. Toujours prévue pour un déploiement sur iPhone, car il s'agit du type d'appareil employé par les commerciaux de Budgetbox lors des forums et rencontres.

Se questionnant sur le langage informatique à employer, Apple propose deux langages : Objective-C et SWIFT.

Le premier langage, destiné à être abandonné par la firme à la pomme et remplacé par SWIFT, présentait toutefois l'avantage d'être stable et bien connu. De plus, la première application de démonstration, ainsi que les produits actuellement déployés sont tous codés avec Objective-C, pour la partie iOS. Cependant, Apple l'ayant annoncé, l'avenir des langages de programmation de la firme doit se faire au travers de SWIFT.

Ainsi, afin de pouvoir obtenir la compétence de coder en SWIFT, de pouvoir rester en phase avec ce que fournit Apple, la direction de Budgetbox choisit alors de totalement redévelopper une application de démonstration, celle-ci devant entièrement être réalisée en usant de SWIFT, tout en reprenant les fichiers de base fournis par un dossier compressé, déjà employé pour la première application de démonstration.

1.3 Le calendrier

Figure 2: la tenue d'un calendrier est primordiale

Tandis que le projet Scan&Shop se préparait à être lancé, il apparut que, compte tenu des évènements survenus ces deux dernières années et la raréfaction des salons permettant aux commerciaux de Budgetbox de démarcher de nouveaux clients, le temps laissé au développement fut laissé libre.

De plus, conscients du fait que je ne sois pas encore un habitué du langage SWIFT et que, pour des raisons techniques évidentes, je n'aurai nullement l'occasion de l'aborder au CESI, mon tuteur, ainsi que mon manager, optèrent pour un développement plus lent, certes, mais qui devait être également efficace sur le long terme et me permettre, dans un avenir proche, de travailler sur les futures solutions iOS de l'entreprise.

1.4 Le cahier des charges

Aucun cahier des charges n'a été conservé pour la précédente application de démonstration. Par ailleurs, aucun cahier des charges ne fut rédigé pour ce projet, tout devant se baser sur l'existant, afin de gagner du temps dans le développement.

Cela étant, des spécifications fonctionnelles furent rédigées au fur et à mesure de l'avancement des travaux de programmation, afin de pouvoir cadrer le projet et vérifier, au travers des tests d'acceptance, que chaque fonctionnalité soit présente.

1.5 Les difficultés annoncées au lancement du projet

Dès l'annonce du projet, plusieurs difficultés se firent jour.

En effet, fort d'une courte expérience de huit mois en programmation et principalement axée sur l'usage d'Objective-C et des technologies Web, grâce au travail lors des sessions au CESI, il m'apparut que développer cette application représentait un intéressant et très gros défi personnel et professionnel.

Mon tuteur, bien qu'expert dans l'utilisation d'Objective-C, m'annonçait ne pas avoir de compétence dans l'usage du langage SWIFT. De même, aucun autre membre des équipes de développement ne semblait n'avoir jamais eu à tester ce langage. Dès lors, je devais apprendre, depuis les plus basiques rudiments, un nouveau langage de programmation. Puis, il me faudrait utiliser ces nouvelles connaissances aussitôt acquises pour les mettre à profit dans la réalisation du projet.

Si l'environnement de développement restait le même que lorsque j'eus à travailler sur la première application de démonstration, à savoir Xcode, mes premières missions consistèrent dans la maintenance et l'amélioration de cette dernière application. Ainsi, l'application devant utiliser un fichier *Excel* pour simuler une base de données, j'eus à remplacer ce fichier par un fichier de type JSON, ainsi que de permettre l'utilisation des données qu'il contenait pour assurer le fonctionnement de l'application. En effet, afin d'assurer un fonctionnement plein et entier, la première application de démonstration devait fonctionner sans jamais avoir recours à internet.

J'eus, entre autres, à faire s'afficher une vue spécifique à un produit du panier que l'on sélectionne. Tout cela étant réalisé en usant du langage Objective-C. Une de mes dernières interventions sur l'ancienne application démonstrative consista à la rendre polyvalente et à pouvoir, en fonction de la carte de fidélité scannée, passer d'un magasin général (comme Carrefour), à une enseigne spécialisée dans le bricolage et les travaux (à l'exemple de Leroy Merlin).

Bien que l'on me proposât de commencer par faire de petits projets, je décidais plutôt de commencer mon apprentissage de SWIFT avec le projet. N'étant pas rebuté par les difficultés et les défis que cela représentait, bien au contraire, car je les considère comme un véritable moteur de motivation à réussir et à ne jamais rester dans une quelconque zone de confort, afin de toujours évoluer et d'aller de l'avant.

2) Déroulement du projet

Déroulé sur plusieurs temps et en de nombreuses étapes, l'évolution de ce projet est allée de pair avec ma propre courbe d'apprentissage. Respectant les procédures SCRUM, j'ai pu m'atteler à la réalisation du projet en complétant les objectifs fixés dans des tickets, répartis sur des périodes de deux semaines. Depuis une page blanche, je me suis lancé dans la programmation de cette nouvelle application et bien que je sois seul à travailler sur le codage iOS, je pus compter sur le soutien de mes collègues des autres équipes, ne serait-ce que sur la manière de coder efficacement et proprement, gardant, de fait, le contact avec les autres équipes.

2.1 Trame principale

Lorsque le projet de nouvelle application de démonstration fut lancé, ma première tâche consista à l'imaginer. Aussi, je m'armais de papier et d'un crayon afin d'y coucher toutes mes idées, nées avec le travail effectué sur la première version, avant même d'écrire la moindre ligne de code. Je dessinais des esquisses et des croquis de chaque vue, leurs fonctions, leurs éléments, et annotais toute information pouvant être utile lors du développement.

Dans le même temps, je suivis des cours Udemy et Openclassrooms sur l'emploi du langage SWIFT. Réalisant les exercices afin de gagner en compétences, je commençais à développer des applications simples pour mieux appréhender SWIFT et à mieux comprendre ses mécaniques. Ces exercices, réalisés sur mon temps personnel, me permirent de gagner beaucoup de temps dans la réalisation du projet. J'acquis les bases nécessaires pour répondre aux premiers besoins spécifiques du projet lors des premiers développements.

Lors de cette période, mon tuteur se trouva être le Product Owner du projet et avait la charge de la rédaction des tickets de travail. Ceux-ci furent rédigés en adéquation avec les idées que nous avions émises et notre vision première de la nouvelle démonstration. Celle-ci devait, en effet, fortement s'inspirer de son prédécesseur dans les fonctionnalités attendues. Vinrent les premiers objectifs et le lancement, à proprement parler, du projet. Je commençais par mettre en place l'architecture, en

utilisant le Storyboard fourni par Xcode. Architecture rapidement suivie par l'import des données depuis un fichier de type JSON, contenu dans un dossier compressé.

Rapidement, je mis en pratique ce que j'avais acquis avec mes premiers apprentissages en ligne, y adjoignant les bonnes pratiques acquises lors des sessions et des projets CESI, et au contact de mes collègues plus expérimentés. Je codais une première version de cette nouvelle application. Je créais un menu principal dont la fonctionnalité consistait à permettre la sélection d'un magasin, de scanner une carte de fidélité, d'enregistrer les informations du client recueillies au sein de la base de données interne, tout en affichant diverses recettes de cuisine, un accès au panier de courses et le lancement d'un nouveau parcours. Je programmais un premier panier, permettant d'ajouter et de calculer le prix total, sans aucune réduction, des articles qui étaient ajoutés à mesure qu'on les scannait. Puis, par simple sélection via une pression, j'avais rendu possible l'affichage d'une vue spécifique au produit sélectionné, présentant de nombreuses informations concernant celui-ci, comme une description plus complète.

Cette première période se termina bien que le travail sur le panier, le cœur de l'application, devait se poursuivre. En effet, à la suite de discussions avec mon tuteur, je lui proposais, non pas d'utiliser des designs, images et dessins achetés sur Internet, mais plutôt de faire appel aux designeurs de Budgetbox, dans les locaux de Paris, et ce, afin d'utiliser au maximum les compétences internes à l'entreprise. Bien que le personnel de Paris ne soit pas rompu à ce genre d'exercice, ils ont également relevé le défi et nous ont fourni des maquettes graphiques correspondant aux besoins. Un suivi régulier fut organisé via des réunions avec le logiciel Microsoft Teams.

J'entrais alors pleinement dans la seconde phase de développement. En effet, avec l'arrivée des maquettes graphiques, il m'apparut aussitôt que je fus dans l'obligation d'entièrement repenser l'application. Les éléments déjà codés n'étaient plus en adéquation avec les nouveaux attendus visuels et les fonctions de chaque vue. Dès lors, je me résolus à pratiquement reprendre le projet depuis le départ. Cependant, avec des visuels plus nets et des objectifs plus clairs, le développement se fit plus rapidement. D'autant plus que je n'hésitais pas à réemployer des morceaux de code précédemment programmés et s'intégrant parfaitement pour gagner du temps.

Parmi les nouveautés, une animation de lancement d'application devait être ajoutée. Une vidéo fut fournie par le studio parisien, mais, dans l'optique d'apprendre à coder en SWIFT, je décidais non pas d'afficher la vidéo, alourdissant l'application, mais à la coder entièrement. Étant donné que mon tuteur m'avait fait comprendre que le temps n'était pas un problème, je pus prendre le problème à bras le corps. Ce nouveau départ fut l'occasion d'intégrer dans le projet toutes les notions de codage propre que j'avais pu apprendre depuis. Loin de chercher à me simplifier le travail en utilisant immédiatement des librairies et des cocoapods pour obtenir des facilités, je préférais travailler sans, dans l'unique but d'aller le plus loin possible avec le langage et sa compréhension. Cette manière de faire, quoique chronophage, m'a néanmoins permis d'apprendre beaucoup. Tous les éléments

techniques abordés ici font l'objet d'une description plus approfondis dans la troisième partie de ce rapport.

Puis, vint la troisième et dernière phase du projet.

Figure 3: Un design simple et épuré

Le Product Owner étant accaparé par des dossiers clients faisant état d'une grande urgence dans leur suivi, et devant ma capacité à évoluer de manière autonome et à avoir la capacité de répondre aux besoins avec un minimum d'informations, il put se contenter de me donner un axe de travail, tandis que me revenait la charge de créer les tickets affiliés au travail demandé. Disposant d'une expérience de plus d'un an avec la méthodologie SCRUM, je pus mener à bien ces tâches supplémentaires et à en assurer la découpe.

2.2 SCRUM process

Au sein de l'entreprise Budgetbox, les équipes de développement suivent les processus que l'on trouve dans la méthodologie SCRUM.

La méthode SCRUM, dite agile, offre la possibilité de découper les tâches à effectuer et le temps de travail, afin de gagner en efficacité et en organisation. De ce fait, elle permet d'améliorer la productivité de nos équipes qui viennent rapporter leur retour d'expérience lors des *sprint reviews* et rétrospectives qui ont lieu en fin de sprint, un lundi sur deux.

Ces périodes de travail, chronométrées sur un temps de deux semaines, permettent une planification sur ce temps prévu du travail à effectuer. Le travail est pioché dans une véritable réserve prévue à cet effet, en fonction des thèmes, produits, etc. Chaque développeur se voit donc attribuer un certain nombre de tâches à réaliser dans ce temps imparti. Cependant, chacun de ces travaux se voit correspondre un temps de réalisation, correspondant à l'appréciation du développeur, ainsi qu'un niveau d'effort, déterminé collectivement, aidant à l'appréciation globale dudit ticket. Ainsi, une fois tous les jours ouvrés de chaque développeur pourvu de travail, la répartition se termine et la période nouvellement créée est ouverte.

Il est important de noter que chaque développeur se voit remis un objectif périodique à atteindre pour attester de la réussite du travail à faire.

Dès lors, tous les matins, une réunion formelle, appelée *daily*, a lieu. Échange quotidien lors duquel, à tour de rôle, chacun va pouvoir s'exprimer sur le travail effectué, les difficultés rencontrées, un rapide et très court retour d'expérience au besoin, ainsi que le travail que l'on escompte réaliser. C'est également le moment privilégié pour poser toute sorte de questions ou appeler de l'aide pour débloquer des situations de travail dangereuses pour mener l'objectif à son terme.

À la fin de la période, les équipes sont réunies pour attester de l'atteinte des objectifs, ou non, mais également de faire part de ses principales difficultés et d'apporter un retour d'expérience plus approfondis et global sur la période écoulée. C'est également à cette occasion que chacun présente le résultat de son travail au reste

des équipes qui peuvent alors faire un retour et mettre en exergue des manquements, oublis, etc. Le but étant de pouvoir se corriger aussitôt et présenter aux clients les produits les plus fonctionnels et propres possibles.

C'est donc sur ce rythme de travail que le projet de nouvelle application de démonstration fut réalisé. Le découpage des tâches en tickets tendant à être le plus fin possible, afin de garantir l'accessibilité des objectifs à atteindre, vis-à-vis de mes compétences techniques grandissantes, mais également de mes lacunes ou écueils.

2.3 L'apprentissage « sur le tas »

Comme présenté parmi les premières difficultés immédiatement identifiables au lancement du projet : l'apprentissage sur le tas.

Présenté parmi les premières et plus grandes difficultés auxquelles je dû faire face lors du lancement du projet : un apprentissage « sur le tas ».

En effet, au sein des équipes Connected-Shopper, nul n'utilisait le langage SWIFT, soit au quotidien, soit sur son temps privé. Seul mon tuteur, M. Kevin JUE, possédait une maîtrise de l'outil Xcode et du langage d'Apple Objective-C. Mais, cependant, pas de SWIFT, si ce ne sont que quelques concepts.

Ayant des objectifs à réaliser, je n'eus qu'à m'adapter à la difficulté et à apprendre par moi-même ce langage dont je ne savais rien ou presque. En effet, j'avais bien essayé quelques petits morceaux de code auparavant, mais sans jamais aller très loin. Afin de pallier mon manque de connaissances techniques, je commençais par chercher un moyen simple et efficace de prendre en main le langage et de commencer à réaliser mes premiers objectifs.

Ce moyen, je le trouvais sur Internet, notamment avec les sites de cours dématérialisés que sont Openclassrooms et Udemy. Le premier, après avoir téléchargé leur jaquette de présentation du cursus *développeur iOS*, me permit d'accéder à de nombreux cours, à la difficulté grandissante, qui m'apportèrent les bases nécessaires. Quant au second, je trouvais des cours, en français, sur le développement iOS également, abordant des thèmes similaires mais complétant certains manques dans les cours d'*Openclassrooms* et inversement. Rapidement, en travaillant les deux sources, et en complétant via des recherches sur *Stackoverflow*, je parvins à obtenir un niveau de débutant qui m'offrit la possibilité de résoudre des travaux toujours plus compliqués et ardus. Toutefois, lorsque survenait un travail très particulier, comme redessiner un élément graphique comme la barre de bas d'écran, élément natif des produits iOS, de manière plus customisée, les chaîne *Youtube* de développeurs américains me furent d'un réel secours également.

C'est donc autant sur mon temps personnel que sur le temps de travail (qui avait été mesuré et préparé à cet effet), que je travaillais le langage SWIFT. Ainsi, en réalisant les exercices des cours sur mon temps personnel et en travaillant les objectifs de travaux en entreprise, j'acquis de solides bases qui me permirent de monter en compétences.

Il est important de noter que les apports du CESI, non pas sur le langage SWIFT en lui-même, mais plutôt sur l'emploi des autres langages de manière propre, ainsi que

mes propres travaux personnels, m'apportèrent une plus grande confiance et de premières connaissances et expériences sur la manière de coder proprement. Ces acquis, je vins les importer directement dans mon projet d'entreprise, afin de toujours m'améliorer et suivre, dès le départ, les bonnes pratiques.

Si mon tuteur était le plus expérimenté dans l'utilisation des technologies d'Apple, il s'avéra qu'il ne fut pas le seul à posséder une expérience Apple. En effet, au fur et à mesure de nos échanges, monsieur Pierre LEGRAS s'avéra être en capacité de pouvoir m'apporter une aide certaine lors de travaux particulièrement ardus, à mon humble niveau, via des conseils techniques, des aides à la recherche ou encore par des explications concernant des bonnes pratiques. Les revues de code qu'il eut à faire sur le projet sont souvent accompagnées de commentaires des plus pertinents qui m'aident considérablement à m'améliorer.

Ainsi, en combinant toutes ces sources de connaissances, tant techniques que théoriques, complétées par un travail persistant et en continu m'ont permis d'évoluer de manière toujours plus aisée dans la technologie iOS qu'est le langage SWIFT. Désormais, j'ai acquis un niveau me permettant de travailler sur des sujets plus pointus et de travailler à améliorer les détails. De plus, il m'est apparu, au travers d'autres travaux employant d'autres technologies, que j'avais acquis une bonne capacité d'apprentissage, véritable atout pour des évolutions futures.

2.4 Mes relations avec l'équipe

Employé, en premier lieu, au sein de l'équipe *Connected-Shopper*, je fus, lors de l'année 2022, versé dans l'équipe Scaneo-cloud.

Occupant l'unique poste de développeur iOS de l'entreprise, j'eus à travailler seul, en-dehors de toute équipe. Aussi, afin de palier au risque d'isolation, malgré un poste se tenant dans un espace ouvert, je partageais entre mes différents collègues les revues de codes du projet de démonstration.

Dès lors, les retours que je reçu furent des plus motivants et m'apportèrent beaucoup de savoir et d'expérience que je mis à profit pour continuellement améliorer la qualité de mon code, mais également à rester en phase avec le reste des membres des équipes. C'est de cette manière que j'appris des principes tel celui du SOLID.

3) L'environnement de développement

La programmation en usant d'un langage d'Apple nécessite un environnement de développement (IDE) particulier. En effet, puisque soumis à une utilisation d'Xcode, application de développement uniquement disponible sur l'App Store, il est impératif d'être en possession d'un ordinateur Mac. La possession d'un iPhone est un plus non-négligeable pour construire notre application dessus et procéder à toute une batterie de tests simples et visuels. Toutefois, ce dernier n'est pas obligatoire.

3.1 Xcode

Xcode est l'environnement de développement créé, géré et distribué par Apple et il est totalement dévolu à de la programmation native en Objective-C et SWIFT.

Figure 4: le sigle d'Xcode

Cet environnement va nous permettre de développer nos solutions iOS, macOS tvOS et watchOS, soit un large panel de produits de la firme Apple et intègre la possibilité de sélectionner plusieurs critères et/ou options pour le développement.

Par exemple, lors de la création d'un nouveau projet, nous avons le choix du langage à utiliser, ou encore l'emploi des Core Data ou encore celui du storyboard. Notons que, mis à part le choix du langage de programmation, il nous est tout à fait possible d'ajouter par la suite certains des éléments qui nous étaient proposés lors de l'initialisation dudit projet.

Le storyboard fournit par Xcode est un puissant outil permettant une réalisation rapide, simple et efficace de nos vues à afficher, ainsi que des informations qu'elles doivent contenir. Il propose également des éléments graphiques prêts à l'emploi et simples à placer. Reliées à un contrôleur spécifique, ces vues peuvent par la suite être complétées via du code. Ce qui nous offre l'opportunité de changer dynamiquement leurs éléments par exemple. Mais Xcode permet l'émulation d'appareils de la marque Apple, à l'image d'un iPhone, et dans toutes les gammes commercialisées ou ayant été commercialisées, afin que nous puissions tester sur le type d'appareil sélectionné, et ce, quand bien même nous ne disposions pas du support physique affilié.

Xcode est donc l'outil le mieux adapté au développement natif iOS et fait l'objet d'un suivi régulier de la part d'Apple qui fournit régulièrement des mises à jour. En juin 2021, la version 13 sortait et implémentait le SWIFT 5.5. En juin 2022, Apple annonçait la sortie de Xcode 14.

3.2 Le matériel

Afin de remplir au mieux mes missions, je me suis vu confier par l'entreprise un macBook Pro, ainsi qu'un iPhone, d'abord un 6 SD, remplacé au cours de l'année 2021 par un iPhone 11.

3.3 SWIFT, le jeune langage d'Apple

Langage phare et récent de la firme à la pomme, SWIFT est un langage orienté objet présentant de nombreux avantages, ponctués toutefois de quelques faiblesses. En 2021, il fut classé parmi les dix technologies les plus recherchées.

Figure 5 : Sigle de SWIFT

a. Définition

SWIFT est donc un langage de programmation orienté objet qui vit le jour en 2014. Création d'Apple, il fut principalement pensé dans sa conception pour permettre le développement natif iOS et macOS, sachant qu'il offre également la possibilité d'un développement iPadOS, tvOS et watchOS.

Son objectif principal est d'offrir aux développeurs une grande puissance de développement de leurs applications iOS.

Notons qu'il s'agit d'un langage de type Sources Ouvertes, trouvable sur le site internet Github (https://github.com/apple/swift).

b. Caractéristiques

Ce langage est multiparadigmes et dévolu à un usage général. Il apporte de nombreuses fonctionnalités rapides, tous ces éléments venant se combiner pour rendre le langage moderne, mais également aisément accessible et simple d'utilisation, évitant les contraintes à son bon fonctionnement.

c Quels sont ses avantages ?

SWIFT présente de nombreux avantages que nous pouvons lister ainsi :

- Facilité d'accès
- Rapidité, sécurité et multiplateforme
- Ses bibliothèques dynamiques

En effet, une des plus grandes forces du langage est son accessibilité. Conçu pour des développeurs débutant (à l'image du rédacteur du présent document), il dispose d'un véritable bac à sable permettant de coder et tester le langage rapidement et d'en observer les résultats immédiatement. Il s'agit du *Playground*. Notons que celui-ci n'est pas conçu pour faire des applications complètes, mais simplement de découvrir le langage en manipulant des éléments simples et natifs.

Facile à apprendre et à utiliser, SWIFT est très concis et intuitif. Par retour d'expérience, son usage au quotidien est appréciable et permet d'obtenir un code fonctionnel aisément.

Disposant d'une technologie de compilateur Machine virtuelle de bas niveau et d'une bibliothèque standardisée, le langage est extrêmement rapide et puissant. Moderne, il garantit d'une grande sécurité, du fait des retours d'expériences notamment avec Objective-C, son prédécesseur. SWIFT fournit de nombreuses fonctionnalités de sécurité. Ainsi sa gestion automatique de la mémoire, à la différence de l'Objective-C où l'où devait préciser les pointeurs lors de la déclaration d'une variable, en plus de son type.

Un des principaux avantages de ces sécurités est que cela force le développeur à rédiger un code propre et cohérent, garantissant à moindres frais une bonne lisibilité, tout en se prémunissant d'erreurs, à l'exemple d'une re-déclaration d'une même variable, mais d'un type différent.

Disposant d'une communauté forte et active, SWIFT prend en charge de nombreuses plateformes, outre Apple, mais il peut être utilisé sous Linux, Windows ou encore Ubuntu.

Ce langage intègre des bibliothèques dynamiques, ces dernières n'existant qu'en-dehors du code et n'étant téléchargées qu'au besoin, à contrario des bibliothèques statiques. Ce qui allège le temps de chargement et le poids de l'application développée.

d Quelles sont ses limites ?

Certes avantageux, le langage SWIFT présente toutefois certaines faiblesses qui peuvent ralentir son utilisation lors de développement de solutions. Lesdites faiblesses peuvent être répertoriées ainsi :

- La jeunesse du langage
- L'incompatibilité avec de plus anciennes versions d'iOS

Présenté et rendu disponible en 2014, SWIFT reste, malgré tout, un langage assez jeune. Nous pouvons avoir certaines de ses capacités et ressources qui ne sont pas gagnées de robustesse, à contrario de langages de programmation plus anciens.

Si nous avons noté qu'il est multi-plateforme, SWIFT est clairement un langage pensé pour être optimum lors d'un développement iOS natif et l'utilisation de l'IDE d'Apple Xcode.

Il est à noter que les mises à jour sont très fréquentes, ce qui peut poser certains problèmes par moment.

Lorsque l'on développe en usant de SWIFT, il faut bien avoir en tête que toutes les versions d'iOS ne peuvent en bénéficier. En effet, la version minimale que supporte SWIFT est la version iOS7. En deçà, les appareils ne supportent que des applications en Objective-C, ce qui freine considérablement la rétrocompatibilité pour les personnes disposant d'iPhone d'anciennes générations, aujourd'hui plus produite par la firme à la pomme.

e Comparativement à Objective-C

Si l'on doit comparer les deux langages d'Apple, les deux sont orientés objets et conçus pour le développement OS X et iOS. Pour autant, SWIFT n'est pas le successeur direct d'Objective-C.

Chacun possédant ses capacités, forces et faiblesses qui lui sont propre, mais il est possible de les combiner. En effet, lorsque l'on développe en SWIFT, nous pouvons, à loisir, coder en Objective-C, sans que cela ne soit un problème. Certaines fonctions utilisées en SWIFT sont annotées comme étant des fonctions Objective-C. SWIFT se voit offrir le luxe de la possibilité d'user des API d'Objective-C sans le moindre conflit.

Notons toutefois, que SWIFT, à la différence d'Objective-C qui date de 1984, est un langage, nous l'avons vu, moderne et disposant d'une syntaxe plus simple, qui offre la possibilité d'un gain de temps dans le codage. À la différence, Objective-C ne

dispose pas d'une grande intuitivité et dispose d'une syntaxe beaucoup plus complexe, ce qui ralentit le temps de codage.

Apple rapporte que SWIFT est 2.6 fois plus rapide qu'Objective-C.

Rappelons qu'avec SWIFT, nous n'avons pas accès à toutes les versions d'iOS, et particulièrement les plus anciennes, antérieures à iOS7, au contraire d'Objective-C qui reste compatible avec toutes les versions d'iOS et macOS.

f Représente-t-il l'avenir du développement iOS ?

Mais alors, SWIFT est-il véritablement un langage de programmation d'avenir pour iOS ?

Considéré comme étant très jeune, il est ressenti comme étant, pourtant, un langage prometteur.

Suivi et mis à jour régulièrement par les équipes d'Apple, il gagne continuellement en maturité et dispose de toujours plus de fonctionnalités et viendrait à remplacer l'Objective-C en tant que principal langage de programmation d'Apple. De même, est apparu SWIFT-UI qui apporte son lot de nouveautés et de possibilités nouvelles dans le développement de solutions iOS.

4) La chasse aux files d'attente en caisse

Depuis quelques années, les grandes surfaces, et désormais même de plus petits magasins, facilitent au maximum le passage en caisse de leurs clients. En effet, les longues et interminables files en caisses sont chronophages et peuvent laisser un mauvais souvenir à leurs clients qui, potentiellement, pourraient ne plus revenir. Pire encore, avec le développement des achats en ligne, beaucoup de clients finissent par ne plus se déplacer en magasin, privilégiant Internet pour leurs achats, même courants.

Figure 6: supprimer les files d'attente, un enjeu crucial...

4.1 Principe

Il fallut donc entièrement repenser le parcours d'achat client en magasin et trouver un moyen de diminuer les files d'attente, voire de les supprimer.

Il fut choisi de mettre en place des terminaux de paiement, permettant aux clients de valider leur panier de la même manière que depuis leur écran d'ordinateur, chez eux. Ce qui amena à la création des caisses en libre-service.

Allant plus loin dans ce concept, c'est dans ce contexte qu'apparurent les scannettes. Dès lors, la clientèle des magasins se voyait offrir la possibilité de scanner eux-mêmes les articles qu'ils ajoutaient à leur panier, leur offrant alors la possibilité de suivre la taille et le coût de leurs achats, mais également de ne plus avoir qu'à fournir ladite scannette directement à l'hôte de caisse, gagnant un temps considérable.

Il n'était plus alors nécessaire de décharger l'entièreté de son chariot pour que chaque article soit enregistré avant encaissement. En effet, tout le contenu ayant été précédemment scanné, il ne restait plus aux clients qu'à régler leurs achats, diminuant drastiquement le temps de passage en caisse.

Le concept fut encore amélioré avec la mise en place de caisses dévolues spécialement aux clients utilisateurs de scannettes, abaissant plus encore le temps de passage en caisse.

Dans un souci de constamment rester à la pointe, les enseignes s'équipent désormais de solutions applicatives natives, en complément des scannettes, via les smartphones et iPhones de leurs clients. Le principe restant le même, les avantages également puisque l'accès aux caisses *Scan'Lib* reste accessible.

5) Les maquettes graphiques

Afin de pouvoir planifier les tâches et les principaux axes de travail, mais également de permettre une projection sur la forme de la future application de démonstration, il était nécessaire de posséder des maquettes graphiques.

5.1 De mes ébauches…

Lors du lancement du projet, les seules bases permettant de tisser un premier fil conducteur furent les fonctionnalités de la première application de démonstration. Cependant, l'idée consistait à la mettre à jour, notamment sur le côté graphique.

Ainsi, laissé libre de pouvoir réfléchir au design élémentaire, je croquais mes idées, les couchant sur papier tout en y annotant les principales fonctionnalités attendues par vue et de soumettre ces croquis à l'approbation de M. JUE, également Product Owner du projet. Inspirés de la première démonstration, ainsi que des discussions techniques, mes croquis ont donc servi de base pour les premiers développements de la nouvelle application.

Cependant, ces croquis bien que précis dans leur description, ne pouvaient correctement répondre au besoin. N'ayant, pour le moment, pas de compétences fortes dans le design, une faiblesse apparaissait et risquait de desservir dans le temps long. Aussi, alors qu'une première idée fut d'acheter des icônes et des idées de design auprès de sociétés tiers sur Internet, une autre solution fut finalement envisagée, plus économe et efficace.

5.2… Aux maquettes de Paris

En effet, ayant appris l'existence d'une équipe spécialisée dans le design, au sien du bureau de Budgetbox de Paris, j'ai proposé que la mission de créer des maquettes graphiques leur fut confiée, étant donné qu'il s'agissait de l'équipe possédant l'expérience nécessaire pour cela.

Dès lors, l'équipe design du studio de Paris se mit au travail et nous fournit rapidement de premières esquisses et maquettes, qui furent soumises à notre approbation. Travaillant avec l'outil en ligne *Figma*, ils purent nous partager aisément leurs réalisations, avec les côtes, polices d'écritures et autres informations graphiques fondamentales.

Toutefois, loin de rester inactif en attendant la réception des maquettes graphiques, je poursuivis le développement, en me concentrant sur le fonctionnel, au détriment du visuel. Le but étant d'avancer sur la partie métier du code, qui est géré ici, dans les locaux de Rouen de l'entreprise. De plus, ce travail me permit de gagner en expérience dans la programmation en SWIFT, en attendant les maquettes graphiques.

Lorsque les premières d'entre elles nous furent livrées, je pus les programmer. Mais le travail précédemment effectué se retrouvait alors, en grande partie, obsolète. Je me résolu d'être dans l'obligation de le reprendre entièrement. Cependant, cette refactorisation du code me permit, non seulement d'assurer une meilleure gestion de mon code une fois réécrit, mais également de le rendre plus efficient, lisible et réutilisable. Je pus, ainsi, intégrer plus facilement les éléments visuels.

Par la suite, je vous décris de plus ample et plus complète manière les différentes étapes et mécanismes de la réalisation de ce projet.

III La réalisation du projet

1) La mise en place de l'architecture.

La première étape dans la réalisation du projet fut de mettre en place l'architecture. Ainsi, devant m'inspirer de l'architecture de la première application, et via l'utilisation du *storyboard*, je mis en place celle-ci. Les prérequis importants consistaient en deux éléments bien particuliers que sont la *NavigationBar* et la *TabBar*, des éléments iOS permettant de naviguer aisément dans l'application.

Cependant, de par mes lectures et premières expériences en programmation, j'avais appris la puissance du papier et du crayon pour y coucher idées et concepts. De cette manière, je pu corréler ma vision de l'architecture de l'application avec la vision de mon tuteur. Le prototypage terminé, je basculais dans l'environnement de développement Xcode.

Usant du storyboard, je créais rapidement les vues nécessaires, les nommant et les affiliant à des contrôleurs leur étant dévolus. La particularité était que le contrôleur de la TabBar devait **obligatoirement** se situer tout en amont de la hiérarchie, afin qu'il soit utilisable partout dans le reste de l'application.

Cette première étape achevée, je passais à la suite.

Dans la suite du développement, j'eus à revenir plusieurs fois sur l'architecture du projet, l'adaptant aux différents besoins. Procédant à des ajustements nécessaires pour un fonctionnement optimal. Tel fut le cas, notamment, lorsque j'eus à implémenter une animation au lancement de l'application.

2) Récupération et décompression

. Notre application de démonstration, pour des raisons de performances et de sécurité, doit fonctionner sans jamais utiliser Internet. En effet, destinée à des salons et

autres rencontres pour démarcher de nouveaux clients, la publicité serait des plus mauvaise si, faute d'une bonne connexion internet, l'application ne pouvait fonctionner de manière optimale. Il fallait donc lui créer une base de données complète embarquée.

Là où la première application de démonstration usait d'un fichier *Excel* pour importer les données nécessaires, cette application le faisait à partir d'un fichier de type JSON, lui-même contenu dans un dossier compressé.

Dès lors, il me fallut progresser en deux étapes : la décompression du dossier, puis l'extraction des données contenu dans le fichier JSON.

2.1 La décompression

En premier lieu, je devais décompresser le dossier dénommé « démo », embarqué avec l'application, afin d'en exploiter toutes les ressources. Or, il n'existe pas de méthodes natives et directement accessible. J'eus recours à un *cocoapod*, *ZIPFoundation*, pour obtenir la méthode nécessaire.

Afin de pouvoir effectuer la décompression, il fut nécessaire de spécifier le chemin dudit dossier, tout en précisant le nom et le type d'extension. Ces précisions faites, nous pouvons lancer la décompression, mais celle-ci se fait sur un temps très important, pouvant aller jusqu'à deux minutes, ce qui représente un problème de taille.

En effet, un tel temps de travail n'est pas viable pour un emploi régulier. Fort heureusement, la solution était pensée pour n'avoir recours à ce temps de décompression qu'une unique fois, lors du premier lancement de l'application.

Un ***cocoapod*** est une librairie officielle, intégrable à un projet iOS uniquement, permettant d'importer des fonctionnalités développées par d'autres équipes.

Exemple :

BarcodeScanner, librairie permettant d'obtenir un scanner de codes-barres et QR Codes.

C'est en usant d'une condition simple que nous vérifions que notre dossier est déjà décompressé ou non. Ceci, simplement en retournant un booléen sur la présence ou non d'un dossier contenant tous les fichiers du dossier compressé. Ceci fait, je programmais pour que notre code puisse aller chercher le fichier nommé « *data.json* » au sein de notre dossier décompressé, afin de poursuivre la suite du travail de récupération des données. Notre dossier maintenant décompressé, et les informations qu'il contenait désormais accessibles, nous vérifions au préalable et par une seconde vérification conditionnelle, l'existence de ce fichier JSON. Ceci fait, il est temps d'en extraire les données.

2.2 L'extraction

Xcode propose l'utilisation d'un élément permettant d'embarquer des données en une base, en SQLite, dans l'appareil même. Ce sont les *Core Data*. Je reviendrais dessus sur le chapitre affilié. C'est donc en utilisant les méthodes natives des Core data que je comptais extraire les données pour les regrouper dans une base qui doit être exploitée par la suite.

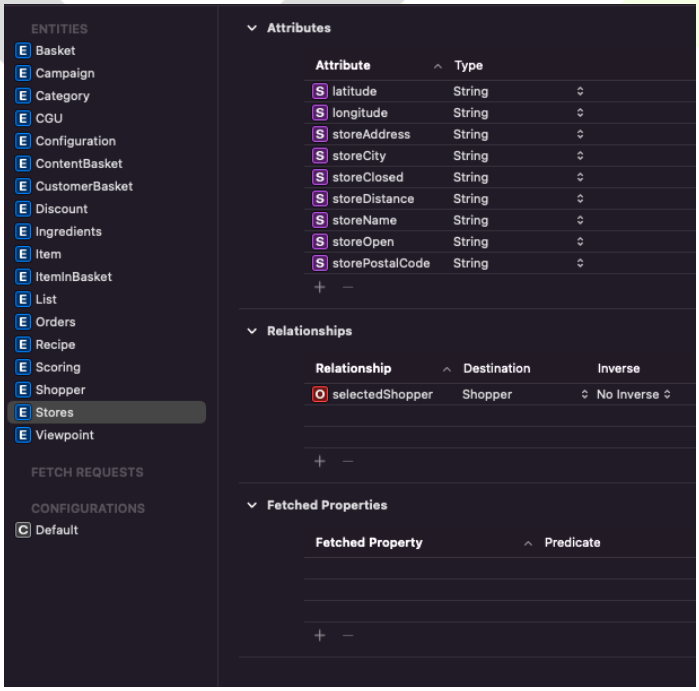
Afin de remplir cet objectif, nous commençons par récupérer sous la forme de *Data*(données), le contenu du fichier JSON. Puis, nous désérialisons celles-ci sous la forme d'un Dictionnaire. En procédant ainsi, il est alors possible de boucler sur ce-dernier pour en récupérer les informations tour à tour. Ainsi, par l'utilisation d'un switch, nous venons récupérer chaque clef (exemple : Campaign, Item, etc.) et le tableau qui consiste en la valeur associée à la clef, pour les extraire, une à une afin de remplir notre base de données. Chaque cas de ce switch correspond à une entrée d'un énumérateur, le *DagaieParser*. Le but est de gagner en flexibilité, ainsi qu'en projection de futures améliorations et, ou corrections.

2.3 Le parse des données

Une fois l'extraction des données effectuées, Je pus les transférer vers la base de données embarquée.

Notons qu'afin de pouvoir optimiser le code, les entités au sein de la base de données, ont été préparées en amont, en déclarant chaque entité et l'ensemble de ses attributs dans un fichier Core Data.

Suivant le modèle défini en amont, je procède à une vérification des valeurs des clefs et leur égalité. Si c'est le cas, les données contenues dans la valeur associée sont enregistrées comme autant d'attributs. Le processus est alors automatisé,



offrant un avantage non-négligeable quant à son réemploi ou l'apport de nouvelles entités aisément.

L'entité nouvellement créée est alors enregistrée dans le contexte. En effet, si nous ne procédons pas à son enregistrement, l'entité est écrasée par la suivante et ainsi de suite, ne laissant que la dernière entité de disponible.

Désormais en possession d'une base de données complète et fonctionnelle, il m'est alors possible de créer les prochaines vues et d'interagir avec nos entités et leurs attributs pour la suite du développement de Scan&shop.

3) La Base de Données

XCode permet, outre les NSUserDefaults, l'utilisation des Core Data pour le stockage permanent de données.

3.1. Core Data

Core Data est une base de données SQLite locale. Munie d'une API et orientée objet, elle nous permet de stocker nos données directement dans notre appareil (ici, iPhone). Également, Core Data vient transformer les informations stockées directement en objet manipulable par SWIFT, comme n'importe quel objet.

Core Data s'appuie sur une technologie : Core Data Stack qui s'articule autour de cinq classes :

- *NSManagedObjectModel*
- *NSPersistentStoreCoordinator*
- *NSManagedObjectContext*
- *NSManagedObject*
- *NSPersistentContainer*

Ainsi que sur trois fichiers :

- *DataModel.xcdatamodelld*
- *DataModel.momd*
- *DataModel.sqlite*

La première couche est représentée par NSManagedObjectModel, puisqu'elle va nous permettre de définir le schéma d'organisation de nos données, et donc d'en faire des modèles.

C'est dans ce fichier que l'on va définir nos objets

3.2. Quelles différences avec les NSUserDefaults.

Ainsi que nous l'apprend la documentation officielle d'Apple, les *NSUserDefault* et Core Data nous permettent, tous deux, de permettre le stockage permanent d'informations sur notre appareil. Cependant, ils n'ont pas été conçu pour les mêmes besoins.

En effet, si les *NSUserDefault* sont, ni plus ni moins, qu'un dictionnaire de données, nous sommes très limités quant à leur utilisation et les types des données enregistrées. Nous ne pouvons conserver, en leur sein, uniquement des données du type Properly List, à savoir : Array, Dictionary, String, Date, Data, Int, Double, Float.

Il n'est pas possible de faire persister des données d'un type que nous aurions créé, par exemple un Item.

Les *NSUSerDefault* sont utilisés pour sauvegarder des données très simples, telles les préférences utilisateurs. Dans notre projet, nous les utilisons afin de connaître l'état de notre session de courses (est-elle en cours, ou non ?) qui déterminera une partie de l'affichage de notre menu principal, ainsi que des items accessibles dans notre TabBar. On pourra également, lors du scan de la carte de fidélité, enregistrer les informations de l'utilisateur, ou encore le magasin que l'utilisateur a sélectionné.

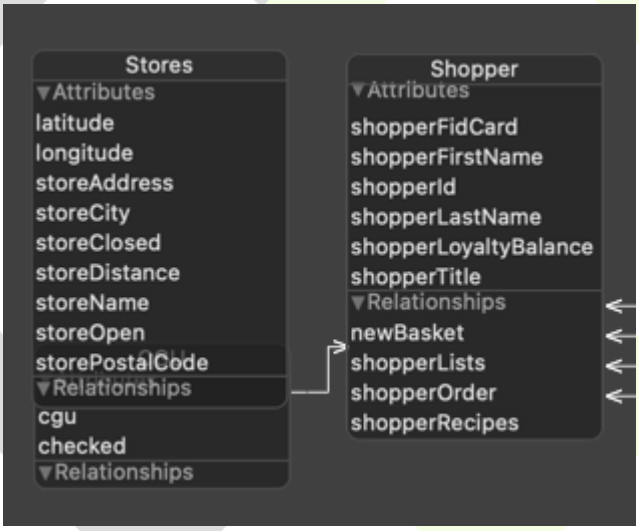


Figure 9: Exemple d'entités sous Core Data

4)- Une animation au lancement.

Alors que la première démonstration était pourvue d'une animation d'attente de spinner, une roue se dessinant et s'effaçant tout à la fois, lorsqu'on la lançait, le studio design de Paris fournit une vidéo présentant une nouvelle animation, faisant apparaître les lettres du nom de l'application, Scan&Shop, les unes après les autres, changeant de couleur, rejointes par le logo, avant une translation vers le haut de l'iPhone.

Cette animation est là, notamment, pour faire patienter pendant la décompression du dossier au premier lancement, mais également par la suite, en guise d'ouverture.

Les projets SWIFT gérés avec Xcode possèdent une vue appelée *Launchscreen*, ou vue de lancement. Cette vue est la première que l'utilisateur visualise lorsqu'il ouvre l'application. Le problème fut que cette vue ne permet pas d'exécuter des évènements complexes, telles des animations, mais uniquement d'afficher des éléments statiques.

Aussi, et afin d'insérer mon animation, j'eus à devoir retravailler l'architecture en incorporant un nouveau contrôleur de vue et la vue afférente. De cette manière, je pu y incorporer l'élément dynamique.

Désormais, deux choix s'offrirent alors à moi :

- Soit j'insérais directement la vidéo, telle qu'elle, dans ma vue. Ce qui ne manquerait pas d'alourdir l'application, avec les risques que la vidéo ne se joue pas avec une bonne qualité d'image. Mais, ceci pour un gain de temps de réalisation non-négligeable.

Soit, je recréais, via le code, intégralement l'animation. Beaucoup plus long à coder, car je ne connaissais encore rien aux animations avec SWIFT, mais pour un gain de connaissances techniques plus important et la possibilité de personnaliser et retravailler aisément l'animation dans le temps, et donc, de meilleures possibilités d'évolutions aisées dans le futur.

Face à ses deux solutions, j'envisageais celle qui devait pouvoir m'apporter le plus de compétences techniques et, étant donné que j'étais là pour apprendre je choisissais la seconde option. De plus, accepter de sortir de sa zone de confort et de prendre des risques calculés, offre une meilleure courbe d'apprentissage et des connaissances techniques bien plus étoffées.

Ce fut un véritable défi technique, puisqu'il me fallait être parfaitement raccord avec le modèle fourni dans la vidéo. Afin de mener à bien ce travail, je l'accomplissais en plusieurs étapes bien distinctes :

- La reconfiguration du storyboard
- La configuration des éléments
- L'animation à proprement parler

4.1 Reconfiguration du storyboard

La première étape consista donc à reconfigurer mon storyboard. En effet, jusqu'à présent, le point d'entrée (après le *LaunchScreen*, qui est un élément obligatoire) était le *TabBarController*. Or, il me fallait faire apparaître mon animation avant tout autre élément graphique, et surtout ma barre de bas d'application ne devait pas paraître à l'écran.

Pour cela, j'ajoutais un contrôleur de vue dans mon storyboard. D'une simple condition, je l'annotais comme étant le point d'entrée de l'application. Xcode assurant alors la gestion pour son lancement.

Maintenant, il me fallut programmer l'animation.

4.2 La configuration des éléments

La configuration des éléments graphiques se fit en plusieurs temps. Premièrement, j'eus à créer mon texte, sans animation, afin de placer correctement chaque élément, de manière conforme à la première partie de la vidéo fournie.

En prévision de la suite, je conçu une fonction qui découpais mon texte en autant de label que celui-ci contenait de lettres. Le but étant de pouvoir effectuer par la suite le changement de couleur dudit texte lors de l'animation plus aisément.

Ayant recours à plusieurs vues d'empilement, *stackview*, je créais une classe venant étendre la classe des vues d'empilement, lui ajoutant des méthodes à réemployer pour la création de l'animation. Le but recherché étant d'alléger le code de l'animation.

Le texte placé, je travaillais alors sur l'image du logo : le losange. Celui-ci devant venir se placer sur la droite du texte, avec comme effet de l'encadrer en partie. Là encore, je jouais sur les contraintes programmatiques. Car, toute l'animation et ses éléments graphiques sont créés de manière programmatique. Usant de la méthode *CGAffineTransform*, qui permet de transformer nos éléments, par translation ou en jouant sur leur taille, je pus décaler texte et logo, de manière que tous soient à la place qui leur était destinée.

Nos éléments graphiques étant maintenant en place, il ne me restait plus qu'à les animer.

4.3 L'animation

Une fois de plus, je découpais en plusieurs sous-tâches la programmation de l'animation, afin de ne pas disperser mes efforts et les rendre futiles.

La première étape consistait à faire apparaître mon texte, avant que celui-ci ne vienne se déplacer légèrement durant l'apparition des lettres, et ce afin que le logo puisse grandir et se mettre en place, comme s'il venait du fond de la vue. Pour cela, j'usais de la méthode *animate*, animer en français, des vues d'empilement contenant mes éléments graphiques. Cependant, et afin d'éviter une apparition trop brutale, l'animation se lançait en fonction d'un minuteur que j'ajoutais via le code.

Une des principales difficultés, en sis de l'apprentissage et de l'implémentation des méthodes d'animation, fut le chronométrage qui se devait d'être parfaitement dosé pour que les éléments interagissent à la perfection et soient conformes à la vidéo.

Nos lettres en place, je pus dès lors intégrer leur changement de couleur. Je chronométrais ainsi : tandis que la lettre suivante apparaissait, la précédente changeait alors de couleurs, donnant une impression de continuité, telle une vague courant sur les flots. Une fois de plus, tout se joua sur un chronométrage parfait. Programmant grâce à des boucles j'automatisais le procédé, afin que celui-ci soit plus aisément réutilisable par la suite, autant que de besoin.

L'animation finale, qui ne peut être déclenchée que lorsque l'on est certain d'avoir un dossier contenant nos data qui soit décompressé et accessible. Celle-ci consiste en un mouvement vers le haut de notre texte, replaçant chaque lettre de manière reformer le nom de l'application, Scan&Shop, mais sur une seule ligne, au niveau de ce qui sera la *NavigatuionBar*.

Il est important de noter que notre animation a, notamment, pour but d'occuper l'utilisateur pendant que la décompression du dossier, l'extraction des données et leur répartissement auprès des diverses entités de la base de données embarquée. Aussi, tant que ces actions ne sont pas notées comme étant terminée, l'ultime partie de l'animation ne doit pas pouvoir se jouer.

Une fois ces actions terminées, notre texte et le logo viennent translater vers le haut de l'écran de l'iPhone, en se reformant sur une seule ligne. Puis, nous ouvrons la page du menu principal.

Afin d'informer l'utilisateur de l'étape dans laquelle se trouve la décompression, du texte s'affiche au bas de l'écran, annonçant les différentes étapes du processus.

Lorsque s'affiche le message « Application prête », l'application nous informe que les données ont bien été extraites et réparties correctement. Il est alors temps de se rendre au menu principal.

5) Le menu principal

Partie conséquente de l'application, le menu principal permet à l'utilisateur de réaliser de nombreuses actions, comme sélectionner son magasin, voir des recettes de cuisines, etc.

Face à l'important travail devant être effectué, les tâches furent découpées en fonction de blocs de fonctionnement. Pour une plus grande simplicité de lecture, je procède à la description de chaque bloc indépendamment.

5.1- La mise en place du menu principal

La création du menu principal intervient après l'arrivée des maquettes graphiques de l'application. Ce menu présentait d'importantes difficultés de prime

abord. En effet, outre le principe de l'en-tête se réduisant ou s'agrandissant en fonction de l'action de l'utilisateur, il me fallait obtenir un menu déroulant décomposé en plusieurs éléments indépendants. Chacun de ces éléments devant permettre un accès à d'autres vues de l'application.

Figure 10: Collection de vues choix magasin et scan carte de fidélité

Ainsi, l'écran du menu principal permet à l'utilisateur de réaliser les actions suivantes :

- L'affichage d'un message de bienvenu avec le prénom de l'utilisateur connecté et le total de sa cagnotte
- La possibilité de scanner une carte de fidélité
- La sélection de son magasin
- Un accès à sa liste de courses
- Visualiser tous ses coupons de réductions en attente d'utilisation
- Donner un accès direct aux ayons d'un magasin
- Visualiser diverses recettes de cuisine

Afin de répondre au besoin, la solution envisagée fut de créer une classe de type *UIViewController* qui hériterait de plusieurs classes en fonction des besoins, devant regrouper une UIView et une *UICollectionView*.

La principale difficulté fut de s'assurer du bon fonctionnement de la collision entre l'en-tête et le haut de l'écran, pour obtenir l'effet escompté. Aussi, je décidais, dans un premier temps, de partir depuis une vue simple et y ajouter, via le code, une autre vue devant contenir les informations de l'en-tête et, en dessous, une vue scrollable, qui contiendrait tous les autres items demandés.

Travaillant à partir des maquettes graphiques, je codais cette partie cruciale de l'application. Ne sachant ce que fut une collision d'en-tête, je passais quelques temps à rechercher et à questionner autant internet que mes collègues. Finalement, je trouvais un tutoriel expliquant précisément la démarche à suivre pour sa création.

Répliquant et adaptant à mon propre projet ce tutoriel, j'obtins rapidement l'effet recherché. Désormais, il me fallut programmer le reste des informations à afficher souhaitées.

Avant de continuer plus avant, il me faut d'abord soulever un point très important concernant le codage de ce menu principal.

Au démarrage de ce codage, je ne vis pas de meilleure solution que d'ajouter, par la voie programmatique, chacun des éléments, afin de les positionner et de les travailler indépendamment.

Cependant, alors que j'avais, pour ainsi dire, terminer de créer entièrement le menu principal, je compris avoir fait fausse route. Loin d'être maintenable, mon code était compliqué à lire et demandais une grande gymnastique mentale pour se projeter mentalement sur le positionnement des éléments. Ce point étant particulièrement problématique dès lors que je souhaitais ajouter de nouveaux éléments. De plus, cela alourdissait drastiquement le code, car il me fallait préciser chaque contrainte graphique de chaque élément, avant d'ajouter ce-dernier à la vue principale. Pour autant, bien qu'ayant dépensé beaucoup de temps de travail et d'efforts quant à la réalisation de ces tâches, je pus néanmoins m'améliorer grandement au niveau technique. En effet, j'appris alors à me passer de l'utilisation du storyboard pour la création de vue de manière programmatique. Ce fait m'aida à mieux appréhender la construction graphique et la gestion automatique derrière le storyboard d'Xcode.

Face à cet alourdissement, l'augmentation du risque de générer des erreurs et la qualité du code toujours moindre, en ajoutant ce que j'avais appris au travers de ces tâches, je pris la décision, tout en informant mon tuteur, de reprendre, entièrement, le codage de ces tâches. Ayant découvert de nouveaux concepts et de nouveaux éléments en SWIFT entretemps et pouvant m'aider à réaliser un code plus propre et maintenable dans la durée.

Dans la suite de ce document, je vous décris la manière dont j'ai codé l'écran d'accueil actuel, tout en réutilisant le code fonctionnel déjà existant.

Cherchant à respecter les principes SOLID, je posais d'abord à l'écris tous mes algorithmes, ainsi que mon pseudo-code, m'obligeant à retravailler mes méthodes. Alors que des interactions avec l'utilisateur étaient déjà en place, je choisis de les mettre en suspens via un booléen, afin de ne pas avoir à me disperser dans la reconstruction du menu principal. Ce redécoupage me permit de mieux attribuer les responsabilités à chaque fichier et à grandement alléger le code.

5.2- Choisir le magasin et entrer sa carte de fidélité

Après avoir recadrer mon travail, je remplaçais la vue scrollable par une collection de vues (*UICollectioniew*), là aussi scrollable mais découpée en autant de cellules qu'on le précise lors de la construction de cet élément natif.

La première proposition devant être faite auprès de l'utilisateur est de pouvoir sélectionner un magasin, ainsi que d'entrer sa carte de fidélité, via un scan de cette-dernière.

Ainsi, j'ajoutais dans la première cellule une autre collection de vues, la *ShopAndFidCarCollectionView*. Cette collection, présentée à l'horizontale, doit présenter à l'utilisateur les informations concernant le magasin qu'il a sélectionné, ainsi que concernant sa carte de fidélité.

M'appuyant sur la maquette graphique, je découpais le travail en trois principales sous-tâches : le choix magasin, le scan d'une carte de fidélité, la mise à jour de la collection de vues et l'ajout d'une cellule reprenant le code barre de la carte de fidélité entrée. Choisissant de travailler dans l'ordre d'apparition, j'attaquais le développement du choix d'un magasin.

a. Le choix du magasin

Lorsque l'utilisateur, qu'il soit connecté ou non, sélectionne un magasin la cellule appropriée affiche à l'écran les informations dudit magasin, à savoir le nom de l'enseigne, son adresse et ses horaires. Incluant un bouton, cette cellule permet également d'en changer aisément et à tout moment.

A l'inverse, lorsqu'aucun magasin n'est encore sélectionné, on affiche à l'écran une cellule spécialement dédiée à cet état de fait qui invite l'utilisateur à en sélectionner un.

J'ai choisi de conserver un fonctionnement simple de la cellule. En effet, une simple pression suffit à accéder à un menu déroulant, avec la possibilité de chercher un magasin spécifique ou bien de limiter notre rayon de recherche, via l'utilisation des coordonnées géographiques, autour de nous pour n'afficher que les magasins proches.

Lorsque l'utilisateur sélectionne un magasin, les informations dudit magasins sont récupérées depuis la base de données embarquée, puis enregistrées au sein des NSUserDefaults, les informations utilisateurs par défaut, dans une classe *StarShop* qui est réemployée lors de la construction de la cellule de magasin dans le menu principal. L'utilisation de ce *NSUSerDefault* se justifie par le fait qu'il s'agit là d'une préférence utilisateur.

Nos informations recueillies, il me faut pouvoir les afficher correctement et en correspondance avec la maquette.

b. La vue de sélection de magasin

Héritant de la classe native d'une vue (*UIView*), cette classe permet, au travers de son unique méthode publique, d'implémenter la vue attendue concernant le magasin sélectionné.

Pour créer cette classe, je repris l'existant du précédent travail qui consista en la programmation sans l'utilisation du storyboard de la vue. Cependant, les maquettes ayant été fournies entretemps, je devais reprendre ce code pour le réadapter au besoin. Afin de mieux me projeter et de mieux discerner les éléments graphiques nécessaires, je couchais sur papier la construction de cette vue, de la même manière que lors de mes travaux sur le web.

Reposant sur l'utilisation de plusieurs vues d'empilement (*UIStackView)*, permettant à nos éléments de s'imbriquer les uns dans les autres, j'appliquais alors ce que j'avais au préalable posé sur papier. J'obtins plus facilement le résultat attendu et sans avoir à gérer de grosses difficultés. Dès lors, j'eus à implémenter la possibilité de retourner au menu principal.

Pour cela, il me fallut instaurer des délégations (*delegates)* dans le code, permettant de mieux gérer ma hiérarchie de vues et ainsi de garder le lien entre les vues.

Le travail sur la sélection de son magasin est décrit plus en détail plus loin dans ce document.

```
weak var parent: HomeViewController?
public var shopperRepository: ShopperRepository!
public var context: NSManagedObjectContext!
```

Figure 11propriétés de parentage, dépôt et contexte

c. La cellule d'ajout de carte de fidélité

Pour que l'utilisateur puisse profiter au maximum de l'application, il doit pouvoir scanner sa carte de fidélité au plus tôt. Pour cela, la cellule d'ajout de carte de fidélité (l'*AddFidCardCell*) est la vue entièrement dédiée à cet objectif.

En effet, de conception simple, avec un message précis et un symbole explicite, l'interaction avec cette cellule permet d'accéder à la vue du scanner. Il peut alors scanner sa carte de fidélité.

Une méthode va alors rechercher dans la base de données embarquée, à partir du code lu par le scanner, à quel client ce code correspond-il. Une fois le client identifié, ses informations sont récupérées pour être enregistrées dans les préférences utilisateurs. Le but est de pouvoir les utiliser aisément, en n'ayant eu à les implémenter qu'une unique fois (principe du Singleton). Le profil sauvegardé, l'utilisateur est alors noté comme étant « connecté ».

Quant à la conception graphique, une fois de plus ce fut une construction faite via le code et non le storyboard. La maquette présentant un élément visuel simple, il me fut aisé de la reproduire avec le code, en suivant le même principe que pour l'élément décrit précédemment.

d. La cellule du code barre

Notre carte de fidélité maintenant scannée, ses informations récupérées au sein des *NSUserDefaults*, il nous faut maintenant l'afficher au sein d'une cellule dédiée, qui doit se situer entre nos deux précédentes.

Parmi les demandes du *Product Owner*, ce-dernier demandait à ce qu'une cellule soit ajoutée après que l'utilisateur ait scanné sa carte de fidélité, affichant, au sein de la vue constituant ladite cellule, le nom du magasin sélectionné, un dessin représentant le code barre de la carte de fidélité et le numéro de cette-dernière.

Dans la conception graphique, j'ai continué dans à entièrement programmer la vue. A ce niveau, il est important de noter que, du fait d'une utilisation de plusieurs cellules totalement différentes dans leur conception, l'utilisation du storyboard, bien que possible, ne me semblait pas être la plus permissive et m'apparaissait comme la plus complexe dans sa gestion et sa mise en place, à contrario de la création par le code des cellules.

Afin que celles-ci soient utilisées dans la collection de vues, je dû les enregistrer au préalable, via une fonction native aux collections. Ceci fait, il me fut loisible de sélectionner la classe de cellule à afficher, en fonction de la valeur d'une chaîne de caractères d'un tableau. Ce-dernier étant la base permettant la construction de la collection, via des méthodes spécifiques aux collections, précisant le nombre de cellules, leur taille, leur contenance, etc. D'ailleurs, si avant tout scan de carte de fidélité, notre tableau ne comporte que deux valeurs, lorsqu'une carte de fidélité est scannée, en revenant sur la vue du menu principal, je recharge la collection de vues présentée ici dans les méthodes du cycle de vie de la vue, tout en ajoutant, au sein du tableau de référence, une chaîne de caractères propre à la cellule du code barre. A la fin du traitement, nous disposons, non plus de deux, mais bel et bien de nos trois cellules attendues.

5.3. Choisissons notre magasin

Cette fonction de l'application présente à l'utilisateur une liste de tous les magasins enregistrés en base de données. Cette liste peut être filtrée via une sélection de rayons de recherches. L'utilisateur peut sélectionner le magasin dans lequel il se rend pour faire ses courses à partir de ce menu. Le choix du magasin est primordial pour permettre à l'utilisateur de pouvoir régler le contenu de son panier à la fin du parcours de courses.

a. Mes Interfaces

A ce niveau de développement, et tandis que j'ai beaucoup gagné en assurance et en connaissances techniques, je vins à m'interroger sur la manière dont je pouvais encore améliorer mon code et sa lisibilité.

Reprenant en exemple mes précédentes réalisations d'API en .NET sur mon temps personnel et à la suite de discussions avec plusieurs de mes collègues, je choisis la voie de l'interface. Le but de ces interfaces étant de simplifier l'utilisation du code, mais également sa maintenance et son interopérabilité en cas d'évolution ultérieures.

Appliquant ces principes, je réalisais la sélection du magasin en instanciant des protocoles, puis des méthodes issues desdits protocoles. Le but ici est d'avoir le code le plus lisible et maintenable possible, mais également de pouvoir l'améliorer dans l'avancement du projet.

b. Mes Dépôts

Mes méthodes déclarées au sein de leurs protocoles respectifs, il me fallut alors les implémenter. Pour cela, je codais des dépôts spécifiques et des contrôleurs dédiés que sont les dépôts de management de localisation et d'items de magasin.

b.1. Le dépôt de management de la localisation

Héritant du protocole de management de la localisation, ce dépôt permet d'implémenter toutes les méthodes déclarées dans le protocole et donc d'être en mesure de calculer une distance en mètres séparant l'utilisateur et un magasin, grâce à la géolocalisation. L'import du module natif *MapKit* fut nécessaire pour avoir accès à toutes les méthodes natives de SWIFT.

Quant à la géolocalisation de l'appareil, au niveau des produits Apple, en tant que développeur nous sommes tenus d'argumenter sur la nécessité d'utiliser ces dispositifs, du fait de la possible intrusion dans des données sensibles de la clientèle de la société Apple. Par ailleurs, lors de la programmation de l'application, je décidais, pour une mise à jour régulière, tant que l'application affichait la vue de sélection des magasins, de la position géographique de l'iPhone. Le dépôt d'items de magasin

Héritant, pour sa part, du protocole d'items de magasin, cette classe implémente plusieurs méthodes servant à récupérer les données concernant les magasins depuis la base de données, via les *Core Data*, mais également à créer des objets du type modèle d'items de magasins.

Une des méthodes implémenter dans cette classe sert notamment à préciser, via une récupération de l'heure, si un magasin est noté comme étant ouvert ou fermé, et à mettre à jour la cellule dudit magasin dans la collection de vues de magasins, notamment le label « Ouvert » / « Fermé ».

c. Le contrôleur de vue mon Magasin

Ce contrôleur de vues gère l'affichage des éléments graphiques pour la vue de sélection des magasins. Ainsi, j'y ai placé deux collections de vues, indépendantes, affichant pour l'une les distances de recherches, à l'horizontale, tandis que la seconde présentée verticalement, affiche les cellules des magasins contenus en base de données. De plus, afin de permettre d'effectuer une recherche manuelle, un champ de texte a été placé en haut de l'écran, offrant à l'utilisateur la possibilité de saisir directement le magasin spécifique qu'il recherche.

Pour un fonctionnement optimal, j'implémentais, là encore, un patron de conception du type Observateur. En effet, la recherche par sélection de rayon, doit pouvoir se lancer lorsque l'utilisateur sélectionne une des distances proposées.

d. La collection de vues de sélection de la distance

Comme présenté précédemment, cette collection de vues présente à l'utilisateur une suite de cellules constituées d'un texte donnant une distance de recherche, en mètres, pour filtrer la liste des magasins.

Afin de remplir son objectif, la collection est construite à partir d'un tableau de chaînes de caractères. Durant la boucle qui instancie nos cellules, chacune de ces chaînes de caractères servira à créer le texte à afficher dans la vue contenue dans la cellule.

Quant à l'Observateur, j'ai placé une méthode notifiant de la sélection d'une distance de recherche. Cette notification appelle alors la méthode se trouvant dans le contrôleur de vue des magasins. Ce-dernier va alors recharger la collection de vues des magasins, prenant en paramètres la distance de recherche.

Avant le rechargement, un calcul de distance et une comparaison est réalisée pour chaque magasin. En effet, après une remise à zéro du tableau servant à la création des cellules de magasin, on réalise une boucle qui vient mesurer la distance entre le magasin et la position de l'iPhone, puis une comparaison avec le rayon de recherche. Si la distance est égale ou inférieure à ce rayon, le magasin est alors ajouté au tableau, sinon, il est ignoré et la boucle passe à la prochaine itération.

Enfin, le tableau de modèles de magasins prêt et après avoir fini de boucler, la collection de vues de magasins est rechargée et affichée, mise à jour, à l'utilisateur qui pourra sélectionner le magasin qu'il souhaite ou relancer un filtrage sur une nouvelle distance.

d.1. La cellule de distance

Également de conception classique par rapport aux autres cellules de ce projet, celle-ci, néanmoins, diffère en ce que celle-ci est directement liée au storyboard. En effet, ayant pu déjà bien travailler sur la conception sans aide du storyboard, je souhaitais faire une comparaison entre ces deux méthodes, afin de déterminer dans quels cas l'une est plus appropriée que l'autre.

De fait, dans le code, je n'eut pas à gérer le placement, ni les contraintes graphiques. Je pus entièrement me concentrer sur la mise en forme des éléments graphiques, afin de rester conforme aux maquettes graphiques.

e. La collection de vues de magasins

Cette collection de vues permet uniquement l'affichage des cellules de chaque magasin contenu dans la base de données embarquée.

Aussi, à partir d'un tableau de modèles d'item de magasin, la collection instancie une cellule pour chaque magasin contenu dans ce tableau, à partir de la cellule de magasin. Cependant, une problématique se fit jour, avec l'obligation de gérer le cycle d'ouverture-fermeture des magasins. En effet, dans chaque cellule, un texte doit permettre à l'utilisateur, d'un rapide coup d'œil, si ledit magasin est ouvert ou fermé, en fonction des horaires d'ouvertures entrés en base de données. C'est par l'utilisation d'une méthode présentée précédemment et se trouvant dans le dépôt d'item de magasin que ce calcul horaire est effectué. A partir du booléen retourné par cette méthode, il nous est alors possible de changer la valeur et la couleur du texte, respectant les spécifications techniques à ce sujet.

Notons que lors de l'initialisation de cette collection de vues, une méthode vient récupérer l'heure de l'iPhone et l'utilise par la suite dans notre dépôt et nous retourner le booléen nécessaire à un affichage optimal de toutes les informations du magasin présenté dans cette cellule.

e.1. La cellule de magasin

Cette cellule vient présenter les informations du magasin que nous souhaitons voir apparaître et de manière conforme aux maquettes graphiques et spécificités techniques.

Le label concernant l'état du magasin est le seul élément graphique qui vient changer dynamiquement, comme expliquer précédemment. Pour le reste, les éléments sont statiques et placés grâces aux côtes fournies par les maquettes.

e.2. Le modèle d'item de magasin

L'objet *StoreItemModel* est un objet composé de seulement deux propriétés : *store* et *isOpen*.

Ce modèle se compose de deux propriétés :

- Un magasin (*store*)
- Un booléen est-ce ouvert (*isOpen*)

L'intérêt de contenir une propriété du type Magasin (*Store*) est de pouvoir la travailler plus aisément, particulièrement lorsqu'il me faut y intégrer de nouveaux éléments calculés ou non, à l'exemple du booléen est-ce ouvert (*isOpen*).

6). La collection de vues des coupons de réduction

Ajoutant une nouvelle collection de vues dans notre collection principale du menu d'accueil, il s'agissait pour moi de permettre d'afficher les coupons de réduction, non-utilisés, disponibles à l'utilisateur.

La demande précise que ces coupons doivent être animés et donc présentés sous la forme de fichiers GIF (Graphics Interchange Format). Or, ne disposant pas nativement des méthodes nécessaires à leur emploi et leur animation, je dû résoudre cette première difficulté.

Je trouvais un *cocoapod*, une bibliothèque spécialement dévolue aux solutions iOS, qui permettait la manipulation des fichiers GIF. Et, notamment, le lancement et l'arrêt de leur animation.

L'affichage des cellules, contenant chacune une image, se fit à l'horizontale, de manière à ne pas alourdir le menu principal et le rendre instinctif dans son utilisation. Aussi, afin de faire comprendre instantanément que cette collection est composée de plusieurs images, la taille des cellules fut calculée pour afficher entièrement une première image, tout en laissant entrevoir la suivante sur le bord de l'écran.

Notons que pour des raisons de performances, seule l'image entièrement affichée est animée, les autres sont inertes et ne s'activent qu'une fois qu'elles sont affichées pleinement à l'écran. En disparaissant, elles sont retournées à un état inerte.

La liste de coupons disponibles est faite de manière aléatoire, afin de représenter le compte d'un client ayant déjà utilisé l'application, montrant alors aux futurs clients la manière dont ces éléments s'affichent à l'écran.

Attention, tous les coupons ne sont pas représentés ; En effet, certains ne s'ajoutent à la liste qu'uniquement à l'ajout de certains articles dans le panier, durant une session de courses. Dès lors, ces coupons ne peuvent être affichés que dans le cas d'un utilisateur connecté. Dans le cadre contraire, une vue simple invitant l'utilisateur à se connecter apparaît à l'écran.

6.1. La cellule de coupon

Cette cellule de collection de vues fut la plus simple à créer. Elle ne comporte qu'un uniquement élément qui est une vue. Celle-ci permet d'accueillir une image donnée à partir d'un nom et d'une extension. Une méthode permet alors de retrouver cette image au sein de nos ressources.

Là encore, la création fut programmatique. Bien que l'emploi du storyboard eut pu être une solution tout à fait viable, je désirais pousser au maximum la création programmatique d'éléments visuels, afin de gagner toujours plus en compétences.

6.2. Le modèle de coupon

Afin de respecter le principe du Modèle-Vue-Contrôleur, je créais un modèle de coupon. Ce modèle me permit d'assembler dans le même objet tous les éléments nécessaires à la création d'une vue de coupon et de passer celle-ci à la cellule.

6.3. La vue de coupon

Disposant de la cellule et d'un modèle, je créais une classe pour la vue de coupon.

Cette classe, ne disposant d'aucune logique, n'a pour unique but que de créer la vue à ajouter à la cellule lors de l'instanciation de celle-ci au sein de la collection. Et, une fois n'est pas coutume, c'est via le code que cette vue fut créée.

Afin de rendre la vue la plus douce et plaisante aux yeux, possible, je devais arrondir les angles. Parmi les propriétés natives d'une vue dans iOS, il nous est possible de travailler les angles aux coins, ainsi que d'ajouter une bordure par exemple.

6.4. La cellule de coupon déconnecté

Le principe sous-jacent à l'utilisation des coupons de réduction est qu'ils sont personnels. C'est-à-dire qu'ils ne peuvent être accessibles que dans l'unique cas où notre utilisateur s'est identifié et connecté par le scan de sa carte de fidélité.

Ainsi, je dû créer une vue simple et unique, à afficher dans le cas d'un utilisateur non-connecté. Travaillée de la même manière que ses prédécesseurs, toute la création s'est faite de manière programmatique et n'est composée qu'une unique vue, la vue de coupon déconnecté.

6.5. La vue de coupon déconnecté

Toujours dans le but de rester conforme aux maquettes, cette vue a donc été réalisée sans l'aide du storyboard. Simple dans sa conception, il s'agit de n'afficher qu'un message invitant l'utilisateur à se connecter pour visualiser ses coupons de réduction, avec un label dessiné en forme de bouton aux coins arrondis.

Si j'ai préféré le label au bouton, c'est dans l'idée d'une simplification de l'interaction de la cellule dans son ensemble. En effet, plutôt que de ne cibler qu'uniquement un bouton pour permettre d'afficher la vue d'ouverture du scanner, permettant à l'utilisateur de scanner sa carte de fidélité et, donc, de se connecter, j'ai choisi de rendre cette action possible dès lors que l'on appuie sur la cellule. Le label, n'ayant qu'un usage informationnel était donc plus adapté à la situation.

7). La collection de vues des recettes et les cellules de types de recettes

Ici, nous regroupons deux cellules distinctes, mais interagissant ensemble, à propos de l'affichage des recettes de cuisines : le *RecipesCollectionView* et le *RecipesSortCell*.

Intimement liées ces deux collections de vues ont pour but d'afficher des icônes de recettes de cuisine, en fonction du choix de l'utilisateur. Chaque recette est enregistrée au sein de notre base de données embarquée et est fournie sous forme d'un objet créé automatiquement par SWIFT à partir d'une entité et de ses attributs.

Le but de l'interaction de ces deux collections est d'offrir à l'utilisateur un filtrage des recettes de cuisine par type, en sélectionnant ce-dernier parmi une liste de choix, à l'écran d'accueil.

7.1. La cellule de type de recette

A partir d'une liste de mots-clefs fournie sous forme de chaînes de caractères, je suis venu travailler leur mise en place au sein d'une collection de vue présentée à l'horizontale et pouvant être parcourue librement par l'utilisateur.

Les mots-clefs sont les suivant :

- Tout
- Entrées
- Plats
- Desserts
- Apéritif

Pour chaque cellule, l'utilisateur peut entrer en interaction. Cependant, puisqu'il s'agit de cellules, j'eus à leur ajouter un *UITapGestureRecogniezer* afin d'en faire des boutons, déclenchant la fonction de filtrage des éléments contenus dans RecipesCollectionView.

Chacune des cellules ainsi formées peut interagir avec l'utilisateur, par simple pression du doigt. Ce faisant, et grâce à un Observateur mis en place, nous venons notifier de la sélection d'un mot de clef qui va lancer un rechargement de la collection de vues de recettes depuis le contrôleur du menu principal.

Chaque mot-clef vient lancer la création d'une cellule contenant une vue simple : la vue de sorte de recette.

7.2. La vue de sorte de recette

Classe héritant de la classe générique de Vue (*UIView*), la vue de sorte de recette n'incorpore qu'un label simple dont le texte est fonction de la chaîne de caractère passée en paramètre.

Par des méthodes privées je suis venu mettre en place les règles graphiques nécessaires pour être conforme aux attendus graphiques, à savoir des coins arrondis, une couleur de texte et de fond bien spécifiques.

Une fois de plus, cet élément graphique a été réalisé sans avoir recours au storyboard, toujours dans un souci de travailler au maximum par le code.

7.3. La collection de vues de recettes

Désormais que notre collection de vues présentant les différents types de recettes est parée, il me fallut alors afficher mes recettes.

Cette étape, je l'ai décomposée en plusieurs sous-tâches, chacune relevant d'une classe particulière venant respecter le principe du Modèle-Vue-Contrôleur afin d'effectuer une bonne répartition des responsabilités.

a. Le Modèle de la vue de recette

Ce modèle nous permet d'enregistrer les informations nécessaires à l'affichage des informations d'une recette. L'objectif ici est de n'avoir qu'un seul objet à passer en paramètre et à manipuler, plutôt que de nombreuses propriétés provenant de plusieurs objets et classes différents qui viendrait alourdir et complexifier le code.

Une des principales difficultés de ce modèle fut de récupérer, à partir de la chaîne de caractères symbolisant le nom et l'extension d'une image, ladite image au sein de nos ressources visuelles. Pour cela, une méthode permettant de la récupérer fut ajoutée et renvoyait notre image.

b. La cellule de recette

Cette cellule permet l'affichage d'une vue et est réemployée par la collection de vues de recettes en fonction du nom de la recette passé en paramètre lors de la construction graphique de cette collection.

c. La vue de recette

Également créée via le code, cette vue a nécessité d'être posée au préalable sur le papier, afin de bien discerner chacun des éléments la composant, avant de les insérer dans autant de vue d'empilement (*stackview*) que nécessaires, pour assurer un bon placement et une bonne interaction entre eux.

Désormais en possession de tous nos éléments graphiques, nécessaires, je pus alors coder les relations les liant tous.

7.4. Fonctionnement du filtrage d'une recette

Comme présenté au début de ce chapitre, l'utilisateur doit pouvoir filtrer la collection de vues affichant les recettes en fonction du type de recette sélectionné. Par défaut, ce sont toutes les recettes qui sont affichées.

Pour ce faire, je découvris l'utilité et la puissance du patron de conception Observateur. En effet, en observant des changements d'état dans des classes, il est alors possible de le notifier et de réaliser des méthodes implémenter via des protocoles, les équivalences des Interfaces avec iOS, qui viennent réaliser le code qu'elles contiennent. Etant donné qu'il s'agit d'interfaces, rappelons que chaque implémentation de ces méthodes peut différer d'une implémentation à une autre.

Dans le cas présent, lorsque la notification, provenant de la collection de vues de types de recettes, était exécutée elle lançait depuis le contrôleur du menu principal le rechargement de la collection de vues de recettes, avec en paramètre le type de recette à afficher désormais. Le menu principal, et plus particulièrement la collection des vues de recettes se met aussitôt à jour n'affichant plus à l'utilisateur que les recettes du type qu'il vient de sélectionner, simplifiant ses recherches personnelles.

Ceci vient conclure notre présentation du menu principal de cette application démonstrative. Riche, varié et complet, ce menu fut une véritable opportunité d'apprentissage de nombreux concepts de codages. Ayant nécessité de très nombreuses heures de travail et de remise en cause dudit travail afin de tirer le maximum de mes connaissances de SWIFT et d'Xcode, je pu découvrir le patron de conception Observateur, mais également le principe du SOLID dans ses bases, des éléments graphiques natifs comme la collection de vues.

A ce jour, le menu principal n'est pas encore terminé. Plusieurs éléments doivent encore y être implémenter. Cependant, bien que revêtant une grande importance, il fut décidé de lancer le chantier du métier et de l'utilisation principale de cette application : le panier de courses.

8). Le panier, au cœur de *Scan&Shop*

Le panier est le véritable cœur de l'application. En effet, c'est ici que nous allons pouvoir afficher, au fur et à mesure, la liste complète des articles que nous ajoutons à notre panier lors de notre parcours client.

Notre panier est décomposé en plusieurs classes que nous allons maintenant détailler.

8.1. Le contrôleur de panier

Ce contrôleur est celui régissant la vue du panier et est lié à celle-ci via le storyboard du projet.

Ce contrôleur doit permettre de mettre en forme les éléments graphiques, ainsi que leur mise à jour en fonction de différentes conditions que nous détaillerons plus loin. Dès lors, nous allons relier à notre classe chaque élément présent sur le storyboard afin de l'instancier lors de la création de la vue.

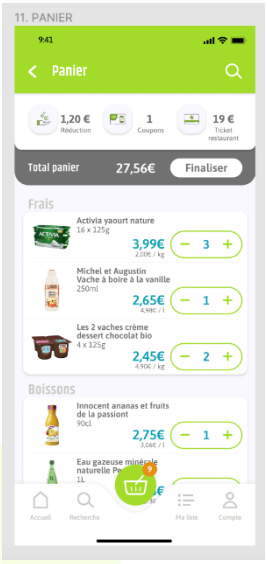


Figure 12: maquette du panier

Du fait du cycle de vie de la vue, nous allons pouvoir instancier tous nos éléments graphiques mis en place avec le storyboard. Ainsi lors du chargement de la vue on appelle les méthodes privées agissant sur l'aspect visuel, tel les coins inférieurs de l'en-tête du panier qui sont arrondis. De même, c'est à ce moment que nous instancions notre barre d'interaction (*TabBar*) situé en bas de notre écran, avec le bouton central qui permet d'ouvrir le scanner et de scanner un code barre et d'ajouter à notre panier l'article afférent.

Via des délégations, des modèles et des contrôleurs, notre panier vient être mis à jour automatiquement avec chaque ajout de produit, tenant parfaitement informé l'utilisateur de l'application sur son contenu.

Du fait qu'il s'agisse du cœur de l'application, j'ai implémenté des tests unitaires et appliqué le principe du Rouge-Vert-Refactorisation pour la création de mes méthodes contenant de la logique.

8.2. La collection de vue du panier

A la différence des éléments du menu principal, ici, j'ai décidé d'utiliser tout le potentiel du storyboard pour la création de ma vue. En effet, simple dans sa conception et ayant déjà compris les mécaniques régissant le placement des éléments avec des contraintes calculées, l'emploi du storyboard était le plus approprié dans la réalisation de notre vue de panier. Ainsi, en sus de l'en-tête de panier, j'implémentais une collection de vue, reliée à une classe spécifique, et lui attribuait directement les contraintes graphiques nécessaires.

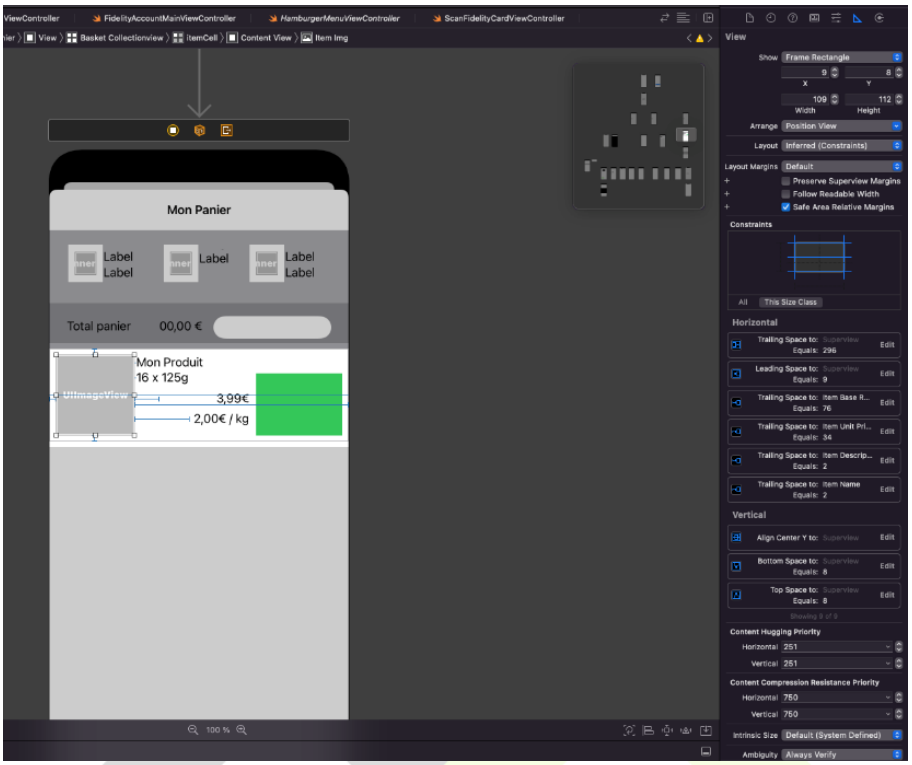


Figure 13: Elément graphique avec contraintes

L'emploi du storyboard dans ce cas, me fit découvrir alors que, si je dus enregistrer les cellules d'une collection de vues lors d'une implémentation via le code, tel ne fut pas le cas avec le storyboard.

En effet, toujours en employant le storyboard, j'insérai une cellule type directement dans ma collection de vues, spécifiant là aussi la classe liée à cette cellule. Notons toutefois que si je n'avais pas implémenté cette cellule via le storyboard, j'aurai été dans l'obligation d'enregistrer cette cellule via le code. De même si je souhaite ajouter une cellule non créée avec l'aide du storyboard.

8.3. Le modèle d'item du panier

Afin de simplifier la manipulation des items composant mon panier, je procédais à la création d'un modèle que je nommais le modèle d'item du panier (*BasketItemModel*). Ce modèle offrit la possibilité de regrouper au sein d'un même objet plusieurs propriétés calculées ou provenant d'autres classes différentes. Mon modèle n'étant composé que des trois propriétés suivantes :

- _item
- Quantité (*quantity*)
- Nom de la section (*sectionName*)

Tout en restant simple, ce modèle offre pourtant une large couverture des besoins d'informations à afficher dans le panier, comme le nom, le prix, ou encore une courte description provenant de la base de données et regroupés dans une entité *Article*. Mais,

en y adjoignant la quantité ici, il m'est alors possible de la changer plus facilement et de sauvegarder le tout après modification et avant affichage à l'écran pour garder informer l'utilisateur sur le contenu réel de son panier.

La propriété de nom de section récupère, depuis une autre entité, Catégorie, en relation avec l'entité d'*Article* et doit permettre un filtrage par catégorie de produits les articles contenus dans le panier. Ceci n'a pas encore été programmer au moment où ces lignes sont rédigées.

8.4. Les données d'item du panier

Cette classe permet l'exploitation et la tenue à jour des informations affichées dans l'en-tête du panier. Elle est nécessaire à son bon fonctionnement.

En effet, nous faisons face à une importante problématique qui est celle de toujours afficher les bonnes informations. Dès lors, et à partir du contenu du panier, cette classe, agissant comme un modèle est notamment composé des méthodes permettant de calculer le prix total ou encore le nombre de coupons de réductions disponibles, avant de renvoyer ces informations directement à l'en-tête du panier. N'oubliant pas d'adjoindre au coût total du panier les différentes réductions s'appliquant.

9). Les tests unitaires

Défini comme objectif annuel, mais également afin de gagner en compétences professionnelles, j'eus l'occasion d'intégrer quelques tests unitaires au sein de Scan&Shop.

Malheureusement, une importante difficulté se fit rapidement jour : le peu d'aide disponible pour implémenter des tests unitaires en usant de SWIFT, que ce soit en entreprise ou sur internet.

En effet, le seul cours abordant le sujet, je le trouvais sur le site d'Openclassrooms, parmi les cours de la formation de développeur iOS. Pour autant, et en m'appuyant sur ledit cours, je parvins à implémenter des tests simples et à commencer à aborder le principe du Rouge-Vert-Refactorisation (en anglais : *Red-Green-Refactor*), méthode au principe très simple : débuter la programmation d'une classe par un test unitaire qui va, fatalement, échouer, puisque la fonction testée n'existe pas encore. Puis on implémente la méthode de manière que le test devienne bon, puis on étoffe ce-dernier pour tendre vers le bon résultat à retrouver à l'issue des tests et des différentes refactorisations.

Cette méthode, quoique longue à appréhender et à utiliser, offre néanmoins la possibilité de créer du code robuste et efficace.

Du fait des tickets réalisés, je fus également limité quant à l'implémentation de tests unitaires, car j'eus à travailler sur beaucoup d'éléments graphiques et de création graphiques, et donc peu de logique à tester unitairement.

Cependant, je pus implémenter des tests unitaires avec le panier. M'offrant une première approche et de premiers essais fructueux, bien que difficiles.

Conclusion

Alors que s'achève cette formation de Développeur Informatique, il me faut maintenant en dresser la conclusion

C'est donc après un contrat de dix ans en tant que sous-officier de la Marine Nationale, spécialisé dans la Conduite des Opérations et l'emploi des radars et sonars que j'ai décidé de changer mon orientation professionnelle.

Sans connaissance aucune lorsque je découvris le CESI en février 2020, je découvris une affinité avec la programmation orientée objet et le monde de l'informatique logicielle avec le test d'entrée pour la formation de Développeur Informatique.

Recruté au sein des équipes de développement de Budgetbox, c'est ici que je vis plus en profondeur ce monde et celui de l'iOS et de ses spécificités. Ainsi que je vous le présentais, Budgetbox est une entreprise proposant des solutions pour améliorer le parcours achat en magasins, tout en offrant aux enseignes la possibilité de distribuer des coupons de réduction et ainsi, fidéliser au mieux ses clients ou encore en conquérir de nouveaux. Mais, c'est au travers d'un projet de création d'application native mobile, complète, que je pus avoir une montée en compétences fortes et appréhender de manière plus fine la tenue d'un projet, sa programmation. Enfin, au travers du processus de codage, je découvris pleinement, et non sans défis à relever, la programmation orientée objet, l'univers iOS et le langage phare d'Apple qu'est SWIFT.

Bien que ce projet ne pût être entièrement terminé, car je devais apprendre un langage de programmation, entièrement et sans la possibilité d'une aide conséquente en entreprise, en sus de changements dans le calendrier, mais encore la découverte d'éléments iOS me facilitant la programmation et qui m'obligèrent à remettre à plat mon travail, le repenser et à le reprendre pour obtenir le meilleur résultat possible.

L'apprentissage de l'environnement iOS se compléta à merveille avec l'enseignement dispensé au CESI. En effet, je pus alors découvrir d'autres technologies, à l'instar de Javascript, HTML, C# ou encore Flutter pour ne citer que celles-ci. Au travers des projets notés, je peaufinais mon apprentissage, revenant sur mon travail et cherchant, constamment, à obtenir le meilleur résultat. Face à ces nombreux défis, l'application d'une grande discipline et d'une bonne tenue d'un calendrier et de méthodes de travail, franchement inspirées de la méthodologie SCRUM, me permirent de pousser plus loin les attendus.

Cette même discipline, je l'appliquais sur mon temps personnel lorsque je souhaitais toujours améliorer mes connaissances, via des cours dispensés sur Internet, sur des plateformes telles *Openclassrooms* et *Udemy*. C'est ainsi que je travaillais le C# et Flutter sur de petits projets personnels, par exemple une API ou encore une application de déménagement. Je dus apprendre comment rédiger un cahier des charges et des spécifications techniques, à me projeter et imaginer mes applications à réaliser, pour ensuite en rédiger les tickets de projet.

Au cours de ces deux années, jamais je ne fus dans une quelconque « zone de confort » puisque j'ai continuellement cherché à en repousser les limites, par un travail régulier. Ne pouvant me satisfaire du minimum, je n'ai jamais hésité, durant cette formation, à sortir des sentiers battus, ni à devoir emprunter des chemins plus difficiles, dès lors que l'objectif que je m'étais fixé, gagner en compétences, était atteint. Et ce, malgré des conditions de travail qui furent, parfois, extrêmement difficiles et pénibles, notamment avec la perte d'une personne très proche.

Pour autant, ces deux années d'apprentissages m'ont conforté dans mon choix de réorientation et de poursuivre mon apprentissage. En achevant aujourd'hui le parcours de Développeur Informatique, c'est un des plus grands défis m'ayant été présenté, que je viens de remporter. Désormais, j'ai pour objectif de capitaliser sur ces acquis pour continuer d'évoluer vers des postes à responsabilités, à l'instar de Manager en architecture logicielle.

Dès lors, je souhaite pouvoir employer au mieux toutes mes compétences en matière de savoir-faire dans la gestion d'équipe et de savoir-être, précédemment acquis lors de mes années de sous-officier de la Marine, en les additionnant à celles, plus techniques, reçues et travailler au CESI et au sein de Budgetbox. La planification et la conduite de projets, en équipe structurée, présentent un attrait certain, formant le nouveau but que je me suis fixé pour les années à venir.

Ainsi s'achève ce rapport, je vous remercie de m'avoir lu et d'avoir pris le temps d'en apprécier la qualité.

Je vous prie d'agréer, madame, monsieur, l'expression de mes salutations distinguées et espère pouvoir échanger avec vous prochainement.

« Mais soldats, vous n'avez rien fait, puisqu'il vous reste encore à faire. »

Napoléon Ier, proclamation à l'Armée, le 26 avril 1796

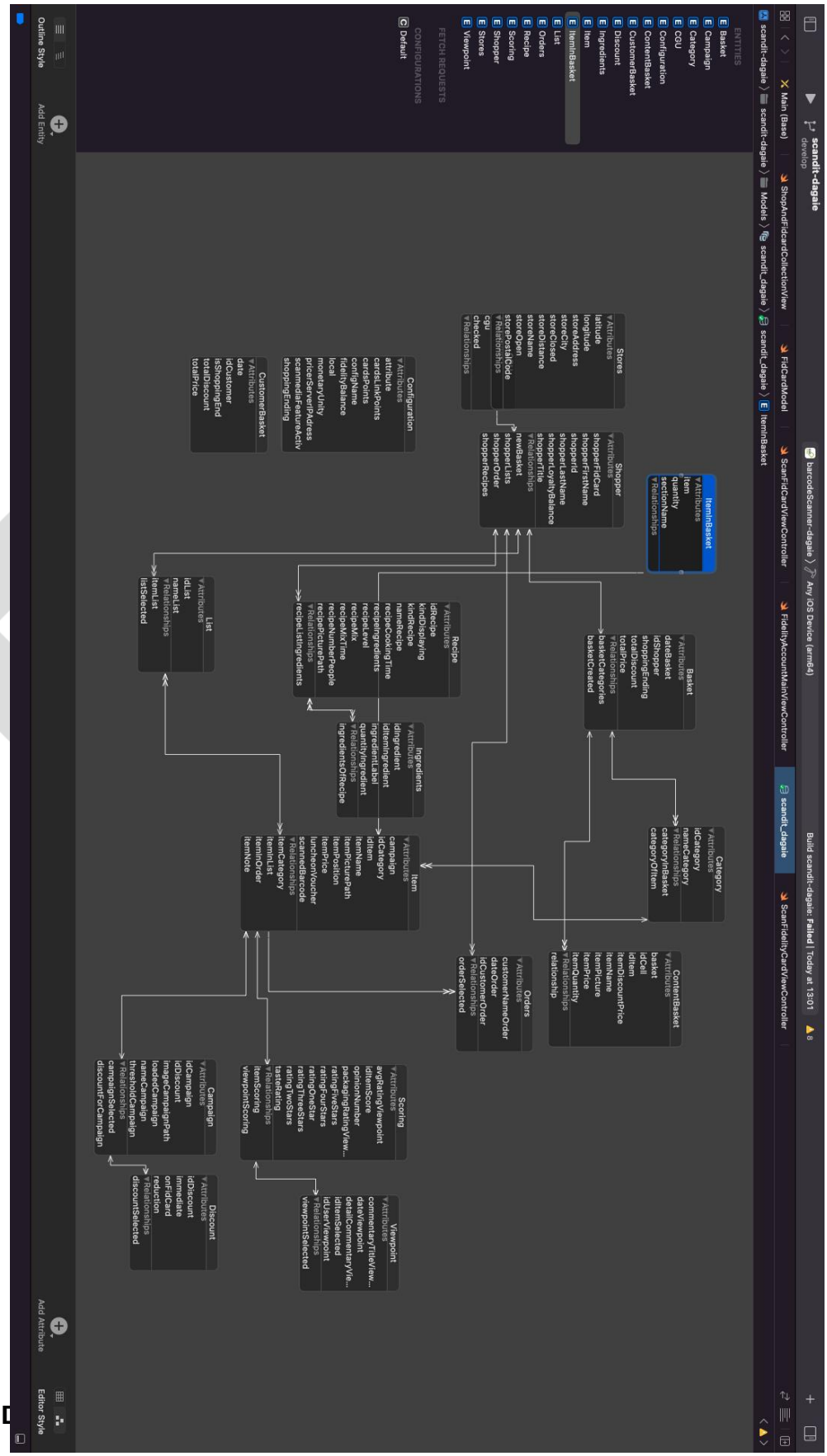
M. FICHOU Kevin

Table des illustrations

Figure 1: Logo de l'entreprise..... 8
Figure 2: la tenue d'un calendrier est primordiale 14
Figure 3: Un design simple et épuré 17
Figure 4: le sigle d'Xcode 20
Figure 5 : Sigle de SWIFT..... 20
Figure 6: supprimer les files d'attente, un enjeu crucial.. 23
Figure 7: supprimer les files d'attente, un enjeu crucial.. 23
Figure 8: Exemple d'une entité et de ses attributs 27
Figure 9: Exemple d'entités sous Core Data..... 29
Figure 10: Collection de vues choix magasin et scan carte de fidélité.. 33
Figure 11propriétés de parentage, dépôt et contexte..... 36
Figure 11: maquette du panier..... 45
Figure 12: Elément graphique avec contraintes..... 46

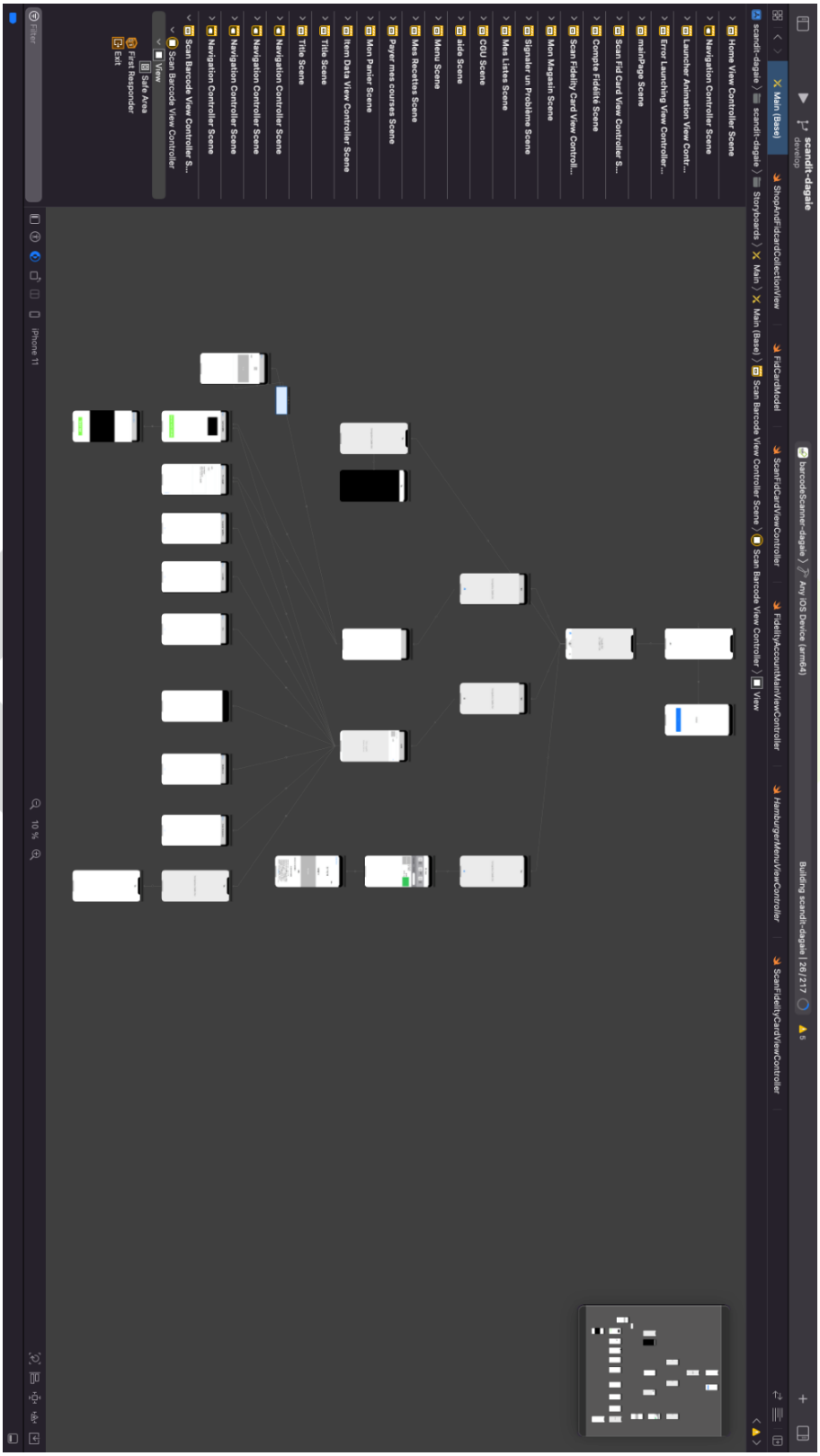
Annexe 1

Base de Données de l'application



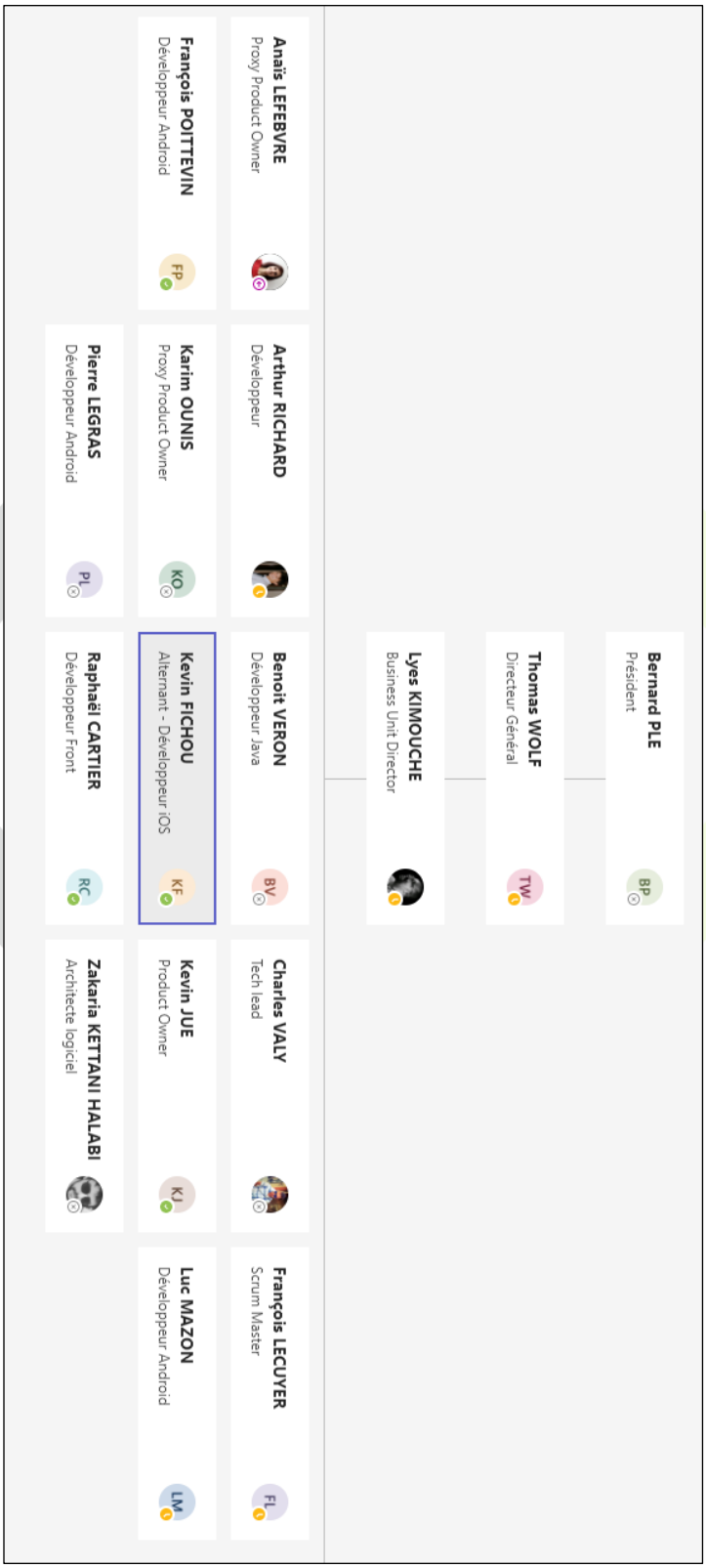
Annexe 2

Storyboard de l'application



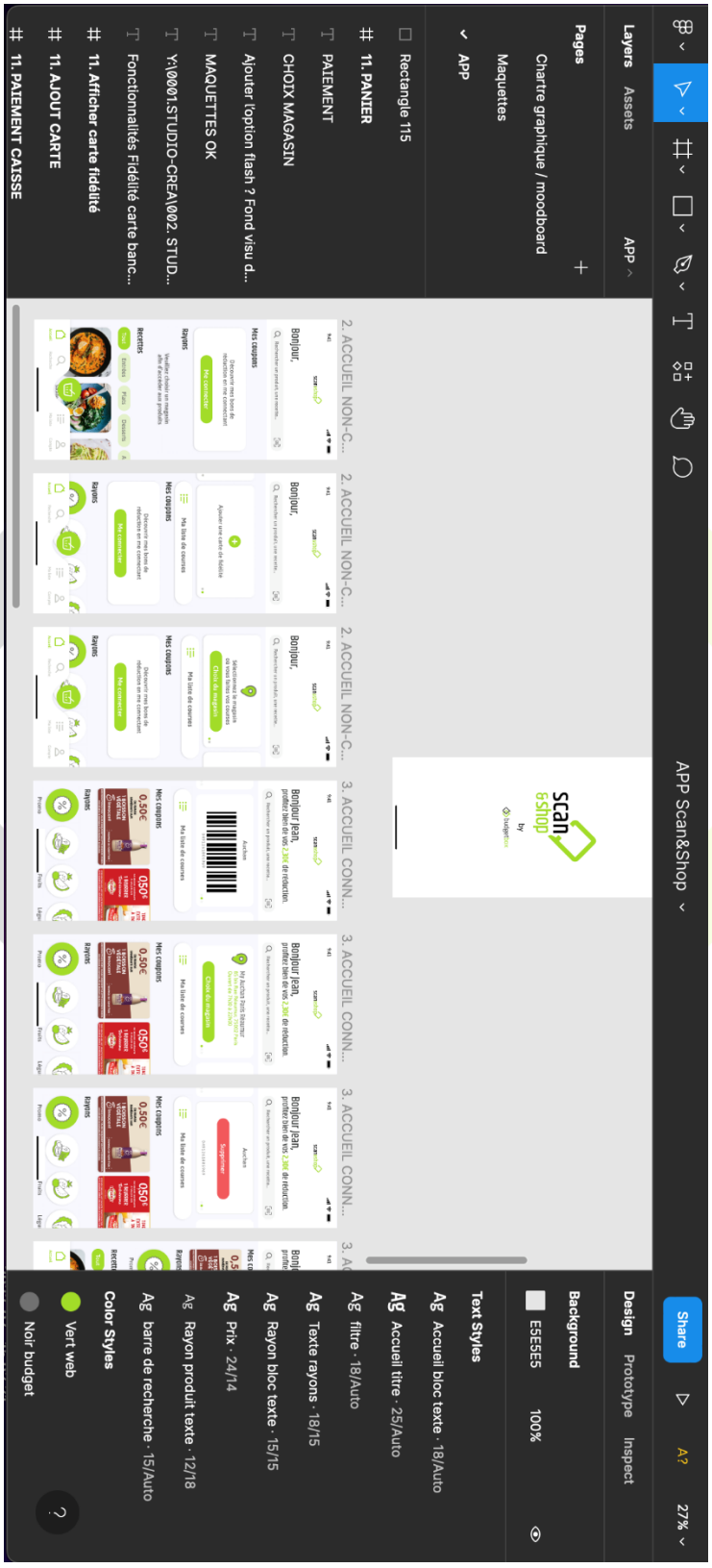
Annexe 3

Organisation du pôle *Connected-Shopper*



Annexe 4

Illustration maquettes graphiques FIGMA



Annexe 5 :

Récapitulatif des entreprises par le site *Antvoice*