Rapport TPA : ToBuy

Luca Bohlen – Cin4B

ETML

89.9 heures

Chef de projet : Gilbert Gruaz

Experts : Xavier Carrel et Carlos Perez

Table des matières

[1 Analyse préliminaire 3](#_Toc72995547)

[1.1 Introduction 3](#_Toc72995548)

[1.1.1 Titre 3](#_Toc72995549)

[1.1.2 Description 3](#_Toc72995550)

[1.1.3 Contexte 3](#_Toc72995551)

[1.2 Objectifs 4](#_Toc72995552)

[1.3 Planification Initiale 5](#_Toc72995553)

[2 Analyse / conception 6](#_Toc72995554)

[2.1 Concept 6](#_Toc72995555)

[2.1.1 Méthodologie de projet 6](#_Toc72995556)

[2.1.2 Interface graphique / Plan de liaison 7](#_Toc72995557)

[2.1.3 Base de données 12](#_Toc72995558)

[2.2 Stratégie de test 12](#_Toc72995559)

[2.3 Risques techniques 14](#_Toc72995560)

[2.4 Planification détaillée 15](#_Toc72995561)

[2.5 Dossier de conception 16](#_Toc72995562)

[3 Réalisation 16](#_Toc72995563)

[3.1 Dossier de réalisation 16](#_Toc72995564)

[3.2 Description des tests effectués 17](#_Toc72995565)

[3.3 Erreurs restantes 19](#_Toc72995566)

[3.4 Liste des documents fournis 19](#_Toc72995567)

[4 Conclusions 19](#_Toc72995568)

[4.1 Bilan des fonctionnalités demandées 19](#_Toc72995569)

[4.2 Bilan de la planification 20](#_Toc72995570)

[4.3 Bilan personnel 20](#_Toc72995571)

[5 Annexes 20](#_Toc72995572)

[5.1 Résumé du rapport du TPI / version succincte de la documentation 20](#_Toc72995573)

[5.2 Webographie 20](#_Toc72995574)

[5.3 Planification Initiale 20](#_Toc72995575)

[5.4 Journal de travail 20](#_Toc72995576)

[5.5 Manuel d’installation 20](#_Toc72995577)

[5.6 Manuel d’Utilisation 20](#_Toc72995578)

[5.7 Archives du projet 20](#_Toc72995579)

# Analyse préliminaire

## Introduction

Ce projet consiste à réaliser une application mobile IOS dans le language de programmation SWIFT. Cette application permettra à des utilisateurs de créer une liste de courses et d’optimiser celle-ci à l’aide de modèles de magasin. Le principe est de simplifier les déplacements dans un magasin traditionnel.

Ce sujet a été choisi avec le chef de projet (M. gilbert Gruaz) lors de notre première entrevue. Un ordinateur est fourni par l’ETML avec les logiciels standards permettant à l’élève de travailler dans des bonnes conditions. L’élève a aussi comme prérequis suivi la formation d’informaticien en entreprise à l’ETML et a suivi les cours du module 335, développement d’application mobile.

Au préalable, une analyse des magasins Migros, Coop et Aldi a été effectuée. En effet, 3 magasins de chaque marque citée ci-dessus ont été visité par l’éléve afin d’évaluer la cohérence des emplacements des rayons afin d’établir un modèle réaliste de ces derniers. Les résultats vont être expliqués plus en détails dans ce rapport et vont être implantés dans l’application en tant que modèle.

### Titre

ToBuy, Application IOS

### Description

Il s'agit de réaliser une application capable de fonctionner sur un iPhone, qui doit permettre aux utilisateurs de préparer une liste de commissions en ajoutant des produits à acheter comme dans un brainstorming, mais qui sera présentée selon l'organisation d'un modèle de magasin. La même liste apparaitra différemment si les achats se font à la Coop, à la Migros, Aldi, etc. Le principe doit permettre d'optimiser le temps passé dans le magasin, en regroupant les produits par rapport à l'achalandage décidé par le magasin.

### Contexte

Comme TPI de fin de 4ème année, l’éléve doit realiser un travail personnel sur la base d’un cahier des charges reçu le 1er jours. Le cahier des charges est approuvé par les deux experts. Il est en outre présenté, commenté et discuté avec le candidat. Par sa signature, le candidat accepte le travail propos.

Le thème du sujet ainsi que le chef de projet a été choisi par l’élève lors du pré-TPI (10 février 2021).

## Objectifs

L’éleve est responsable de livrer, à son chef de projet et aux deux experts, les éléments de suivi de projet suivants :

* Une planification initiale (jour 1) et une planification aussi détaillée que possible (jour 3)
* Un tableau avec des modèles de magasins (au moins trois)
* Un rapport de projet
* Un journal de travail
* Les instructions pour la mise en œuvre de l'application, ou à une simulation de celle-ci.

En plus de cela, une applications mobile IOS fonctionnel comportant les 7 points spécifiques suivant qui seront évalués (Point A14 à A20) :

1. Mise en œuvre d'une fonctionnalité qui permet à l'utilisateur de définir un profil de magasin (fédération de produits selon la topologie du magasin).
2. Mise en œuvre d'une fonctionnalité qui permet à l'l'utilisateur de choisir un modèle de magasin dans lequel il va faire des achats.
3. Mise en œuvre d'une fonctionnalité qui permet à l'utilisateur d'inscrire une liste de commissions.
4. Mise en œuvre d'une fonctionnalité qui affiche l'ordre dans lequel les achats vont se faire selon le modèle de magasin choisi.
5. Mise en œuvre d'une fonctionnalité qui permet à l'utilisateur de décider comment traiter les produits de la liste de commissions qui ne figurent pas dans le modèle du magasin choisi.
6. Les commentaires dans le code sont clairement rédigés avant de coder, de façon à permettre une réécriture dans un autre langage comme pour Android par exemple.
7. L'utilisation, même simplifiée, correspond à quelque chose de réaliste dans un contexte réel

## Planification Initiale

Une des parties importantes de ce projet est la planification initiale et détaillée. En effet, elles me permettront d’organiser mon travail pendant les 90 heures.

Il convenu avec le chef de projet d’une planifictation initiale découpée en 4 parties visibles sur le graphique en secteur :

* La réalisation regroupe les tâches de mise en oeuvre des fonctionnalités demandées
* La documentation regroupe les tâches liés aux rapport et au journal de travail
* L’analyse regroupe les tâches liés à la lecture du CDC ainsi qu’à la documentation et recherche
* Les test regroupe les tâches de conception, réalisation et documentation des tests éffectués

# Analyse / conception

## Concept

### Méthodologie de projet

Pour effectuer mon projet, j’ai choisi d’utiliser une méthode de travail du type « en **cascade** ». Le principe de base de cette méthode est simple, découper le projet en plusieurs tâches à réaliser et à tout prix terminer une de ses dernières pour passer à la suivant. Ce qui fait sa différence avec d’autres méthodologies, c’est qu’il n’est plus possible de revenir sur une tâche effectuée lorsque celle-ci est terminée et bien évidemment, validée par le client (ici le chef de projet).

Pourquoi j’ai choisi cette méthode de travail ?

C’est parce qu’elle est appropriée à ce type de mandat. En effet, sur mon projet, je ne peux pas avancer si une des fonctionnalités ou tâches n’est pas correctement terminée et documentée.

Mais aussi car la méthode dit « en **cascade** » à un cylce de développement (en général ) qui correspond parfaitement à mon style de travail :

1. **Exigences** (Entrevue avec l’expert ainsi que le CDP)
2. **Analyse** (Analyse du cahier des charges)
3. **Conception** (Document de planification initiale/détaillée)
4. **Mise en oeuvre** (Réalisation des tâches)
5. **Validation** (La conception, la réalisation et la documentation des tests)
6. **Mise en service** (Rendu du projet avec la présenation orale)

**Mise en service**

**Validation**

**Conception**

**Analyse**

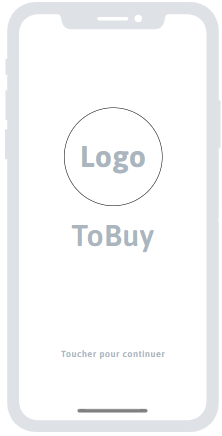
**Exigences**

**Mise en oeuvre**

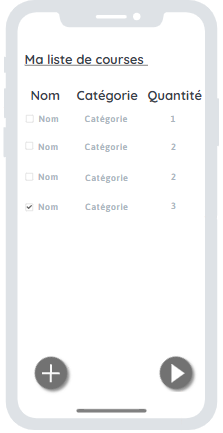
### Interface graphique / Plan de liaison

À l’aide de l’application web de création de maquette [moqups](https://moqups.com/), j’ai crée des interfaces graphiques de chaques pages presentes dans l’application ainsi qu’un plan de liaison de celles-ci. Sur ces maquettes, sont présents les boutons, les titres, les informations textuelles, les champs de textes, les comboboxs et les checkboxs. Ci-dessous, une version graphique ainsi qu’un descriptif de chacunes des pages :

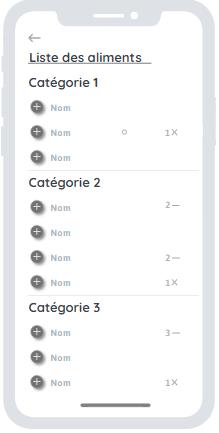
1. **Page titre** : permet "d'accueillir" l'utilisateur en affichant le logo et le nom d'application.



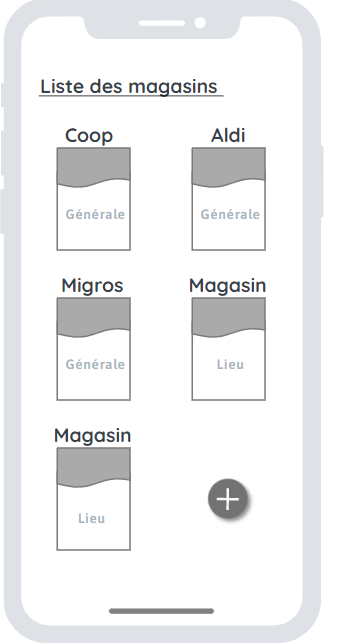
1. **Page de la liste des courses** : permet avec un bouton d'afficher la page "Ajout de produit" et avec un autre bouton de passer directement à la page "Liste des magasins".



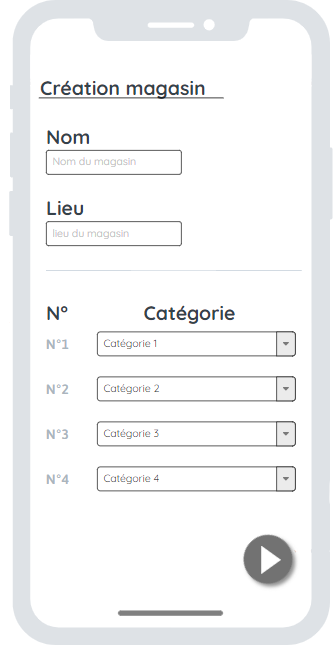
1. **Page d'ajout de produits :** permet d'ajouter ou de retirer des produits dans notre liste. Ils sont triés par catégorie, il n'y a pas de barre de recherche de prévue ainsi que la possibilité d'ajout de produit personnalisé (amélioration possible).



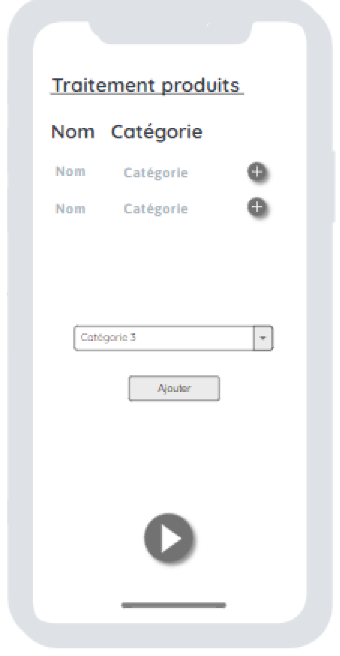
1. **Page de choix du magasin** : permet de visualiser l'ensemble des magasins. Il y un bouton d'ajout de modèle de magasin. Si l'utilisateur clique sur un magasin, il est redirigé sur la page "Ordre des courses".



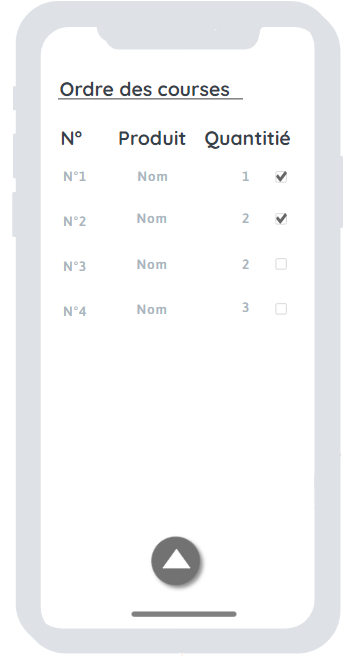
1. **Page création magasin** : permet la création d'un nouveau modèle de magasin. Il faut indiquer le nom du magasin ainsi que le lieu. Ensuite via des comboboxs, on doit désigner l'ordre des catégories de produit. Un bouton en bas de page permet d'envoyer le modèle.

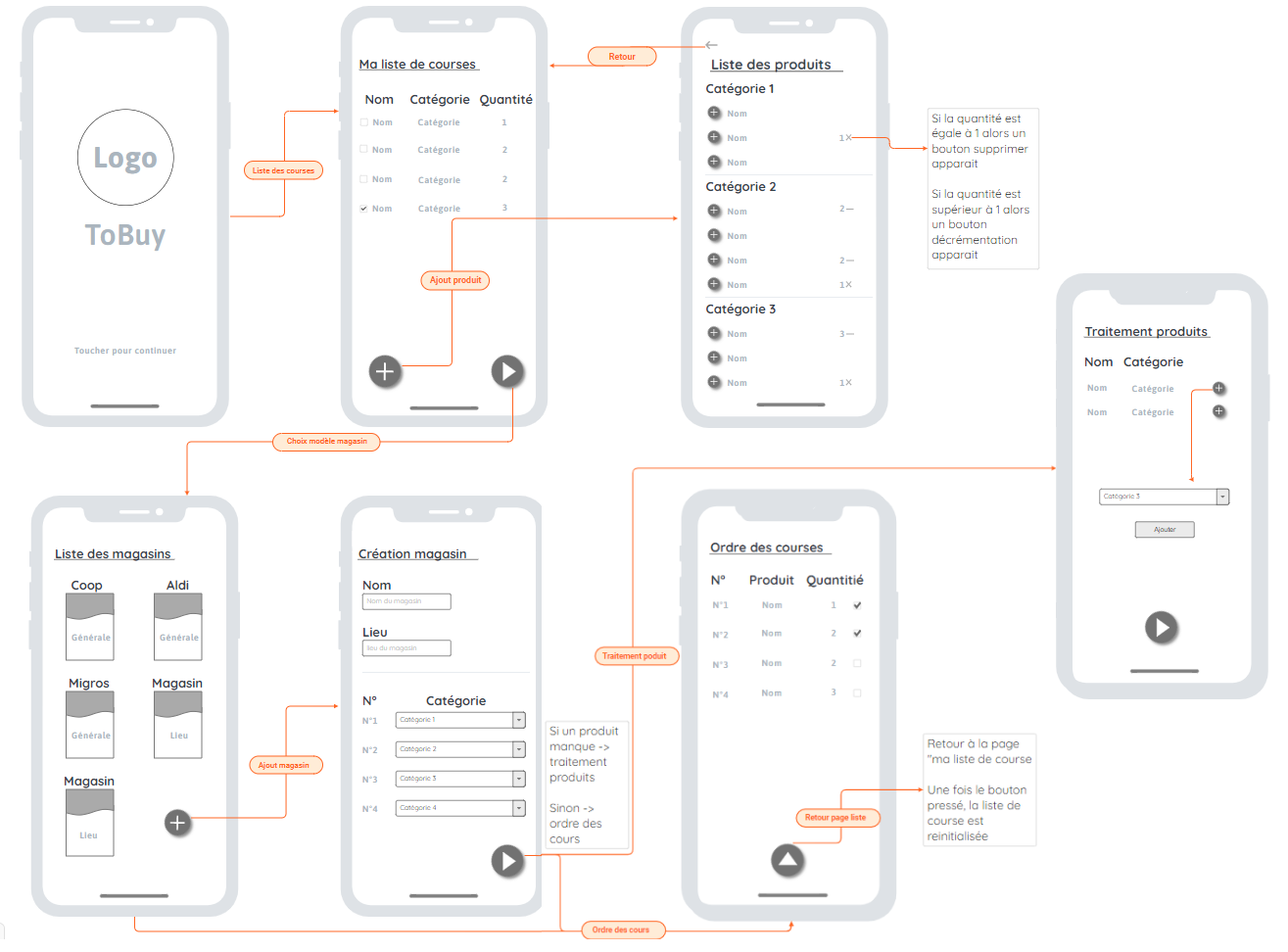


1. **Page de traitement produits** : permet à l'utilisateur de décider comment traiter les produits de la liste de courses qui ne figurent pas dans le modèle du magasin choisi. Via des comboboxs et un bouton de validation. Un bouton en bas de page permet d'envoyer les produits dans le modèle.



1. **Page d'ordre des courses** : permet à l'utilisateur de visualiser l'ordre de ses courses. Un bouton en bas de page permet de retourner à la page"liste de courses".





### Base de données Firebase

J’ai utilisé dans ce projet un ensemble de services d’hébergement, proposé par google, « **Firebase**». Il permet pour n’importe quel type d’application mobile et web (ici iOS). Il rend possible d’héberger en NoSQL et en temps réel des bases de données, du contenu et un système authentification. C’était la première fois que j’utilisais cette base de données. Je l’ai choisi car elle était la plus adapté à mon projet, c’est-à-dire une base de donnée d’utilisation simple et des requetes facilement assimilables. J’ai utilisé les services, FirebaseAuth et le Firestore.

Le Firestore est une base données NoSQL, herbergée directement sur le cloud de Google (Firebase) et orientée « documents ». En effet, le NoSQL à plusieurs « orientation » dans ce projet j’utilise la manière dites « documents ». Ce type de base de données est principalement structuré en trois document :

* La donné brute (**DATA**) : représente la donnée brute que l’on veut sauvegarder. Comme par exmemple un entier int, une chaine de caratères string, ect.
* Le document (**DOCUMENT**) représente le rattachement des données brutes. Chaque data est nécessairement rattachée à un document. Exemple, le document « viande » est composé de plusieurs champs (DATA) : name, type, idle, ect.
* La collection (**COLLECTION**) représente une sorte de liste pour les documents, une collection peut contenir un ou plusieurs documents. Suivons notre exemple, la collection « produit » contient des documents comme « viande » ou « légumes ».



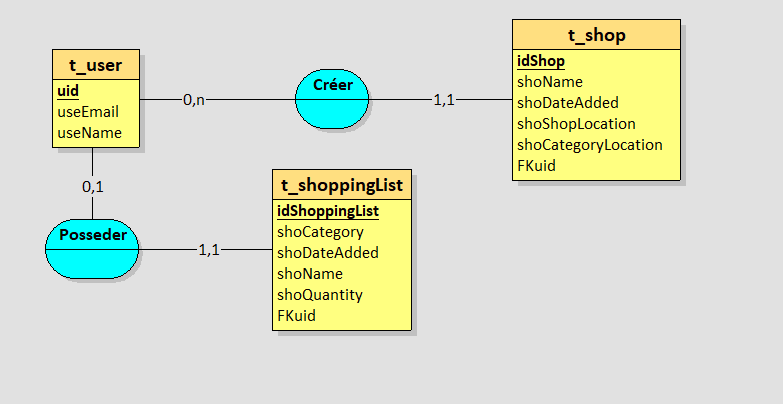
Représentation graphique du noSQL orienté documents

R

### Modèle conceptuel

Avec le logiciel, looping, j’ai créer le modèle conceptuel de la base de données.

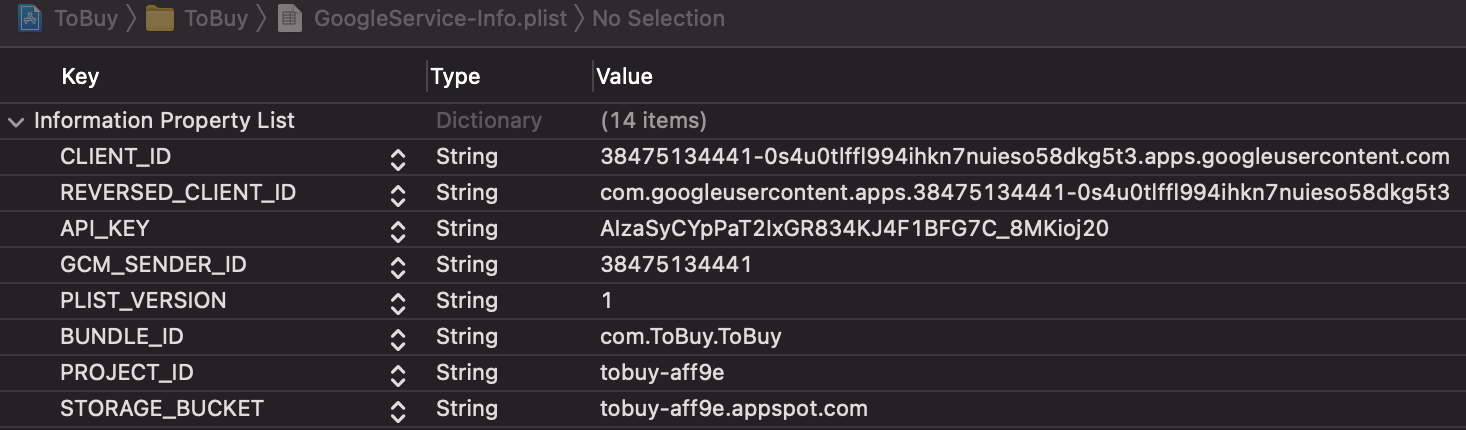
* La table user peut posséder une seule liste de course « posseder 0,1».
* La table shoppingList doit se faire posséder par un utilisateur « posseder 1,1 ».
* La table user peut créer un ou plusieurs magasin « créer 0,n ».
* La table shop doit se faire créer par un seul utilisateur « créer 1,1 ».



Modèle conceptuel de la base de donnée

### Requêtes pour la base de données

Comme dit plutôt, je n’utilise pas le format classique d’une base de données. En effet, les requetes ne se font pas en SQL. Je dois informé dans le fichier « **GoogleServices-inf.plist** », le nom de ma base de donnée, ici tobuy-aff9e. C’est le nom figurant sur l’application web Firebase. Il est alors maintenant possible d’utiliser des requetes via dans ce cas là, les « pods » **FirebaseAuth**, **FirebaseCore**, **FirebaseStorage**, **FirebaseFirestore**. Les explications de ces librairies sont dans le point « Analyse et conception » du rapport



Capture du contenu de la « googleService-info.plist »

## Stratégie de test

La partie des tests va se réaliser en 2 parties. La première partie de ses tests va être omniprésente lors de la partie de réalisation, il s’agira des tests efféctués dès lors qu’un changement ou une modification interviendra dans le code. La seconde partie, va être réalisé à la fin du projet, juste avant le rendu final du projet. La réalisation de ses tests va s’éffectuer grâce au tableau ci-dessous. Je mettrai dans la « description des tests effectués » l’état de chaque fonctionnalités ainsi qu’une description avancée de ceux-ci.

|  |  |
| --- | --- |
| C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\check.png | Symbole validant une fonctionnalité parfaitement fonctionnelle |
| C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\cross.png | Symbole invalidant une fonctionnalité non-fonctionnelle |
| C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\moyen.png | Symbole indiquant une fonctionnalité réalisée à moitié ou uniquement documentée |
|  |  |
|  | Symbole indiquant qu’aucuns tests n’a été éffectué |
|  |  |

|  |  |  |
| --- | --- | --- |
| Nom du test | Descriptif du test | Etat du test |
| Bouton page accueil | Tester si le bouton « invisible » de la page accueil permet de passer à la page « Ma liste de courses » | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png |
| Bouton ajout produit | Tester si le bouton « ajout de produit » de la page « Ma liste de courses » permet d’ajouter de passer à la page « Liste des produits » | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png |
| Bouton choix modèle magasin | Tester si le bouton « choix de modèle » de la page « Ma liste de courses » permet de passer à la page « Liste es magasins » | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png |
| Visuel ma liste de courses | Tester le visuel de ma liste de courses, voir si les produits ajoutés et supprimés sont correctement présents ou non | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png |
| Catégorie liste des produits | Tester la page « Liste de produits » pour vérifier que les produits sont bien classés | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png |
| Bouton incrémentation produit | Tester le bouton permettant d’ajouter un produit dans la liste des courses. Possibilité d’incrémenter ce nombre en re appuyant dessus | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png |
| Bouton décrémentation | Tester le bouton permettant d’enlever un produit dans la liste des courses. Si le nombre de produit devient 0 le bouton d’incrémentation devient un bouton de suppression | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png |
| Bouton magasin prédéfini | Tester le bouton permettant de choisir un modèle de magasin prédéfini. Le magasin prédéfini correspond au magasin séléctionné. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png |
| Bouton magasin personnalisé | Tester le bouton permettant de choisir un modèle de magasin personnalisé. Le magasin correspond au magasin séléctionné. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png |
| Bouton ajout magasin personnalisé | Tester si le bouton « ajout de magasin personnalisé » permet de passer à la page « création de magasin » | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png |
| Nom et lieu d’un magasin personnalisé | Permet grâce au boutou « validation » sur la page  « création magasin » de donner à un nom et un lieu au magasin. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png |
| Ordre de catégorie magasin personnalisé | Permet grâce au bouton « validation » sur la page « création magasin » de donner un ordre de catégorie grâce à des combobox | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png |
| Validation création | Tester si un produit n’apparait pas dans l’ordre du magasin si l’utilisateur est redirigé sur la page « traitement produits » sinon est redirigé dans la page « Ordre des courses » | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png |
| Traitement produit | Tester si grâce au bouton « ajouter » l’utilisateur peut traiter manuellement un produit non ordré | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png |
| Visuel ordre des courses | Tester le visuel de la page ordre des courses « trie » correctement les produits selon le magasin choisi | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png |
| Bouton de terminer | Tester le bouton « terminer » si l’utilisateur supprime bien la liste de course une fois le bouton pressé et est redirigé sur la page « Ma liste de course » | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png |

## Risques techniques

Les risques techniques sont étudiés pendant le pré-TPI. Dans ce projet, les risques pourrait être la nouveauté, nouveau frameworks ou nouvelle base de données. En effet, l’utilisation d’une de ces nouvelles « technologies » pour moi pourrait compromettre le projet. Cependant, les risques on été calculés et inclus dans la planification.

Les solutions qui ont été appliqueés en cas de perturbation durant le projet sont les suivantes :

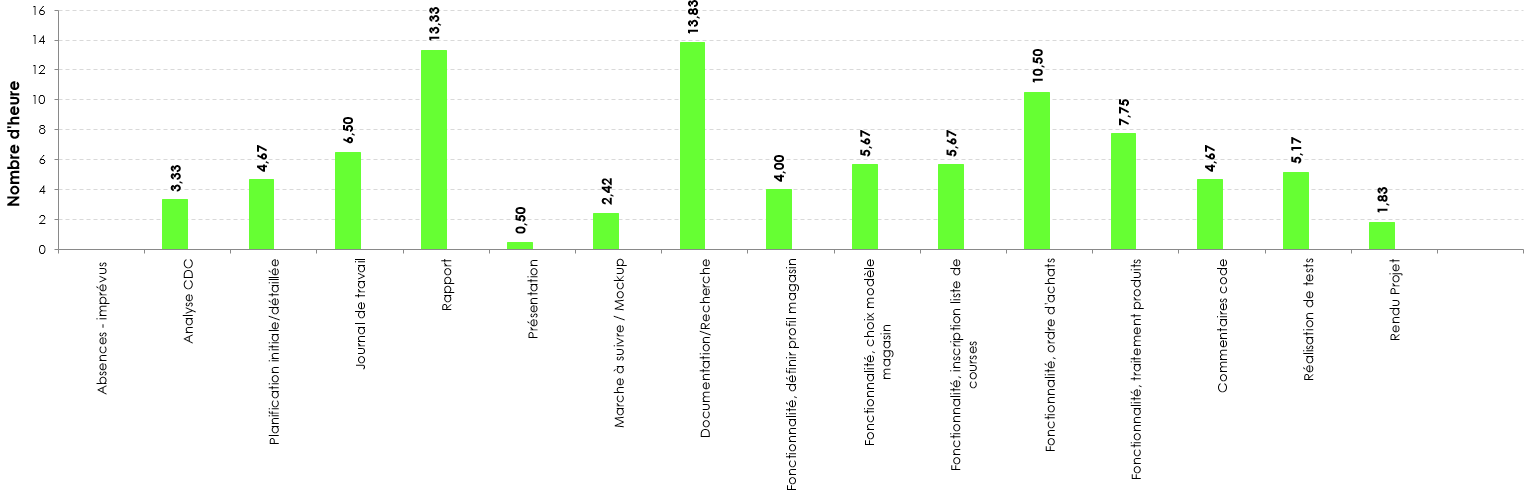
* Mise en ordre des priorités afin de garder en tête le fil rouge du projet
* Formations et documentation sur le sujet
* Demande d’aide à un particulier, mentionné dans le journal de travail

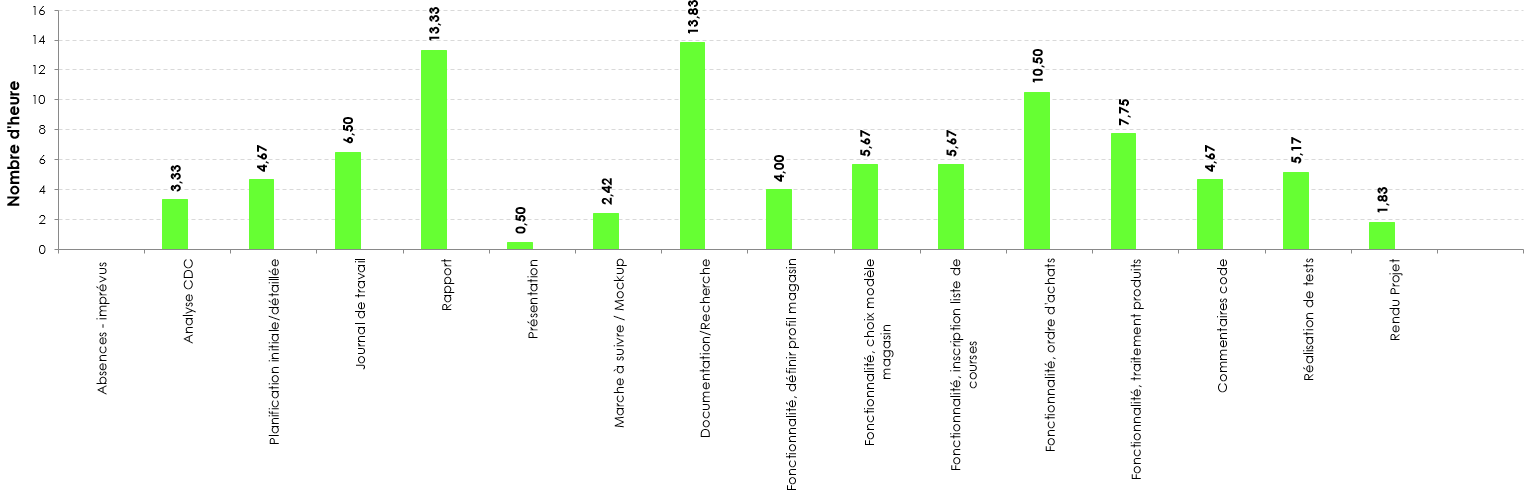
## Planification détaillée

Comme mentionné plutôt, le projet à été coupé en 4 parties. Cependant, j’ai découpé ces 4 parties en tâches bien distanctes afin d’organiser correctement ma planification détaillé. Il s’agit en principe la planification définitive du projet. Si les délais doivent être modifiés, le chef de projet doit être avisé, et les raisons doivent être expliquées dans l’historique.

Pour ce projet de 90 heures, (du mecredi 05 mai 2021 à13h10 au vendredi 04 juin 2021 à 14h) voici un plan du nombre d’heure de chaque taches .

= Représentation graphique des heures





## Dossier de conception

### Matériel

Un MAC de l’ETML m’a été prété le 15 mars jusqu’à la fin de mon TPI par M.Ollivier. il s’agit d’un MacBook Pro :

* Processeur : 2,2 GHz Intel Core i7 quatre cœurs
* Mémoire : 16 Go 1600 MHz DDR3
* Disque de démarrage : Macintosh HD
* Graphisme : Intel Iris Pro 1536 Mo

J’ai choisi de travailler sur ce matériel car ne possédant pas de mac ,c’était le seul mac mis à disposition par l’école.

### Système d’exploitation

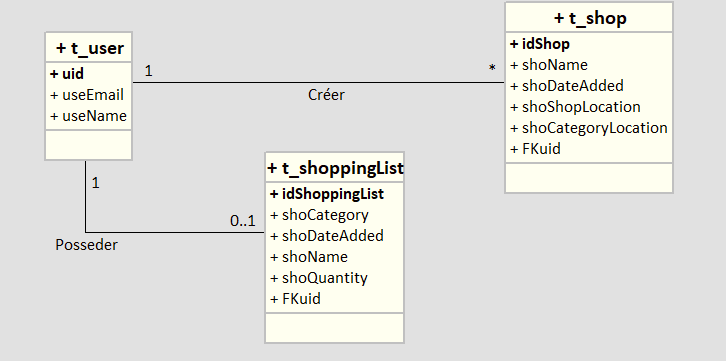
MacOS Big Sur, version 11.2.3

Version déjà installée sur le mac de prêt. Je n’ai donc pas choisi de version spécifique pour réaliser le projet.

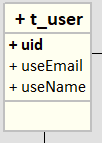
### Base de données

Ensemble de services Firebase proposé par Google. Services utilisés : FirebaseAuth et Firestore. NoSQL orienté documents.

Voici ci-dessous, le modèle relationnel et le contenu détaillé des tables :



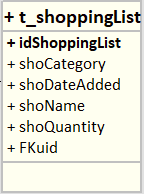
Modèle reltionnel de la base de données

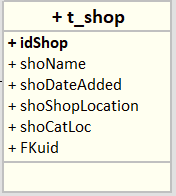


|  |  |  |  |
| --- | --- | --- | --- |
| **Nom** | **Type** | **Propriétés** | **Description** |
| uid | VARCHAR (50) | Primary key | Identifiant de l’utilisateur |
| useEmail | VARCHAR (50) | NOT NULL | Email de l’utilisateur |
| useName | VARCHAR(50) | NOT NULL | Nom de l’utilisateur |

|  |  |  |  |
| --- | --- | --- | --- |
| **Nom** | **Type** | **Propriétés** | **Description** |
| idShoppingList | VARCHAR (50) | Primary key | Identifiant de la liste |
| shoCategory | VARCHAR (50) | NOT NULL | Catégorie du produit dans la liste |
| shoDateAdded | VARCHAR(50) | NOT NULL | Date d’ajout du produit dans la liste |
| shoName | VARCHAR(50) | NOT NULL | Nom du produit dans la liste |
| shoQuantity | INT | NOT NULL | Quantité du produit dans la liste |
| FKuid | VARCHAR(50) | Foreign key | Identifiant de l’utilisateur possédant la liste |

|  |  |  |  |
| --- | --- | --- | --- |
| **Nom** | **Type** | **Propriétés** | **Description** |
| idShop | VARCHAR (50) | Primary key | Identifiant du magasin |
| shoName | VARCHAR (50) | NOT NULL | Nom du magasin |
| shoDateAdded | VARCHAR(50) | NOT NULL | Date d’ajout du magasin |
| shoShopLocation | VARCHAR(50) | NOT NULL | Lieu du magasin |
| shoCatLoc | VARCHAR(400) | NOT NULL | Ordre des catégories |
| FKuid | VARCHAR(50) | Foreign key | Identifiant de l’utilisateur créant le magasin |





Vu que j’ai décidé d’utilisé du noSQL orienté documents, j’utilise comme moyen de stockage des fichier « .json » et je gère les noms ou les locations des objects directement dans le code. Cependant j’ai choisi des « **VARCHAR** (50) » pour rester dynamic. Le « shoCatLoc » est fixé à un maximum de 400 caractères car c’est en fait une liste triant par ordre la location des produits.

Pour ce qui est des magasins, il y a 3 magasins prédéfini ainsi que les magasins créent par l’utilisateur. Les magasins prédéfni sont crée et lié à un utilisateur une fois le compte crée. C’est-à-dire que chaque magasin prédéfini est crée en doublon à chaque fois qu’un nouvel utilisateur intervient dans la base de données.

Les données des magasins prédéfnis sont implantées en dur dans le code, ce n’est pas la meilleure des méthodes, j’aurai pu les créer depuis un fichier mais je n’avais pas prévu cela dans ma planification. Cependant il existe un fichier excel avec les données récoltées durant le pré-tpi. [..\5. Autres\5.3 Modèle Magasin\M-TPI-LucaBohlen-ModeleMagasin.xlsx](../5.%20Autres/5.3%20Modèle%20Magasin/M-TPI-LucaBohlen-ModeleMagasin.xlsx)

### MVC

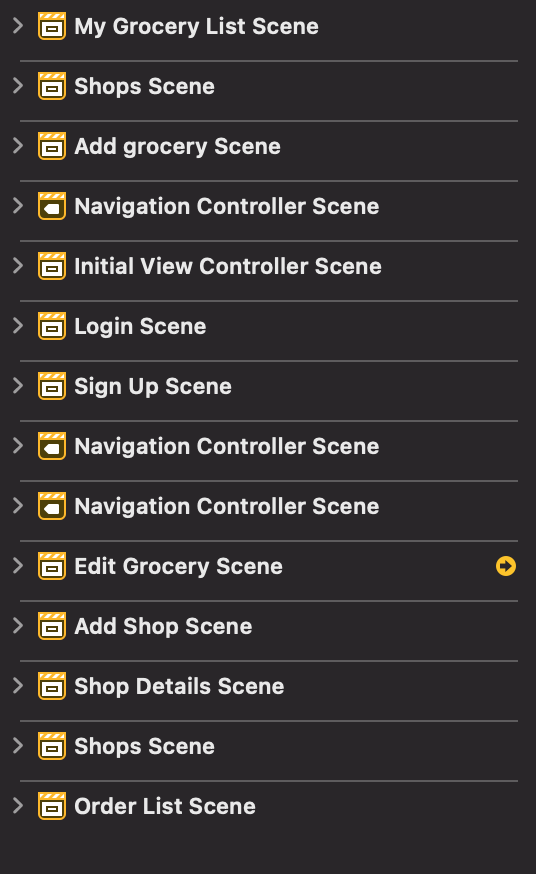
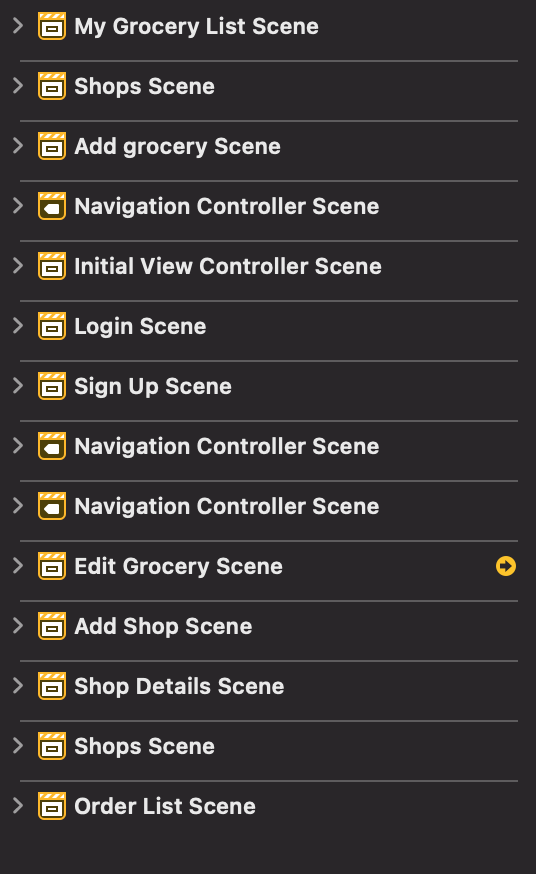
Pour ce projet, j’ai décidé d’implanté une architecture MVC. Elle est fréquemment utilisée pour les applications intégrant de nombreuses interfaces graphiques. Dans mon cas, j’ai beaucoup de pages/interfaces à utiliser.

De plus, l’utilisation de SwiftUI oblige en un sens l’utilisation d’une architecture MVC. En effet, lors de la création de page et d’ajouts d’objects, du code est crée automatiquement de sorte à ce que les parties de codes soit séparé en 3 parties : Modèle, Vue, Controller.

Dans ce projet, les **modèles** sont les « scripts » permettant de lier et de créer les documents de la base de données Firebase. Ci-dessous y sont inscripts tout les scripts « modèles » de mon code :

* User.swift *Utilisateur*
* ShoppingList.swift *Liste des courses*
* Shop.swift *Magasin*
* Networking.swift Class requetes base de donnée
* Public.swift Class variables globales

Pour les **vues**, c’est en faite le storyboard. Il y est inclus les objects de toutes les pages ainsi que les pages en elles-memes. Le storyboard permet également de visualiser les liens crées dans le code, par exemple un bouton diregant dans une autre page. Ci-dessous l’entiereté de mes pages, ici appellées scenes :



Ensemble des interfaces de l’application

Et pour finir les **controlleurs**, sont les scripts permettant de gérer les erreurs ainsi que les utilités de chaque object. Ils seront listés et decris dans le point réalisation du rapport.

### Cocoapods

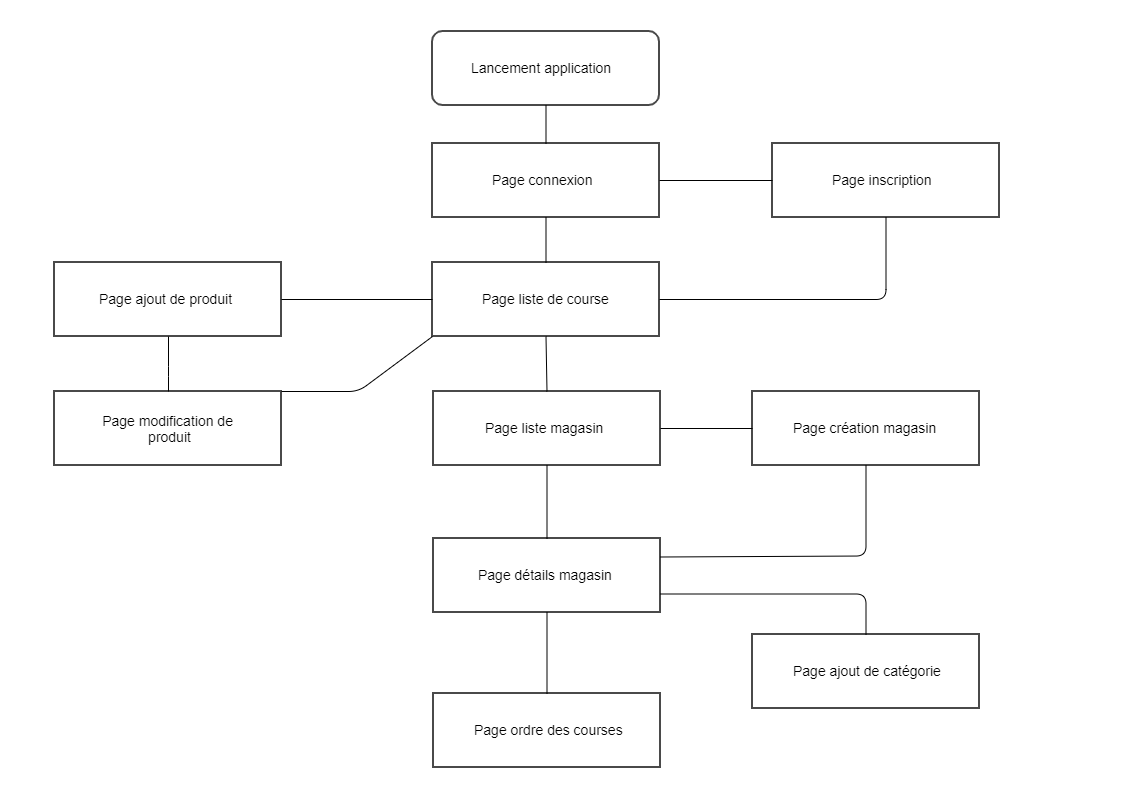
Cocoapods est un framework populaire pour les developpeurs d’application mobile iOS. J’ai choisi d’utilisé un framework dans ce projet pour, premièrement pouvoir communiquer avec ma base de données Firebase et deuxièmement pour faciliter la réalisation des parties plus graphiques du projet.

En fin de rapport dans le manuel d’installation, j’explique comment j’ai installé cocoapods. J’ai utilisé dans ce projet différent pods (librairie) :

* **Firebase/Core**[[1]](#footnote-1) : contient tous les kits de développement Firebase de la plate-forme Apple, à l’exception de FirebaseAnalytics et FirebaseML.
* **Firebase/Auth** [[2]](#footnote-2): : contient tous les kits de développement de l’authentification Firebase de la plate-forme Apple.
* **Firebase/Storage** [[3]](#footnote-3): contient tous les kits de développement du storage de Firebase de la plate-forme Apple.
* **Firebase/Firestore** [[4]](#footnote-4): contient tous les kits de développement du Firestore de Firebase de la plate-forme Apple.
* **IQKeyboardManagerSwift**[[5]](#footnote-5) : permet de contrer les problèmes liés au clavier par défault de xcode, je l’utilise pour l’ajout de produit et de magasin.
* **MBProgressHUD**[[6]](#footnote-6) : permet d’ajouter des fonctionnalités supplémentaire au HUD, je l’utilise pour l’ajout de catégorie dans la page création de magasin.
* **DropDown**[[7]](#footnote-7) : permet d’ajouter une liste déroulante design, je l’utilise plusieurs fois dans le code pour ajouter les catégories.
* **SDWebImage**[[8]](#footnote-8) : permet d’ajouter des cellules d’images, je l’utilise dans ma page liste de course.

### Organigramme

L’organigramme de programmation de ce projet est sous forme de séquence linéaire .



1. L’utilisateur doit se connecter, si il ne possède pas de compte il peut en crée un dans la page d’inscription.
2. Une fois connecté, depuis la page de la liste de course, il peut ajouter un produit dans la page d’ajout de produit. Depuis cette page il peut modifier un produit. Il peut revenir à la page liste des course.
3. Depuis la page liste de magasin, il peut se diriger dans la page de création de magasin. Une fois un magasin crée il peut afficher les détails du magasin dans la page détails.
4. Une fois le magasin séléctionné ou créer, l’utilisateur peut ajouter depuis la page ajout de catégorie une catégorie au magasin
5. Ensuite, depuis la page détails de magasin, l’utilisateur peut valider sa liste course et passer à la page ordre des courses afin de regarder le résultat final
6. Pour finir, depuis la page ordre des courses l’utilisateur peut terminer sa liste des courses. Sa liste est supprimée une fois le bouton terminer pressé.

## Outils et logiciels

### Xcode

C’est l’environnement de développement pour macOS le plus populaire pour développer des application iOS. Version 12.5 (12E262).

### GitKraken / Github

J’ai utilisé le client lourd gitkraken afin de simplifié mon rendu de travail journalier. Comme service d’hébergement j’ai utilisé github, et je l’ai choisi par habitude de travailler avec. Version gratuite.

### Swift

J’ai choisi le language de programmation SwiftUI car c’est le seul language permettant de faire une application mobile que j’ai étudié. Version de développement 5.4.

### Cocoapods

J’ai utilisé le framework pour utiliser ma base de données ainsi que faciliter la partie graphique de l’application. Je l’ai choisi car il est pratiquement indispensable pour certaines fonctionnalités.

### Office 365

La suite office m’a permis de documenté mon projet. L’utilisation de word pour le rapport et d’excel pour le journal de travail et la planification ainsi que de powerpoint pour la présentation. C’est la suite de bureautique que j’ai l’habitude d’utiliser. Version office professionnel plus 2016.

### Moqups

Grâce à l’application Web « Moqups » j’ai pu créer ma maquette ainsi que mon plan de liaison. J’avais déjà pu expérimenté cette application, c’est pourquoi j’ai décidé de l’utiliser dans ce projet. Version gratuite.

### Xcode simulator

Pour tester et simuler mon application, j’ai profité d’utiliser l’émulateur par défaut de xcode. Les simulations ont été éffectuées sur un « iPhone 11 » car c’est le modèle que j’ai physiquement pour tester mon application en conditions réelles. Version xCode 12.5.

### Looping

J’ai utilisé comme logiciel pour créer le modèle conceptuel et le modèle relationnel de ma base de donnée, looping. J’ai choisi de l’utilisé car j’avais pendant le pré-tpi utilisé ce logiciel. Version gratuite.

### Scriben

Afin de gardé une ortographe correcte, j’ai utilisé comme correcteur électronique d’orthographe Scriben. Je l’ai choisi car je l’ai utilisé dans divers projet auparavant.

### Cacoo

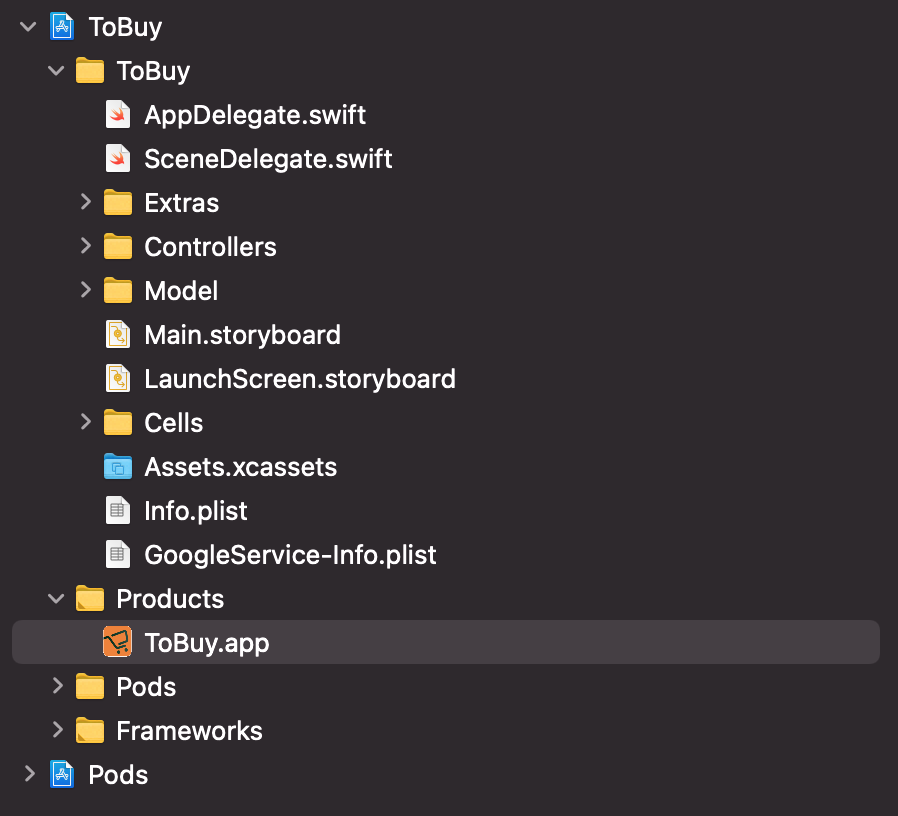
Pour réaliser mon organigramme de programmation, j’ai utilisé l’application web Cacoo. Je l’ai utilisé car elle est simple d’utilisation pour faire des organigramme à séquence linéaire.

# Réalisation

## Dossier de réalisation

### Répertoire logiciel

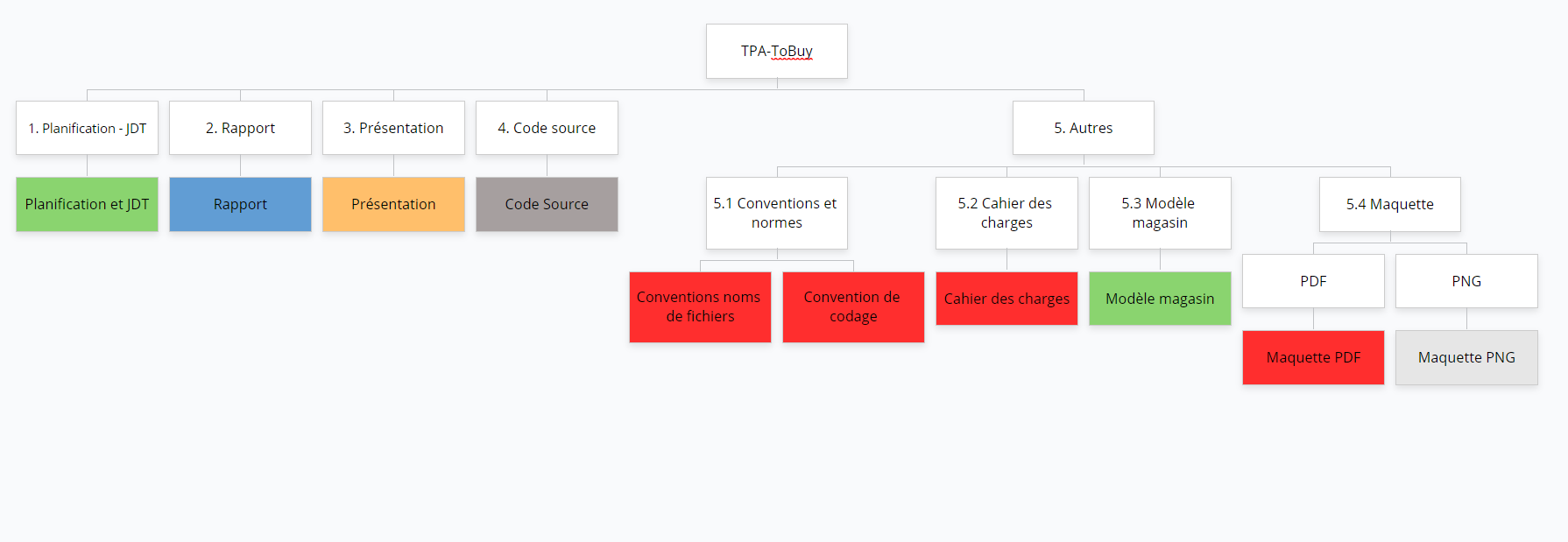
Le répertoire où est installé le logiciel est l’emplacement par défaut des application créée sur Xcode. C’est-à-dire dans le dossier « Products ». Il est nécaissaire de build l’application avant de lancer l’émulateur :



### Fichiers de documentation

|  |  |
| --- | --- |
| **Nom du fichier** | **Description** |
| T-TPI-lucabohlen-PlanificationJDT.xlsm | La planification initiale et détaillée, le journal de travail et le tableau des tâches. |
| R-TPI-LucaBohlen-Rapport.docx | Rapport du projet. L’analyse et la conception, la réalisation, les tests et les bilans du projet |
| P-TPI-LucaBohlen-Presentation.pptx | Support de la présentation du projet |
| D-ICH-PAT01-ConventionNomsFichiers.pdf | Conventions de nomage des fichiers de l’ETML |
| I-ConventionsDeCodageV3.5.0.pdf | Conventions de codage de l’ETML |
| H-TPI-LucaBohlen-CDC.pdf | Cahier des charges du projet |
| M-TPI-LucaBohlen-ModeleMagasin.xlsx | Modèle des magasins étudiés |
| MM-TPI-LucaBohlen-Maquette.pdf | Maquette du projet |

### Arborescence fichiers de documentation

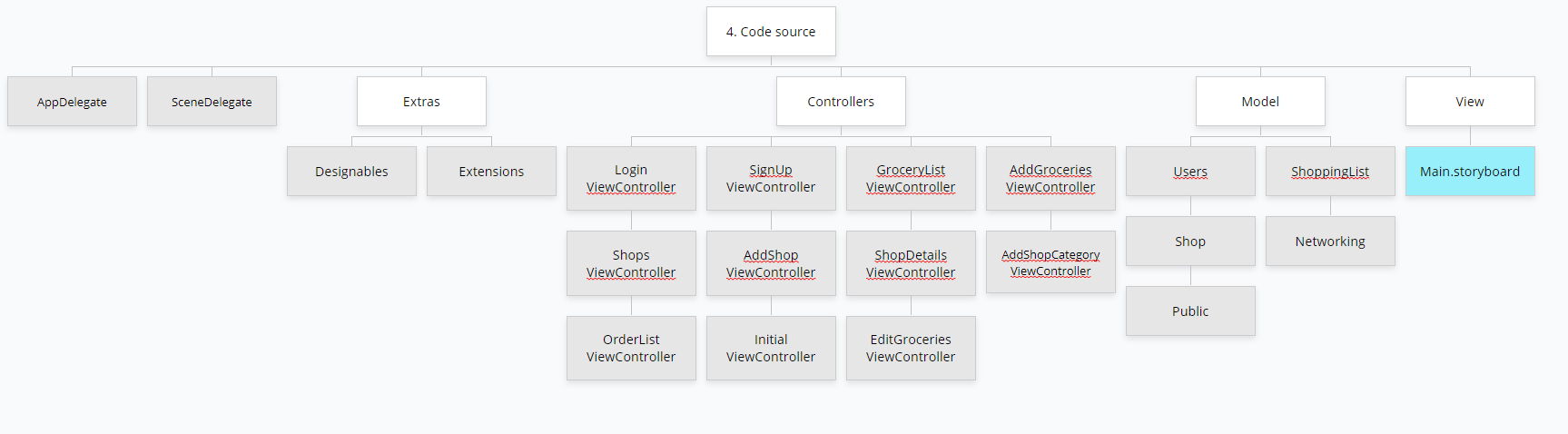


|  |  |
| --- | --- |
|  | = Dossier |
|  | = Fichier excel |
|  | = Fichier word |
|  | = Fichier powerpoint |
|  | = Fichier code source |
|  | = Fichier PDF |
|  | = Fichier PNG |

### Fichiers code source

|  |  |
| --- | --- |
| **Nom du fichier** | **Description** |
| AppDelegate.swift | Script responsable des liaisons avec la base de donnée firebase |
| SceneDelegate.swift | Script responsable de l’affichage de l’écran |
| Designables.swift | Script généré automatiquement qui gère le design de chaque objet crée |
| Extensions.swift | Script généré automatiquement qui gère les box des messages |
| LoginViewController.swift | Script crée semi-automatiquement qui gère la connexion de l’utilisateur |
| SignUpViewController.swift | Script crée semi-automatiquement qui gère l’inscription de l’utilisateur |
| InitialViewController.swift | Script responsable de la première page active, connecté ou non |
| GroceryListViewController.swift | Script responsable de la page de liste des courses et de la déconnexion |
| AddGroceriesViewController.swift | Script responsable de la page d’ajout de produit, gère les erreurs |
| EditGroceriesViewController.swift | Script responsable de la page de modification de produit, gère les erreurs |
| ShopsViewController.swift | Script responsable de la page de la liste des magasins, supprimer, ajouter, modifier |
| AddShopViewController.swift | Script responsable de l’ajout d’un nouveau magasin, gère les erreurs |
| ShopDetailsViewController.swift | Script responsable de la page des détails magasin, modifier catégorie, supprimer |
| AddShopCategoryViewController.swift | Script responsable de la page d’ajout de catégorie, gère les erreurs |
| OrderListViewController.swift | Script responsable de la page final de l’application, affichage résultat et fin de l’application |
| Users.swift | Script responsable des requetes utilisateurs pour la base de données |
| ShoppingList.swift | Script responsable des requetes listes de courses pour la base de données |
| Shop.swift | Script responsable des requetes magasins pour la base de données |
| Networking.swift | Script responsable des appels à la base de données, créer, récupérer |
| Public.swift | Script contenant les variables globales |
| Main.storyboard | Interface permettant de gérer de manière graphique les pages et les objets |

### Arborescence fichiers code source



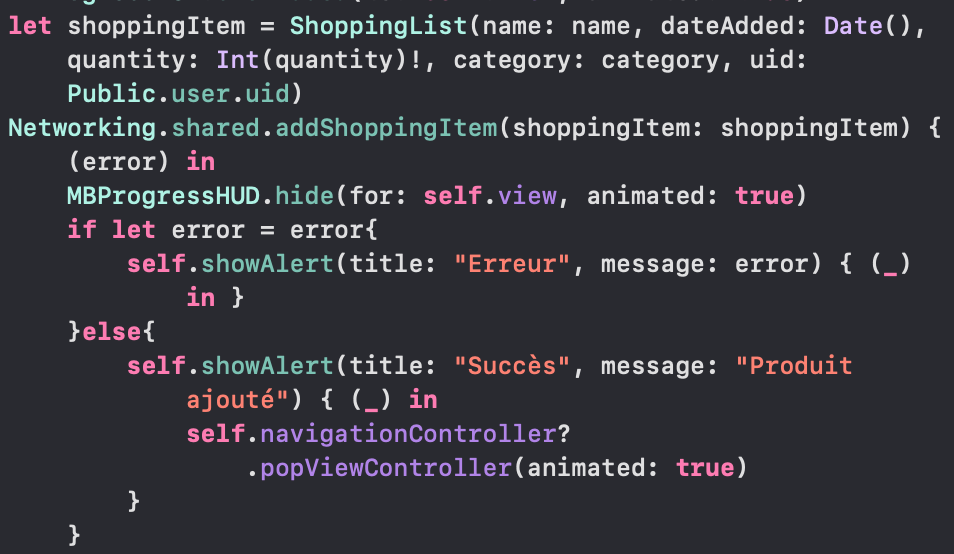
|  |  |
| --- | --- |
|  | = Dossier |
|  | = Fichier swift |
|  | = Interface/View |
|  |  |
|  |  |
|  |  |
|  |  |

### Version de produit

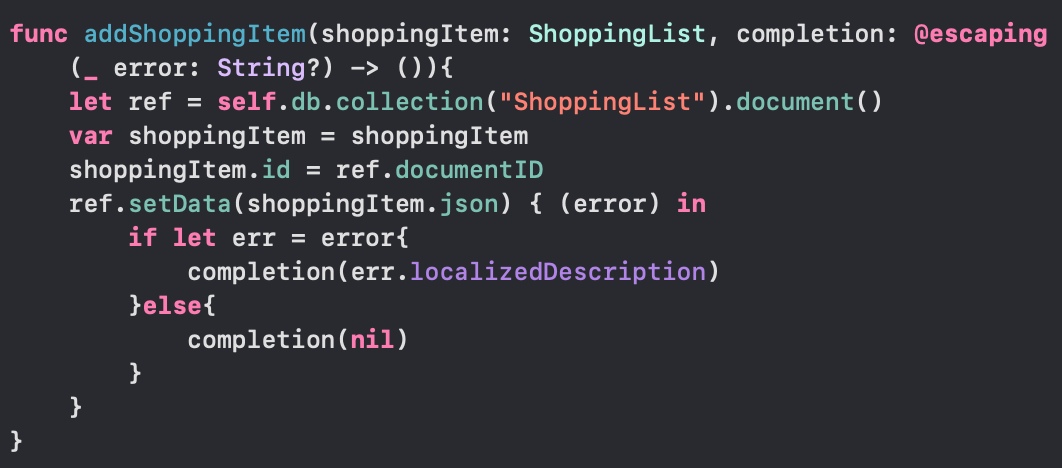
J’ai pendant ce projet fait un commit chaque jours sur le dépôt git[[9]](#footnote-9). J’ai donc environ une quarantaine de commits. Pour ce qui est de la version du produit j’estime avoir 4 versions du projet :

* La première du 10 mai 2021 (commit : Code), version du squelette commentaire demandé par le chef de projet
* La deuxième du 21 mai 2021 (commit : Code database), version où l’implantation de la base de donnée à été terminé
* La troisième du 26 mai 2021 (commit : code source + commentaire), version où les fonctionnalités principales ont été implantées et fonctionnel (ajout produit, modification produit, liste magasin, ajout magasin, ordre des produits)
* La quatrième du 03.06.2021 (commit : code final), version finale du projet avec toutes les fonctionnalités et les détails règlés

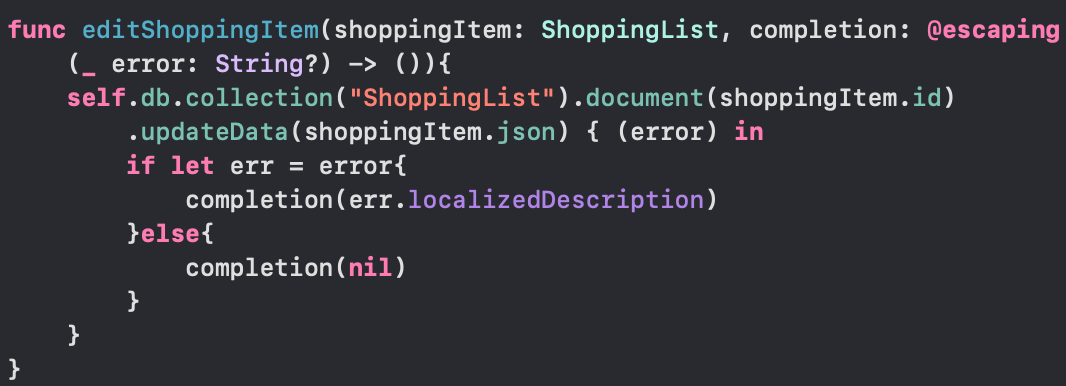
### Ajouts et modification produit

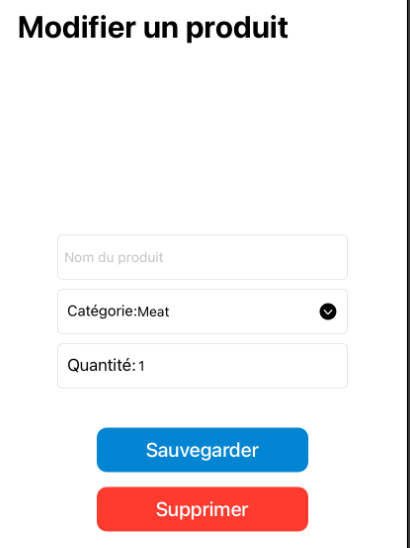
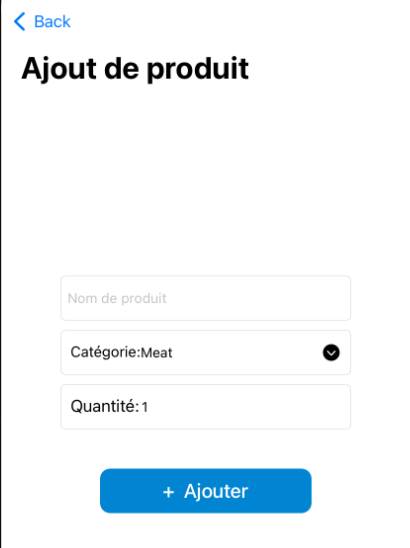
Le système d’ajout et de modification de produit sont pratiquement semblable. En effet, pour ajouter un produit, l’application récupère via les textboxs, la date actuel et la liste des catégories pour les implanter directement dans la base de données. En indiquant l’uid de l’utilisateur.

Networking.shared.addShoppingItem : permet d’ajouter dans le fichier .json de la base de données. J’appelle dans le script « networking » la function ci-dessous « addShoppingItem » :



Depuis cette function ci-dessus, les données du produit sont ajoutées dans un fichier .json crée de la base de données.

Ce principe montré ici est à peut près le même que pour la modification de produit, la différence étant que j’appelle la function « editShoppingItem » et que les données sont remplacé dans un fichier .json déjà existant



### Networking

La méthode utilisé pour l’ajout et la modification de produit est la même que pour l’ajout et la modification de magasin. On récupère les données depuis la vue pour appeler une function depuis le controller pour enfin envoyer les données dans la base de données depuis le model « networking ».

Dans le cahier des charges il est mentionné que je devais ajouter au moins 3 magasin prédéfini, je les ai donc ajouté en dur dans le code, c’est une modification que j’aurai pu faire. Ajouter un .json par magasin et ensuite de mon utilisateur appeler un magasin prédéfini. Actuellement, lors de la création d’un utilisateur, les 3 magasins prédéfini sont recréer en doublon avec l’uid de l’utilisateur créer.

Voici ci-dessous la function « createUser » qui dans un **premier** temps créer le .json de l’utilisateur. Dans un **second** temps instancie les 3 magasins avec les valeurs en dur dans le code, rajoute le uid de l’utilisateur venant d’être crée. Et dans un **troisième** temps créee les .json des magasins avec les données prédéfinies et le uid de l’utilisateur :



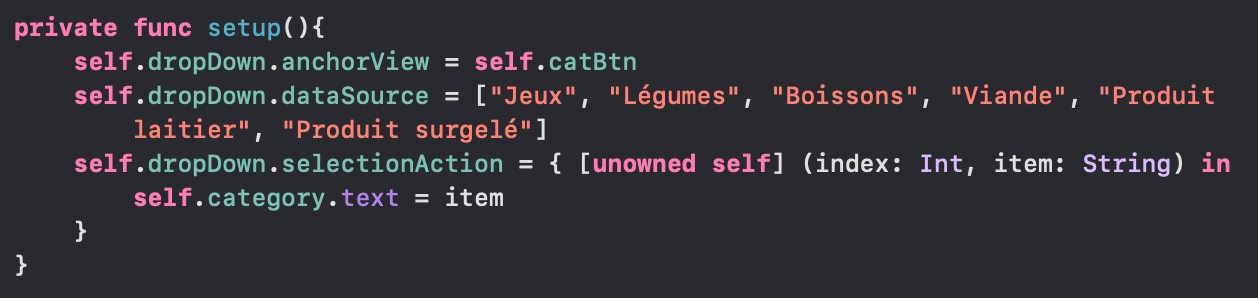
**Deuxième**

**Premier**

**Troisième**

### DropDown

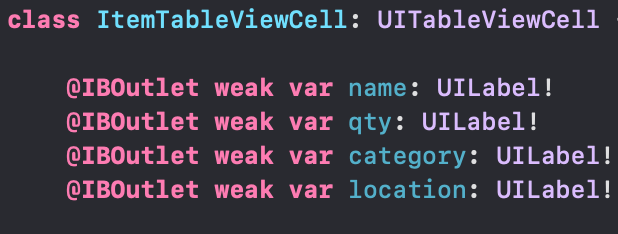
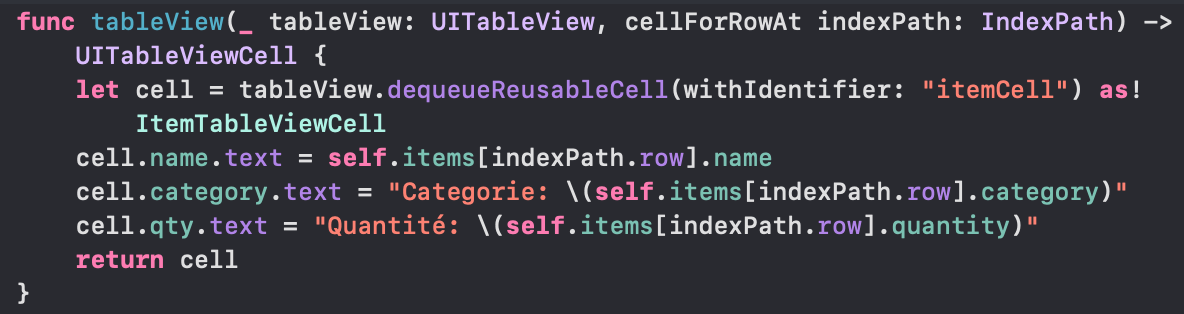
J’utilise pour mes listes la librairies DropDown, j’utilise une liste pour l’ajout de produit, la modification de produit et pour la modification de l’ordre des catégories dans les magasins. Le code est composé de la datasource (données dans la liste) qui est entré en dur. La mention de l’index en int et des éléments en string :



Les éléments on été choisi au préalable durant le pré-tpi pendant l’analyse des différents magasins prédéfini.

### Cellules

Les cellules fournissent la représentation visuelle des lignes d’un tableau. Dans ce projet, j’utilise des prototypes de cellules afin d’afficher dans la liste des courses les produits mais aussi dans la page d’ordre des courses.



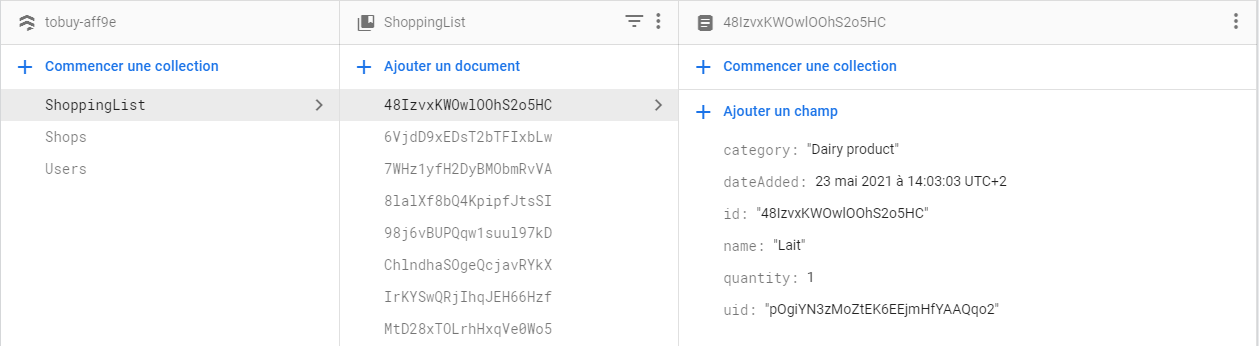
|  |  |
| --- | --- |
| ItemTableViewCell | |
| Représentation visuelle | Contenu de cellule list |

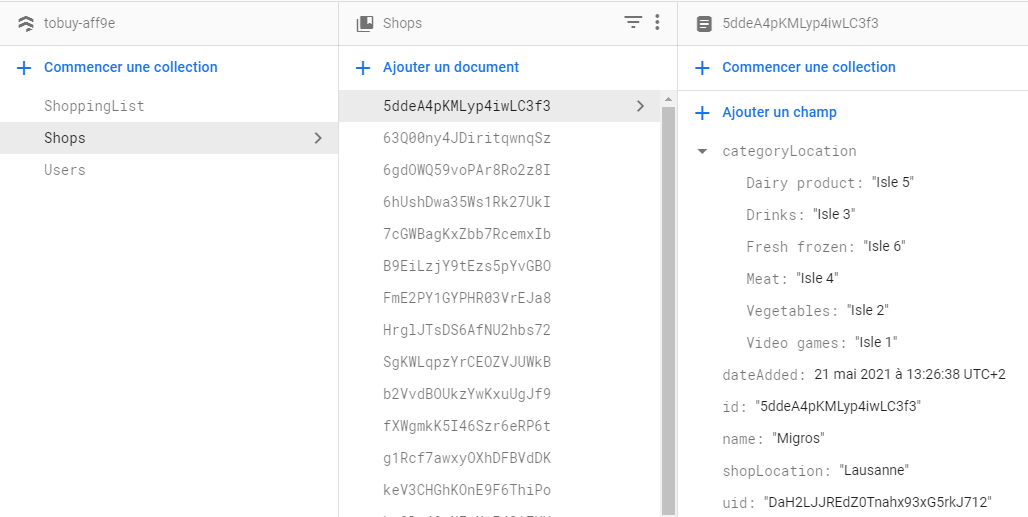
|  |
| --- |
| Utilisation de cette cellule list |

Crée une variable cell en indiquant d’utiliser la cellule (ItemTableViewCell). Ensuite indique le nom la category et la quantité de la cellule. Et pour finir retourne la cellule.

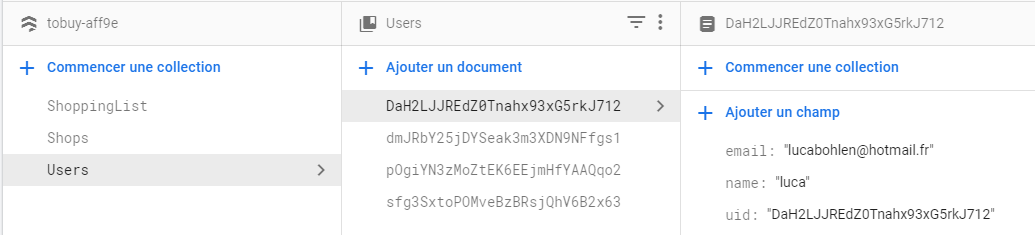
### Dictionnaire des données

Mes données sont sous forme de fichier .json. Ils sont listés selon le modèle noSQL orienté document c’est-à-dire : **collection** > **document** > **data**.

Sur cette capture, on peut voir la **collection** « ShoppingList » qui est la collection possédant la liste de course de chaque utilisateur :

Sur cette capture, on peut voir la **collection** « Shops » qui est la collection possédant les magasins et les magasin prédéfini de chaque utilisateur:

Sur cette capture, on peut voir la **collection** « Users » qui est la collection possédant les utilisateurs avec leur uid :



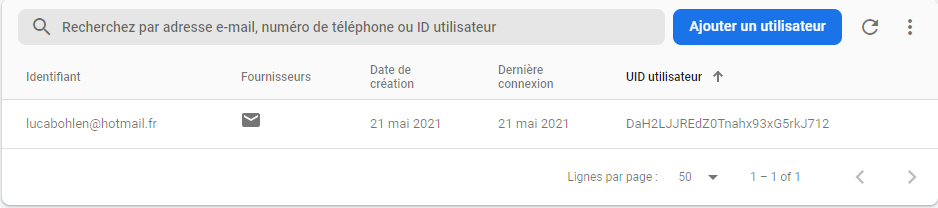
### Changement base de données

Au début du projet, je pensais utiliser la base de données proposé par apple developer « CoreData ». Cependant, j’ai changé pour partir sur firebase comme dit plutôt dans le rapport.

J’ai effectué ce changement car firebase est plus adapté à mon projet. En effet, l’utilisation les données de CoreData est plus compliqué. Il faut pour synchroniser les données sur plusieurs appareils utiliser le CloudKit.

Core data est un framework utilisé pour modéiser des objets dans l’application. C’est le « front end » du stockage des données, où le « back en » doit être en SQL ou des plists. Il s’agit plus d’un concept d’utilisateur unique qui stocke des données localement.

J’ai aussi choisi firebase car elle permet d’ajouter une fonctionnalité de multi-compte. Avec des différents utilisateurs, j’ai plus de facilité à créer des listes uniques.

Killermy Boukhlifa m’a proposé l’idée de firebase avant que j’utilise une base de données, c’est aussi grâce à ce timing que je n’ai pas eu de changement à faire. J’ai quand même eu une période de documentation qui m’a décalé ma planification.

### Modification maquette

Au cours du projet, j’ai du modifier, améliorer ou changer ma maquette de base afin de garder une simplicité d’utilisation.

Premièrement, j’ai ajouté une page de connexion ainsi qu’une page d’inscription. Elles sont apparues avec la base de données :

## Description des tests effectués

*Pour chaque partie testée de votre projet, il faut décrire:*

* *les conditions exactes de chaque test*
* *les preuves de test (papier ou fichier)*
* *tests sans preuve: fournir au moins une description*

|  |  |  |
| --- | --- | --- |
| Description | Conséquence | Résultat |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Liste des documents fournis

*Lister les documents fournis au client avec votre produit, en indiquant les* numéros de versions

* *le rapport de projet*
* *le manuel d'Installation (en annexe)*
* *le manuel d'Utilisation avec des exemples graphiques (en annexe)*
* *autres…*

# Conclusions

*Développez en tous cas les points suivants:*

* *Objectifs atteints / non-atteints*
* *Points positifs / négatifs*
* *Difficultés particulières*
* *Suites possibles pour le projet (évolutions & améliorations)*

## Bilan des fonctionnalités demandées

Concernant les fonctionnalités demandées, toutes les fonctionnalités présentes dans le cahier des charges sont disponibles dans l’application, on y a même rajouté la possibilité de créer un compte pour l’utilisateur car de base nous avions prévu de stocker les données en dur sur le téléphone.

Certaines fonctionnalités sont disponibles sur l’application mais elles sont encore améliorables et ne sont pas forcément présentes comme nous l’aurions souhaité au départ dû à notre manque d’expérience en ce qui concerne la programmation sur mobile (IOS) mais elles répondent tout de même au cahier des charges.

Nous envisageons d’améliorer par la suite notre application maintenant que nous aurons plus de contraintes de temps afin d’ajouter des fonctionnalités supplémentaires qui corresponderaiennt plus à nos attentes.

## Bilan de la planification

## Bilan personnel

# Annexes

## Résumé du rapport du TPI / version succincte de la documentation

## Webographie

## Planification Initiale

Dossier en annexe du rapport.

## Journal de travail

Dossier en annexe du rapport.

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Durée** | **Activité** | **Remarques** |
|  |  |  |  |
|  |  |  |  |

## Manuel d’installation

## Manuel d’Utilisation

## Archives du projet

1. <https://cocoapods.org/pods/FirebaseCore> [↑](#footnote-ref-1)
2. <https://cocoapods.org/pods/FirebaseAuth> [↑](#footnote-ref-2)
3. <https://cocoapods.org/pods/FirebaseStorage> [↑](#footnote-ref-3)
4. <https://cocoapods.org/pods/FirebaseFirestore> [↑](#footnote-ref-4)
5. <https://cocoapods.org/pods/IQKeyboardManagerSwift> [↑](#footnote-ref-5)
6. <https://cocoapods.org/pods/MBProgressHUD> [↑](#footnote-ref-6)
7. <https://cocoapods.org/pods/DropDown> [↑](#footnote-ref-7)
8. <https://cocoapods.org/pods/SDWebImage> [↑](#footnote-ref-8)
9. <https://github.com/LucaBohlen/TPA-ToBuy> [↑](#footnote-ref-9)