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 Glossaire 4](#_Toc73704677)

[2 Analyse préliminaire 5](#_Toc73704678)

[2.1 Introduction 5](#_Toc73704679)

[2.1.1 Titre 5](#_Toc73704680)

[2.1.2 Description 5](#_Toc73704681)

[2.1.3 Contexte 5](#_Toc73704682)

[2.2 Objectifs 6](#_Toc73704683)

[2.3 Planification Initiale 7](#_Toc73704684)

[3 Analyse / conception 8](#_Toc73704685)

[3.1 Concept 8](#_Toc73704686)

[3.1.1 Méthodologie de projet 8](#_Toc73704687)

[3.1.2 Interface graphique / Plan de liaison 9](#_Toc73704688)

[3.1.3 Base de données Firebase 14](#_Toc73704689)

[3.1.4 Modèle conceptuel 15](#_Toc73704690)

[3.1.5 Requêtes pour la base de données 15](#_Toc73704691)

[Comme dit plutôt, je n’utilise pas le format classique d’une base de données. En effet, les requêtes ne se font pas en SQL. Je dois informer dans le fichier « GoogleServices-inf.plist », le nom de ma base de données, ici tobuy-aff9e. C’est le nom figurant sur l’application web Firebase. Il est alors maintenant possible d’utiliser des requêtes 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. 15](#_Toc73704692)

[3.2 Stratégie de test 16](#_Toc73704693)

[3.3 Risques techniques 17](#_Toc73704694)

[3.4 Planification détaillée 18](#_Toc73704695)

[3.5 Dossier de conception 19](#_Toc73704696)

[3.5.1 Matériel 19](#_Toc73704697)

[3.5.2 Système d’exploitation 19](#_Toc73704698)

[3.5.3 Base de données 19](#_Toc73704699)

[3.5.4 MVC 21](#_Toc73704700)

[3.5.5 Cocoapods 22](#_Toc73704701)

[3.5.6 Organigramme 23](#_Toc73704702)

[3.6 Outils et logiciels 24](#_Toc73704703)

[3.6.1 Xcode 24](#_Toc73704704)

[3.6.2 GitKraken / Github 24](#_Toc73704705)

[3.6.3 Swift 24](#_Toc73704706)

[3.6.4 Cocoapods 24](#_Toc73704707)

[3.6.5 Office 365 24](#_Toc73704708)

[3.6.6 Moqups 24](#_Toc73704709)

[3.6.7 Xcode simulator 24](#_Toc73704710)

[3.6.8 Looping 24](#_Toc73704711)

[3.6.9 Scriben 24](#_Toc73704712)

[3.6.10 Cacoo 24](#_Toc73704713)

[4 Réalisation 25](#_Toc73704714)

[4.1 Dossier de réalisation 25](#_Toc73704715)

[4.1.1 Répertoire logiciel 25](#_Toc73704716)

[4.1.2 Fichiers de documentation 25](#_Toc73704717)

[4.1.3 Arborescence fichiers de documentation 26](#_Toc73704718)

[4.1.4 Fichiers code source 27](#_Toc73704719)

[4.1.5 Arborescence fichiers code source 28](#_Toc73704720)

[4.1.6 Version de produit 29](#_Toc73704721)

[4.1.7 Ajouts et modification produit 29](#_Toc73704722)

[4.1.8 Networking 31](#_Toc73704723)

[4.1.9 DropDown 32](#_Toc73704724)

[4.1.10 Cellules 32](#_Toc73704725)

[4.1.11 Dictionnaire des données 33](#_Toc73704726)

[4.1.12 Changement base de données 34](#_Toc73704727)

[4.1.13 Modification maquette 35](#_Toc73704728)

[4.2 Description des tests effectués 37](#_Toc73704729)

[4.3 Liste des documents fournis 39](#_Toc73704730)

[5 Conclusions 39](#_Toc73704731)

[5.1 Améliorations 39](#_Toc73704732)

[5.2 Bilan des fonctionnalités demandées 40](#_Toc73704733)

[5.3 Bilan de la planification 40](#_Toc73704734)

[5.4 Bilan personnel 40](#_Toc73704735)

[5.5 Bilan graphique comparaison 41](#_Toc73704736)

[6 Annexes 42](#_Toc73704737)

[6.1 Webographie 42](#_Toc73704738)

[6.2 Manuel d’utilisation 43](#_Toc73704739)

[6.3 Archives du projet 43](#_Toc73704740)

[6.4 Résumé rapport 44](#_Toc73704741)

[6.5 Planification Initiale 45](#_Toc73704742)

[6.6 Journal de travail 54](#_Toc73704743)

[6.7 Bilan graphique planification 67](#_Toc73704744)

# Glossaire

|  |  |
| --- | --- |
| **F** | |
| framework | Un framework est un ensemble d’outils et de composants logiciels (Cocoapods). |
| **J** | |
| Json | Format de données textuelles dérivé de la notation du langage JavaScript. |
| **M** | |
| Mock-up | Fait référence à une maquette d’interface utilisateur. |
| **N** | |
| NoSQL | NoSQL désigne une famille de systèmes de gestion de bases de données (SGBD) qui s'écarte du paradigme classique des bases relationnelles. |
| **P** | |
| Pods | Un pods est un outil spécifique du framework cocoapods. |
| **S** | |
| Storyboard | L’ensemble des vues de l’application, permettant d’ajouter ou modifier des objets sur les pages. |
| Swift | Langage de programmation pour le développement d’application iOS |
| SwiftUI | Framework de swift permettant de constuire des interfaces. |
| **V** | |
| Varchar | Type de données de longueur variable avec un maximum n. |

# Analyse préliminaire

## Introduction

Ce projet consiste à réaliser une application mobile IOS dans le langage 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 visités par l’élève 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 seront présentés selon l'organisation d'un modèle de magasin. La même liste apparaîtra 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’élève doit réaliser un travail personnel sur la base d’un cahier des charges reçu le 1er jour. 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’élève 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 application 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'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 panification initiale découpée en 4 parties visibles sur le graphique en secteur :

• La réalisation regroupe les tâches de mise en œuvre des fonctionnalités demandées.

• La documentation regroupe les tâches liées au rapport et au journal de travail.

• L’analyse regroupe les tâches liées à la lecture du CDC ainsi qu’à la documentation et recherche.

• Les tests regroupent 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 suivante. 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 cycle 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 présentation orale)

**Validation**

**Mise en oeuvre**

**Conception**

**Analyse**

**Exigences**

**Mise en service**

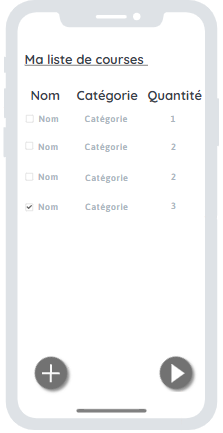
### Interface graphique / Plan de liaison

À l’aide de l’application web de création de maquette [moqups](https://moqups.com/), j’ai crée des interfaces graphiques de chaque pages présentes 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 chacune des pages :

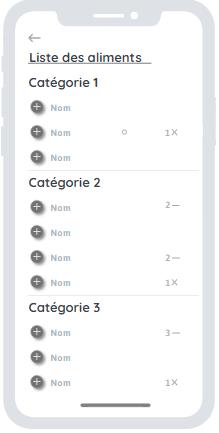
1. **Page titre** : permet "d'accueillir" l'utilisateur en affichant le logo et le nom d'application.



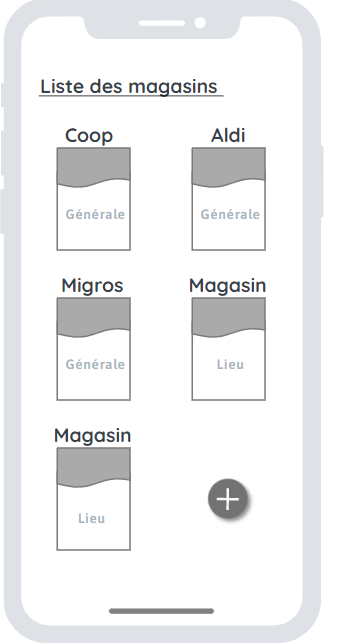
1. **Page de la liste des courses** : permet avec un bouton d'afficher la page "Ajout de produit" et avec un autre bouton de passer directement à la page "Liste des magasins".



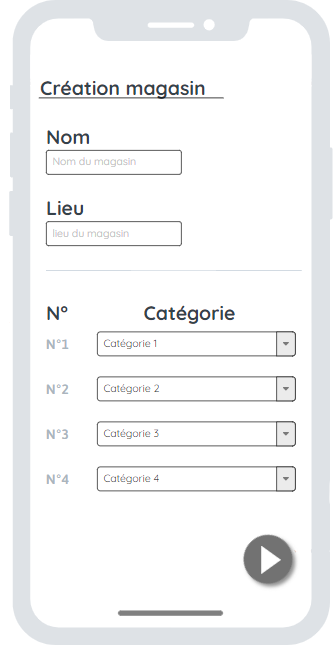
1. **Page d'ajout de produits :** permet d'ajouter ou de retirer des produits dans notre liste. Ils sont triés par catégorie, il n'y a pas de barre de recherche de prévus ainsi que la possibilité d'ajout de produit personnalisé (amélioration possible).



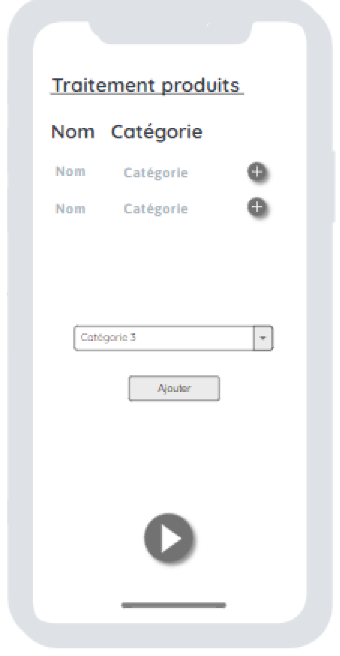
1. **Page de choix du magasin** : permet de visualiser l'ensemble des magasins. Il y a un bouton d'ajout de modèle de magasin. Si l'utilisateur clique sur un magasin, il est redirigé sur la page "Ordre des courses".



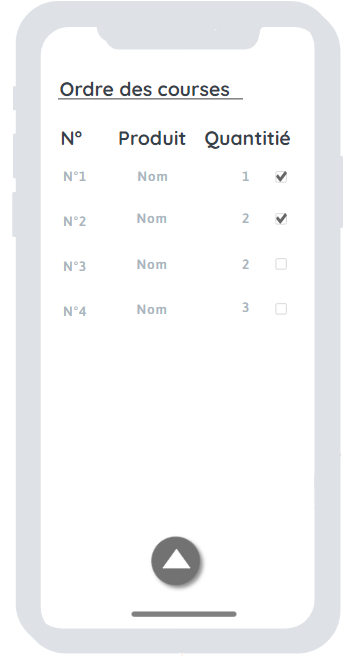
1. **Page création magasin** : permet la création d'un nouveau modèle de magasin. Il faut indiquer le nom du magasin ainsi que le lieu. Ensuite via des comboboxs, on doit désigner l'ordre des catégories de produit. Un bouton en bas de page permet d'envoyer le modèle.

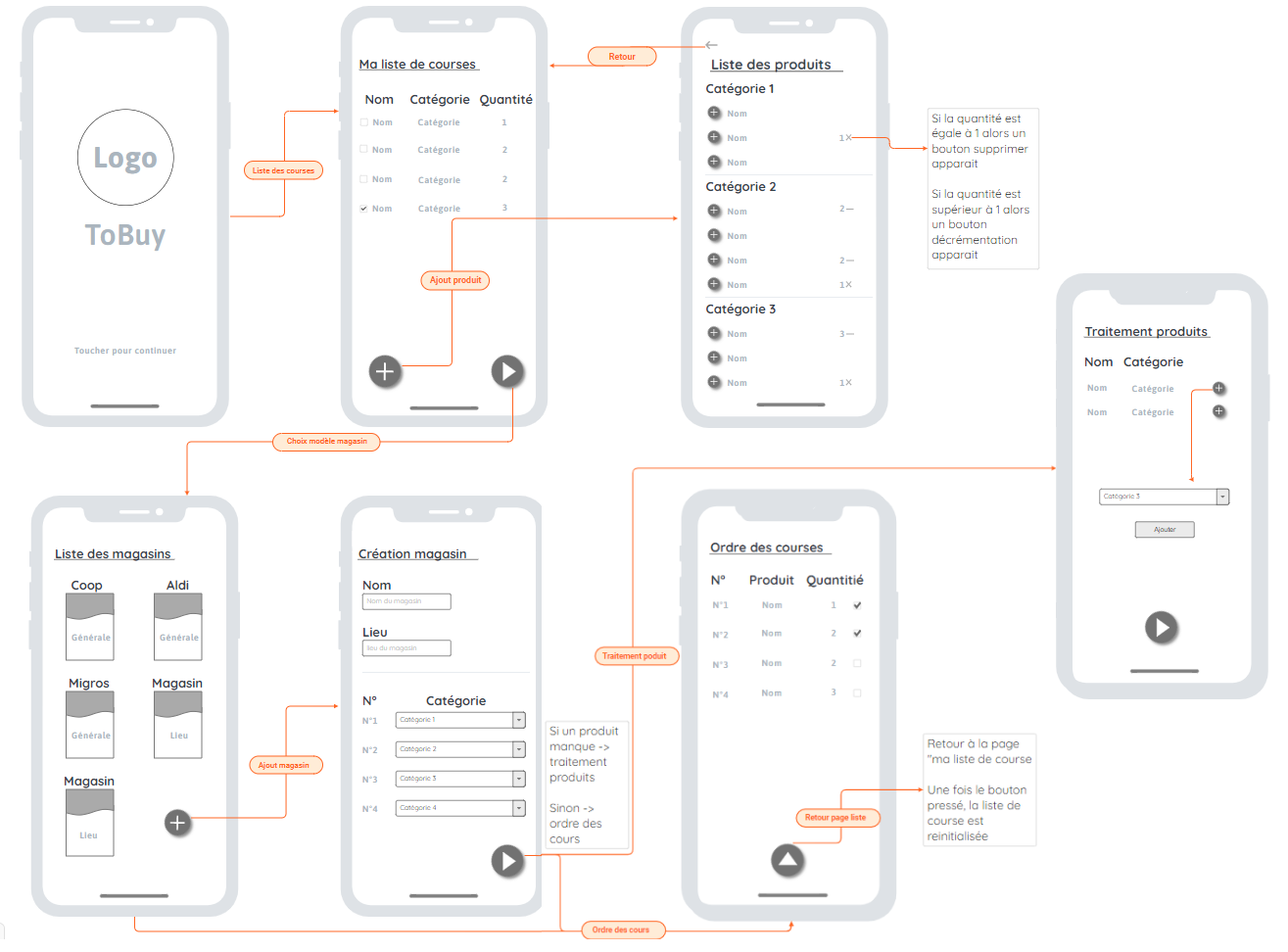


1. **Page de traitement produits** : permet à l'utilisateur de décider comment traiter les produits de la liste de courses qui ne figurent pas dans le modèle du magasin choisi. Via des comboboxs et un bouton de validation. Un bouton en bas de page permet d'envoyer les produits dans le modèle.



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".



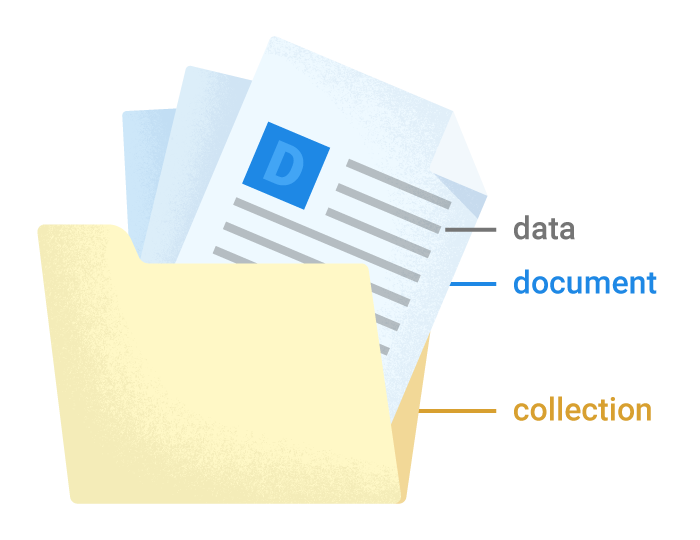


### Base de données Firebase

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 adaptée à mon projet, c’est-à-dire une base de données d’utilisation simple et des requêtes facilement assimilables. J’ai utilisé les services, FirebaseAuth et le Firestore.

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

* La donné brute (**DATA**) : représente la donnée brute que l’on veut sauvegarder. Comme par exemple un entier int, une chaîne de caractères 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, etc.
* 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 ».

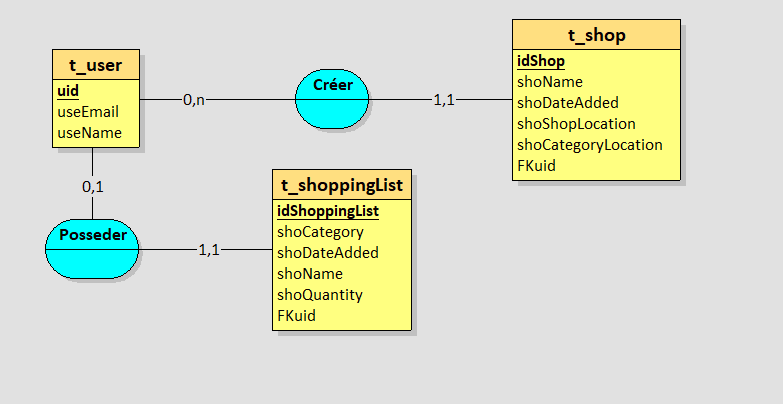


R

### Modèle conceptuel

Avec le logiciel, looping, j’ai crée 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 magasins « créer 0,n ».
* La table shop doit se faire créer par un seul utilisateur « créer 1,1 ».



### Requêtes pour la base de données

### C:\Users\lucbohlen\Downloads\Capture d’écran 2021-05-31 à 08.18.25.pngComme dit plutôt, je n’utilise pas le format classique d’une base de données. En effet, les requêtes ne se font pas en SQL. Je dois informer dans le fichier « GoogleServices-inf.plist », le nom de ma base de données, ici tobuy-aff9e. C’est le nom figurant sur l’application web Firebase. Il est alors maintenant possible d’utiliser des requêtes 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.

## 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 effectués dès lors qu’un changement ou une modification interviendra dans le code. La seconde partie, va être réalisée à la fin du projet, juste avant le rendu final du projet. La réalisation de ses tests va s’effectuer grâce au tableau ci-dessous. Je mettrai dans la « description des tests effectués » l’état de chaque fonctionnalité ainsi qu’une description avancée de ceux-ci.

|  |  |
| --- | --- |
| Nom du test | Descriptif du test |
| Bouton page accueil | Tester si le bouton « invisible » de la page accueil permet de passer à la page « Ma liste de courses » |
| 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. » |
| 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 les magasins. » |
| 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 |
| Catégorie liste des produits | Tester la page « Liste de produits » pour vérifier que les produits sont bien classés |
| 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 |
| Bouton décrémentation | Tester le bouton permettant d’enlever un produit dans la liste des courses. Si le nombre de produits devient 0, le bouton d’incrémentation devient un bouton de suppression |
| Bouton magasin prédéfini | Tester le bouton permettant de choisir un modèle de magasin prédéfini. Le magasin prédéfini correspond au magasin sélectionné. |

|  |  |
| --- | --- |
| Bouton magasin personnalisé | Tester le bouton permettant de choisir un modèle de magasin personnalisé. Le magasin correspond au magasin sélectionné. |
| Bouton ajout magasin personnalisé | Tester si le bouton « ajout de magasin personnalisé » permet de passer à la page « création de magasin » |
| Nom et lieu d’un magasin personnalisé | Permet grâce au bouton « validation » sur la page « création magasin » de donner à un nom et un lieu au magasin. |
| 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 |
| Validation création | Tester si un produit n’apparaît pas dans l’ordre du magasin si l’utilisateur est redirigé sur la page « traitement produit » sinon est redirigé dans la page « Ordre des courses » |
| Traitement produit | Tester si grâce au bouton « ajouter » l’utilisateur peut traiter manuellement un produit non en ordre |
| Visuel ordre des courses | Tester le visuel de la page ordre des courses « trie » correctement les produits selon le magasin choisi |
| 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 » |

## Risques techniques

Les risques techniques sont étudiés pendant le pré-TPI. Dans ce projet, les risques pourraient être la nouveauté, nouveaux frameworks ou nouvelle base de données.

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

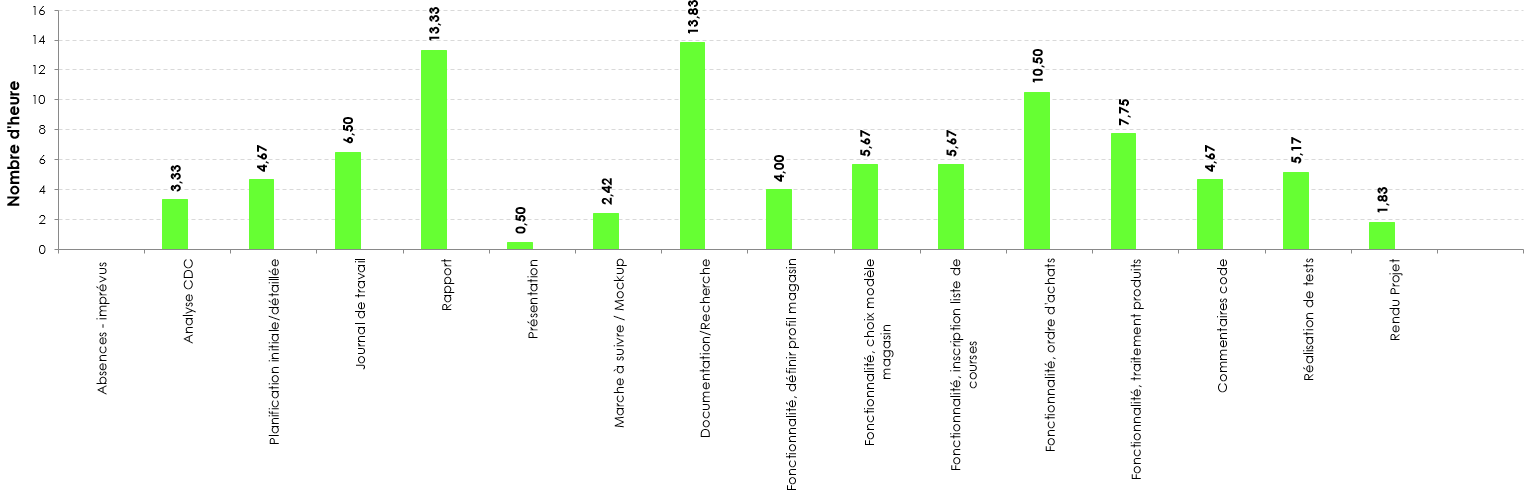
* Mise en ordre des priorités afin de garder en tête le fil rouge du projet
* Formations et documentation sur le sujet
* Demande d’aide à un particulier, mentionné dans le journal de travail

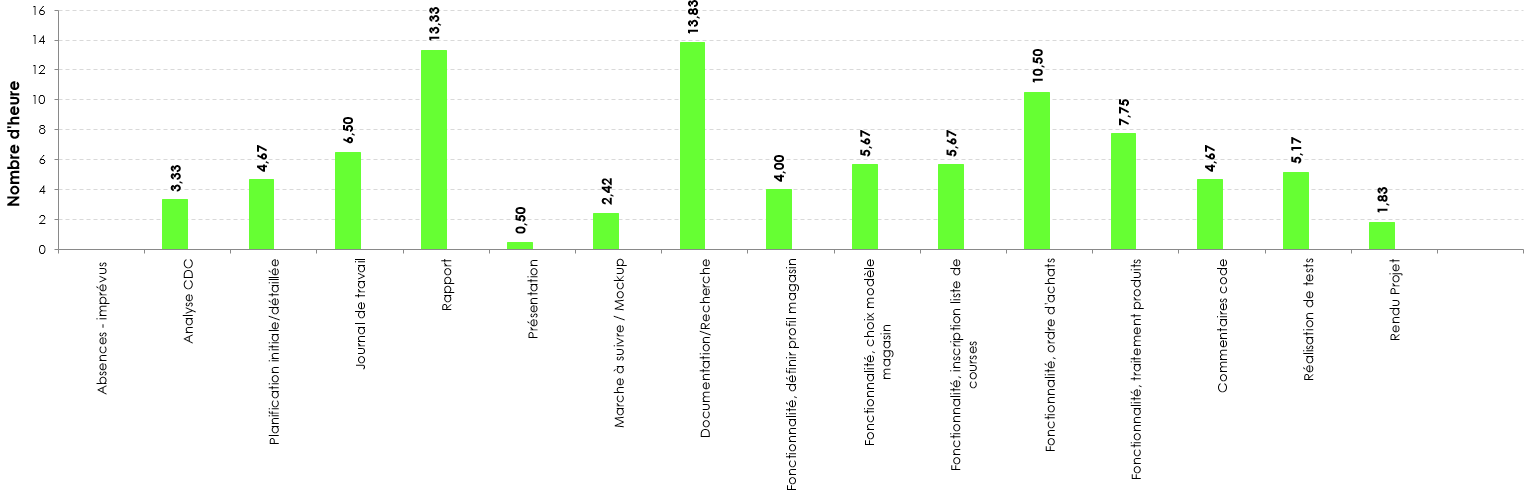
## Planification détaillée

Comme mentionné plutôt, le projet a été coupé en 4 parties. Cependant, j’ai découpé ces 4 parties en tâches bien distinctes afin d’organiser correctement ma planification détaillée. 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 mercredi 05 mai 2021 à 13 h 10 au vendredi 04 juin 2021 à 14 h) voici un plan du nombre d’heures de chaque tache.

= Représentation graphique des heures





## Dossier de conception

### Matériel

Un MAC de l’ETML m’a été prêté 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

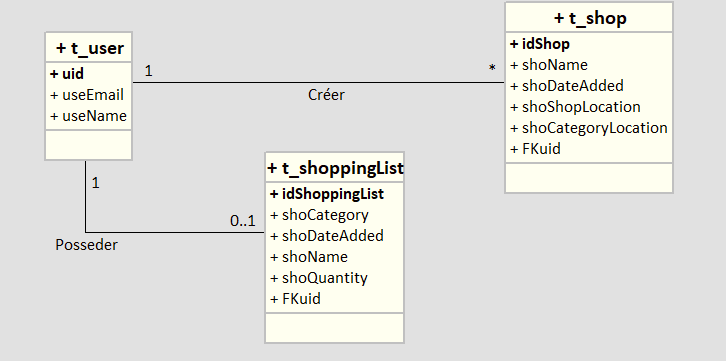
MacOS Big Sur, version 11.2.3

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

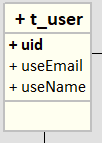
### Base de données

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 :



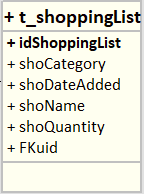
Modèle reltionnel de la base de données

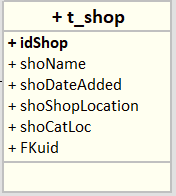


|  |  |  |  |
| --- | --- | --- | --- |
| **Nom** | **Type** | **Propriétés** | **Description** |
| uid | VARCHAR (50) | Primary key | Identifiant de l’utilisateur |
| useEmail | VARCHAR (50) | NOT NULL | Email de l’utilisateur |
| useName | VARCHAR(50) | NOT NULL | Nom de l’utilisateur |

|  |  |  |  |
| --- | --- | --- | --- |
| **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’utiliser du noSQL orienté documents, j’utilise comme moyen de stockage des fichiers « .json » et je gère les noms ou les locations des objets 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 prédéfinis sont créés et liés 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 prédéfinis 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

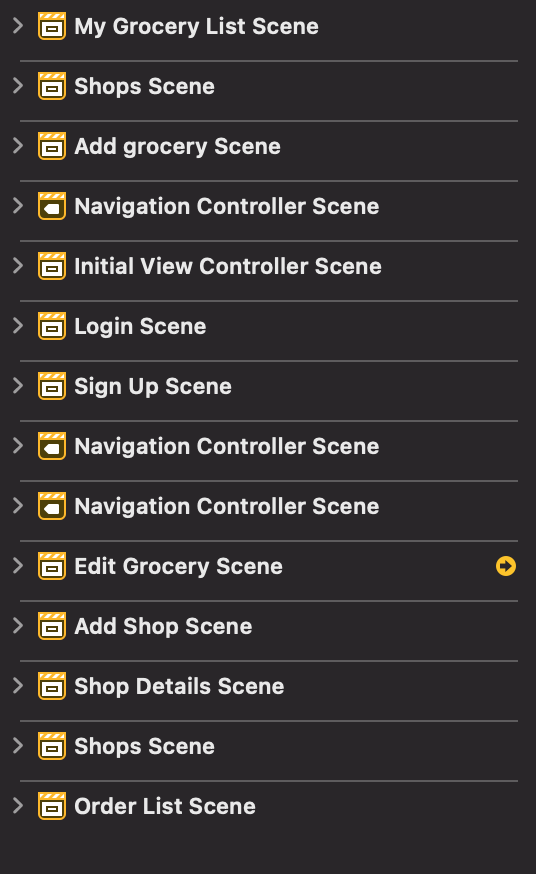
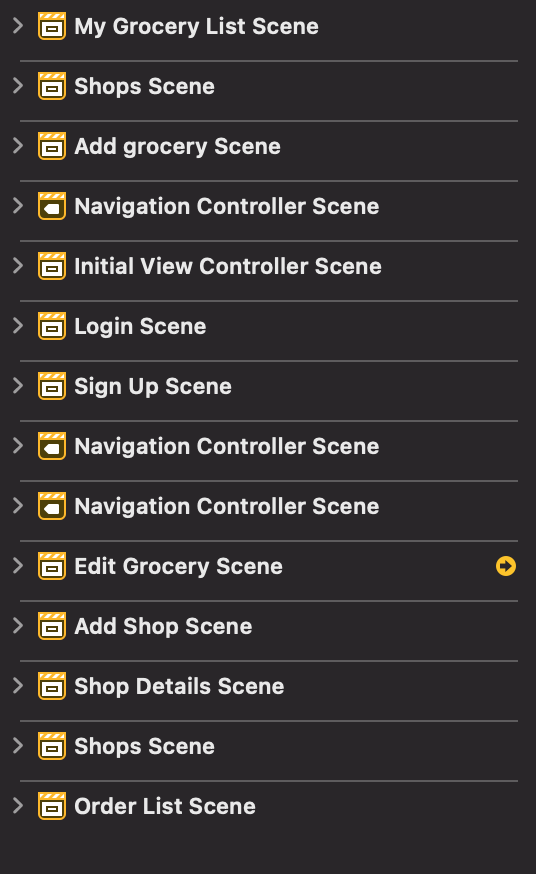
Pour ce projet, j’ai décidé d’implanter une architecture MVC. Elle est fréquemment utilisée pour les applications intégrant de nombreuses interfaces graphiques. Dans mon cas, j’ai beaucoup de pages/interfaces à utiliser.

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’objets, du code est crée automatiquement de sorte à ce que les parties de codes soient 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 inscrits tout les scripts « modèles » de mon code :

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

Pour les **vues**, c’est en fait le story-board. Il y est inclus les objets de toutes les pages ainsi que les pages en elles-mêmes. Le storyboard permet également de visualiser les liens crées dans le code, par exemple un bouton dirigant dans une autre page. Ci-dessous l’entièreté de mes pages, ici appellées scenes :



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

### Cocoapods

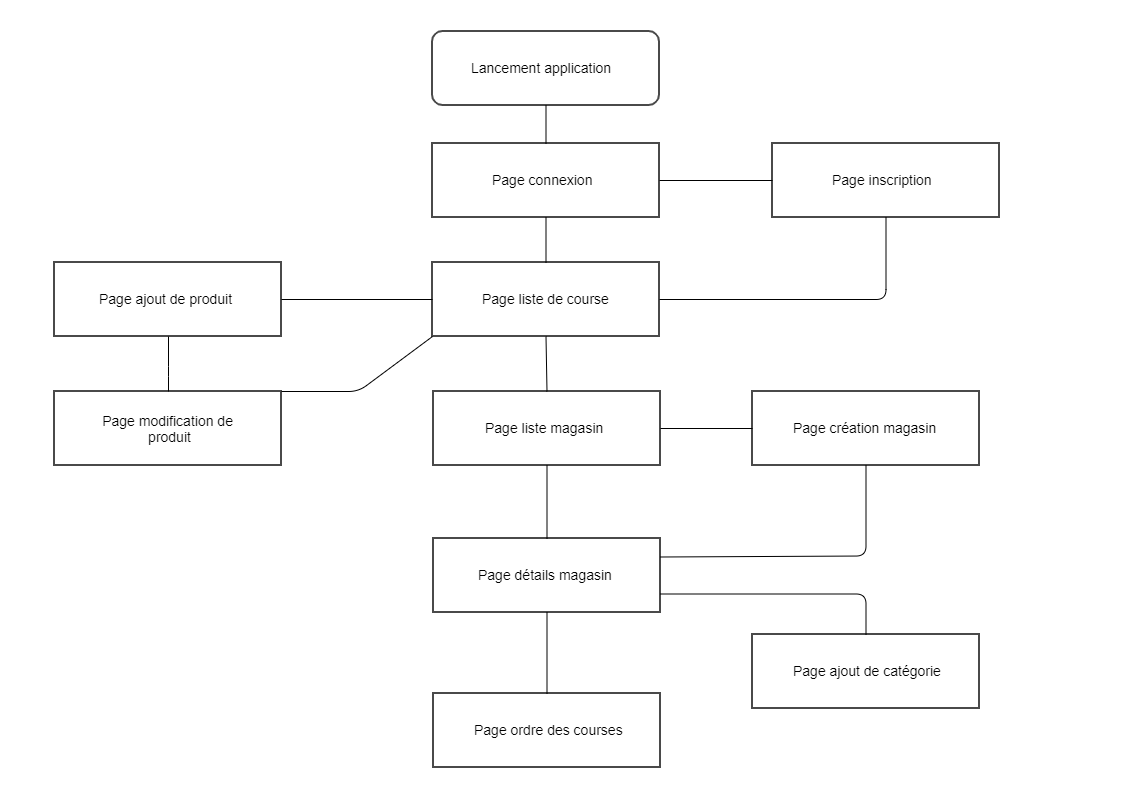
Cocoapods est un framework populaire pour les développeurs d’application mobile iOS. J’ai choisi d’utiliser un framework dans ce projet pour, premièrement pouvoir communiquer avec ma base de données Firebase et deuxièmement pour faciliter la réalisation des parties plus graphiques du projet.

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) : contiens tous les kits de développement Firebase de la plate-forme Apple, à l’exception de FirebaseAnalytics et FirebaseML.
* **Firebase/Auth** [[2]](#footnote-2): : contiens tous les kits de développement de l’authentification Firebase de la plate-forme Apple.
* **Firebase/Storage** [[3]](#footnote-3): contiens tous les kits de développement du storage de Firebase de la plate-forme Apple.
* **Firebase/Firestore** [[4]](#footnote-4): contiens 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 défaut de xcode, je l’utilise pour l’ajout de produit et de magasin.
* **MBProgressHUD**[[6]](#footnote-6) : permet d’ajouter des fonctionnalités supplémentaires 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, s’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 courses.
3. Depuis la page liste de magasins, 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 sélectionné 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 terminé 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 simplifier 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 langage de programmation SwiftUI, car c’est le seul langage 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 documenter 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 expérimenter 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 effectuées 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’utiliser car j’avais pendant le pré-tpi utilisé ce logiciel. Version gratuite.

### Scriben

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

### Cacoo

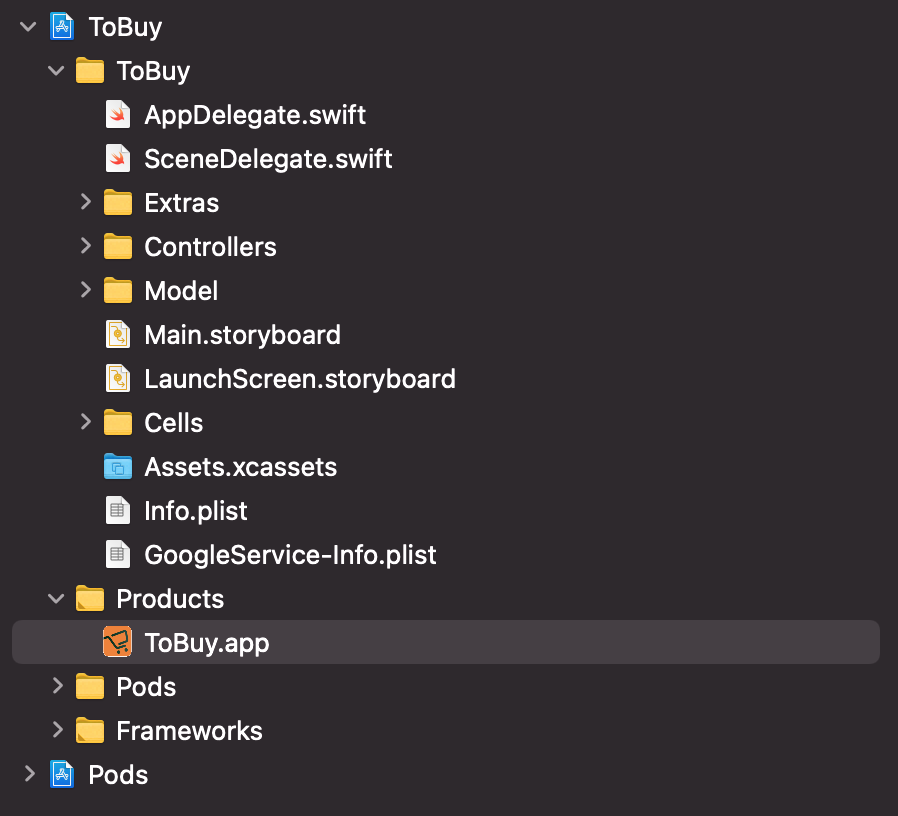
Pour réaliser mon organigramme de programmation, j’ai utilisé l’application web Cacoo. Je l’ai utilisé, car elle est simple d’utilisation pour faire des organigrammes à séquence linéaire.

# Réalisation

## Dossier de réalisation

### Répertoire logiciel

Le répertoire où est installé le logiciel est l’emplacement par défaut des applications créées sur Xcode. C’est-à-dire dans le dossier « Products ». Il est nécessaire 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 nommage 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 |
| MM-TPI-LucaBohlen-Demonstration.mp4 | Demonstration vidéo de l’application |

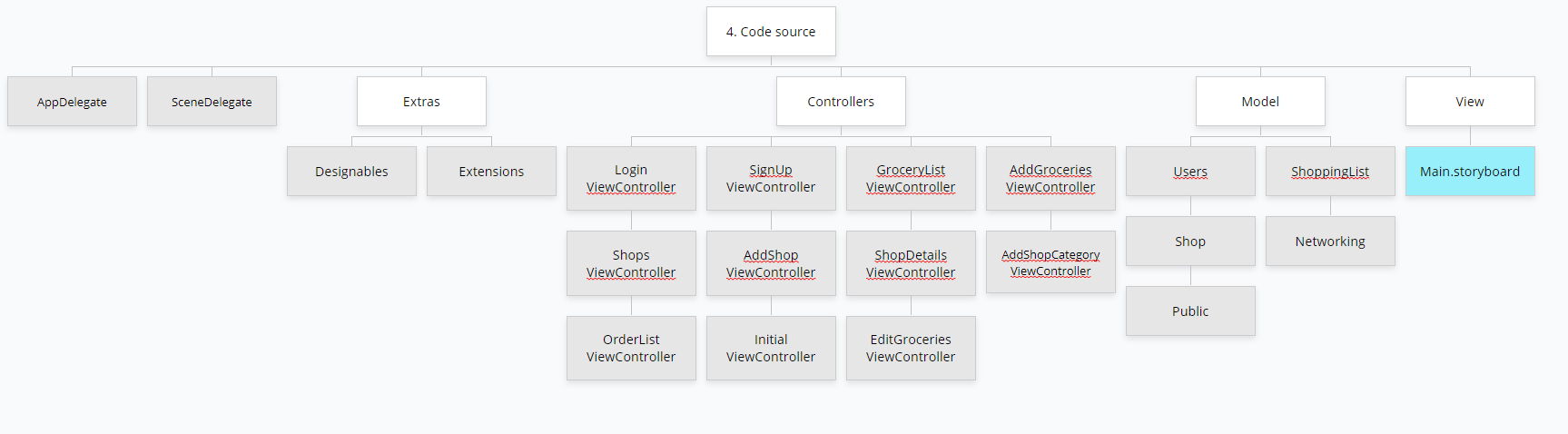
### Arborescence fichiers de documentation

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

### Fichiers code source

|  |  |
| --- | --- |
| **Nom du fichier** | **Description** |
| AppDelegate.swift | Script responsable des liaisons avec la base de données 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 requêtes utilisateurs pour la base de données |
| ShoppingList.swift | Script responsable des requêtes listes de courses pour la base de données |
| Shop.swift | Script responsable des requêtes 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



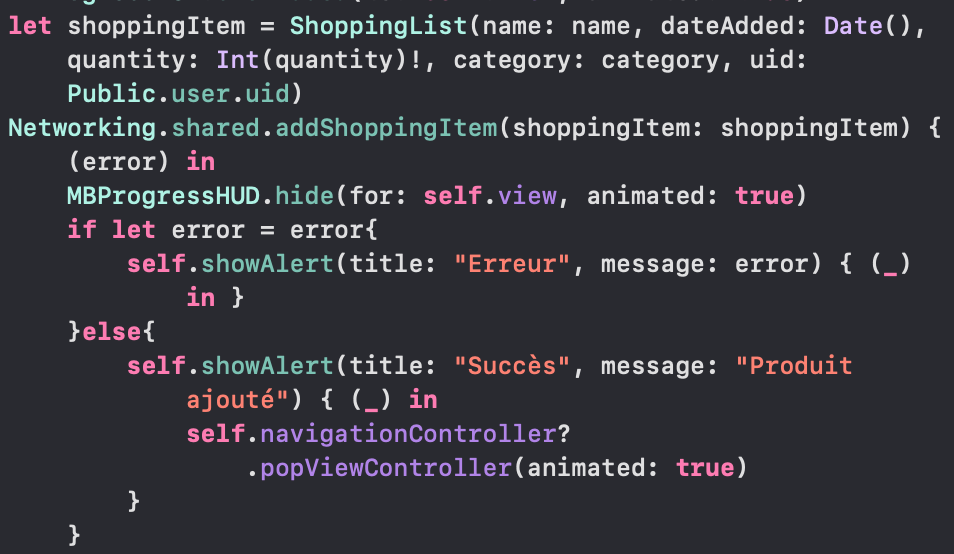
|  |  |
| --- | --- |
|  | = Dossier |
|  | = Fichier swift |
|  | = Interface/View |
|  |  |
|  |  |
|  |  |
|  |  |

### Version de produit

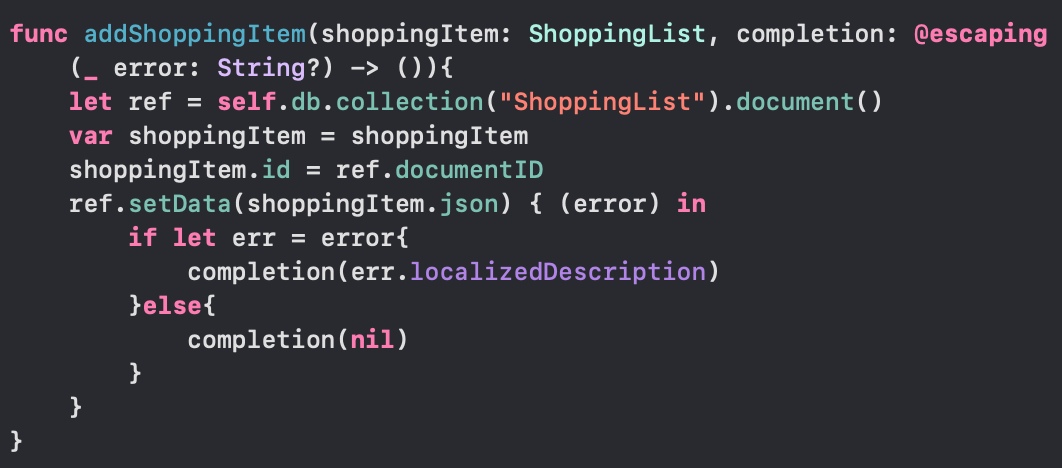
J’ai pendant ce projet fait un commit chaque jour sur le dépôt git[[9]](#footnote-9). J’ai donc environ une quarantaine de commits. Pour ce qui est de la version du produit, j’estime avoir 4 versions du projet :

* 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 données a été terminée
* La troisième du 26 mai 2021 (commit : code source + commentaire), version où les fonctionnalités principales ont été implantées et fonctionnelles (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 réglés

### Ajouts et modification produit

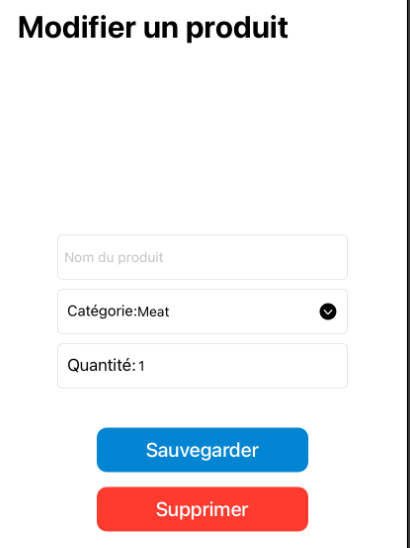
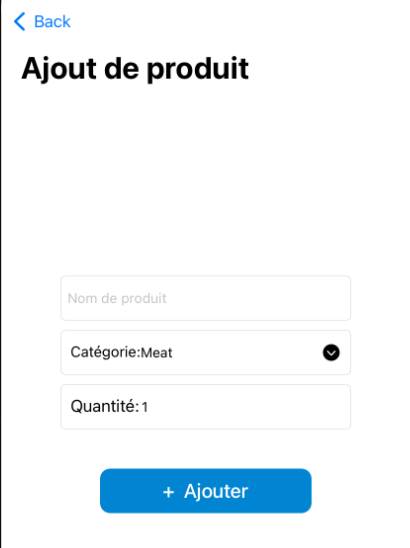
Le système d’ajout et de modification de produit sont pratiquement semblable. En effet, pour ajouter un produit, l’application récupère via les textboxs, la date actuelle et la liste des catégories pour les implanter directement dans la base de données. En indiquant l’uid de l’utilisateur.

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 utilisée 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 magasins prédéfinis, je les ai donc ajoutés 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 prédéfinis sont recréés 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 créé. Et dans un **troisième** temps créee les .json des magasins avec les données prédéfinies et le uid de l’utilisateur :



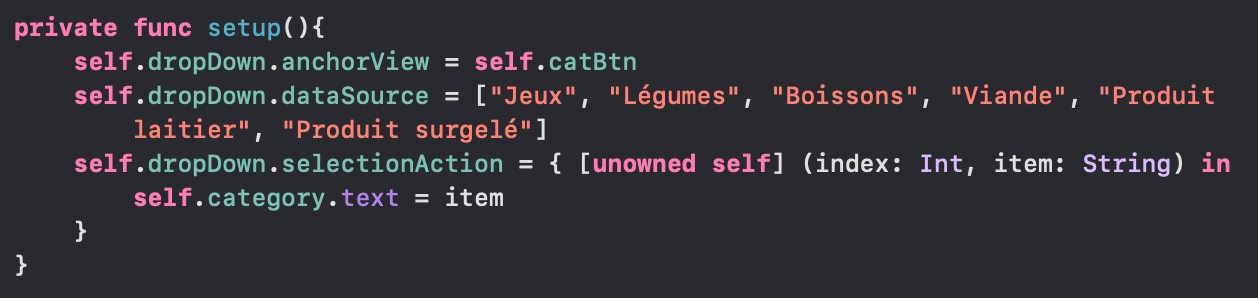
**Deuxième**

**Premier**

**Troisième**

### DropDown

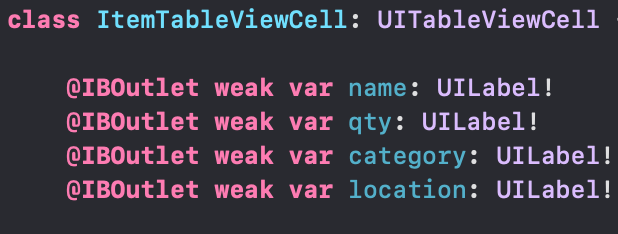
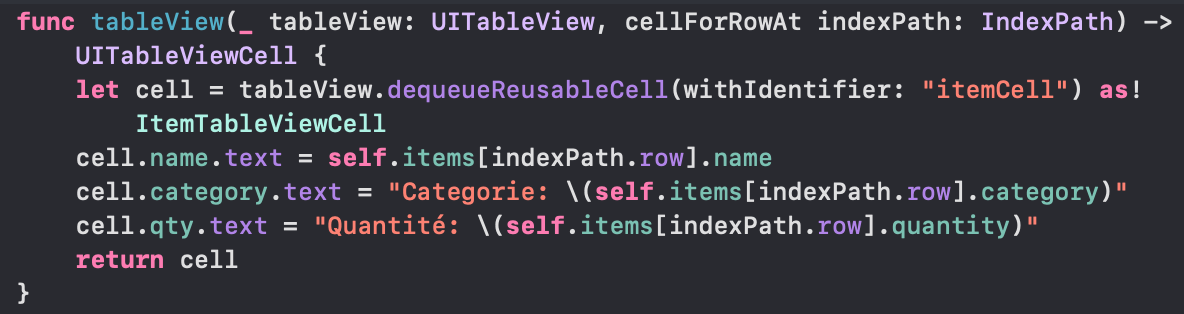
J’utilise pour mes listes la librairie DropDown, j’utilise une liste pour l’ajout de produit, la modification de produit et pour la modification de l’ordre des catégories dans les magasins. Le code est composé de la datasource (données dans la liste) qui est entré en dur. La mention de l’index en int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

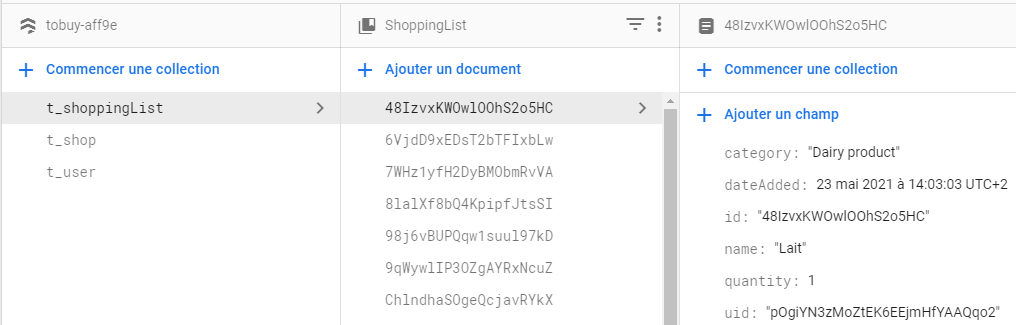


|  |  |
| --- | --- |
| **ItemTableViewCell** | |
| **Représentation visuelle** | **Contenu de cellule list** |

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

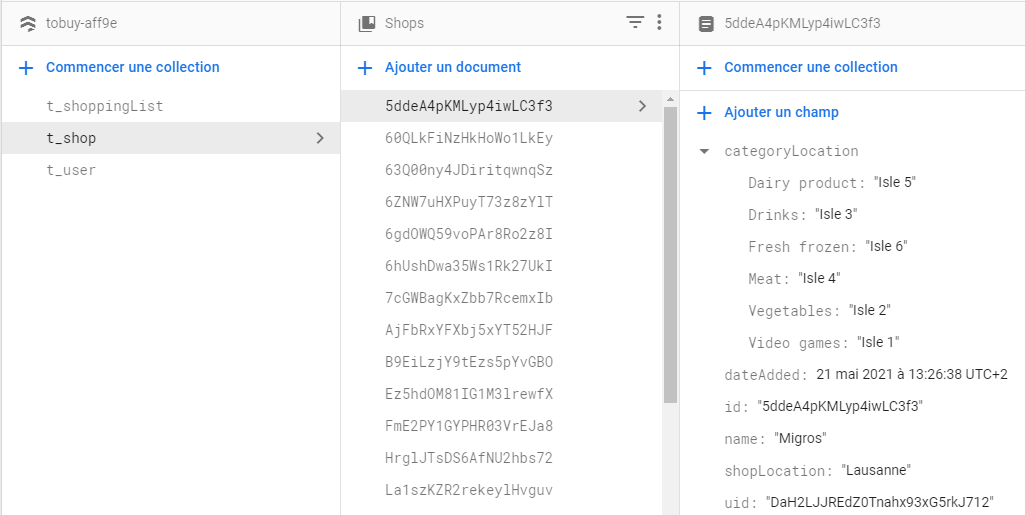
Crée une variable cell en indiquant d’utiliser la cellule (ItemTableViewCell). Ensuite, indique le nom la category et la quantité de la cellule. Et pour finir retourne la cellule.

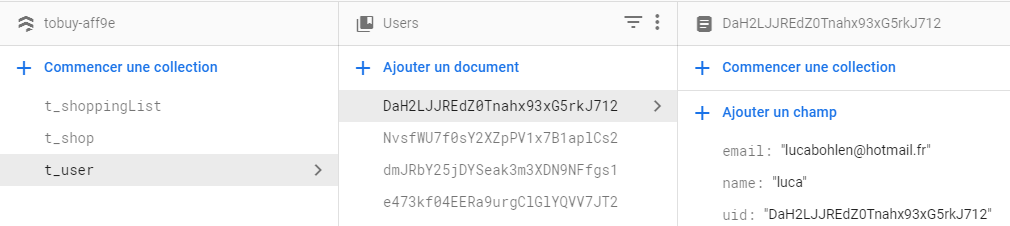
### Dictionnaire des données

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 magasins prédéfini de chaque utilisateur :



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

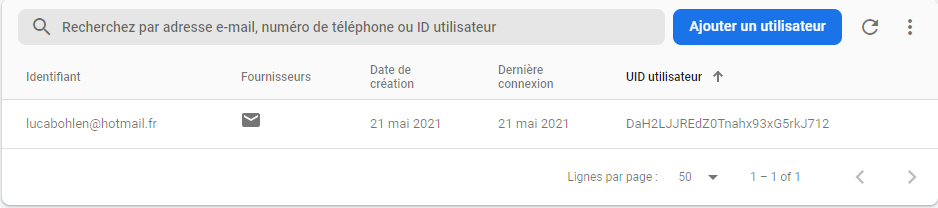
### Changement base de données

Au début du projet, je pensais utiliser la base de données proposée 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.

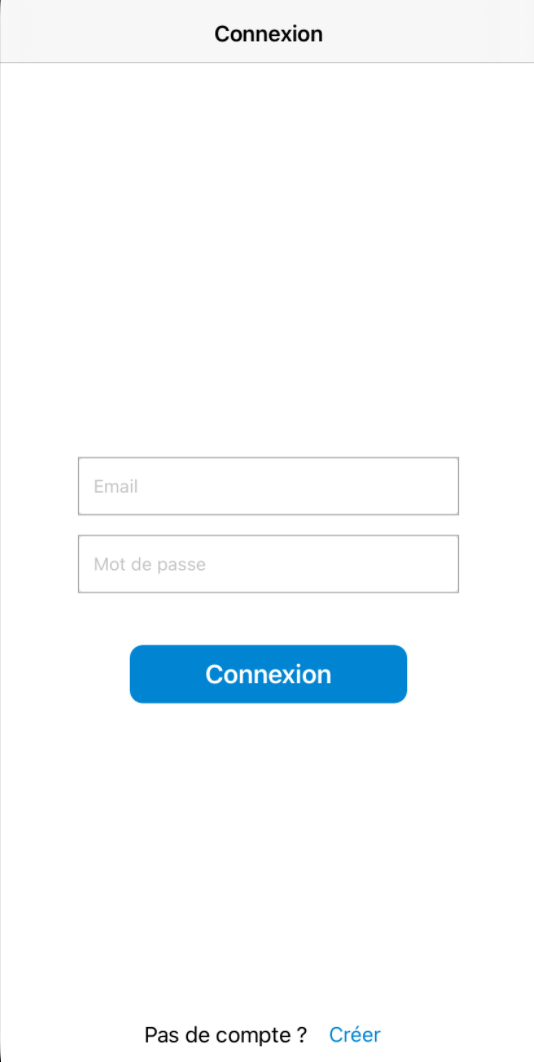
Core data est un framework utilisé pour modéliser des objets dans l’application. C’est le « front end » du stockage des données, où le « back en » doit être en SQL ou des plists. Il s’agit plus d’un concept d’utilisateur unique qui stocke des données localement.

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 dû 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. J’avais besoin de ces pages depuis la création de ma base de données firebase :

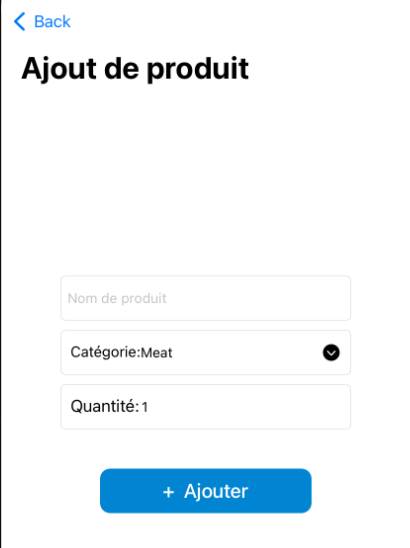
|  |  |
| --- | --- |
| **Page de connexion** | **Page d’inscription** |

Pour la page de liste de course, j’ai changé l’aperçu des produits. Au départ, je voulais les placers dans un simple tableau, mais dès que j’ai vu la possibilité de les mettre dans les cellules, j’ai décidé de changer :

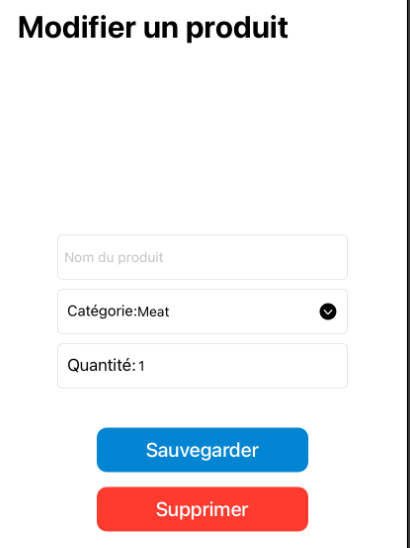
|  |  |
| --- | --- |
| Maquette | Interface réelle |



Ensuite, j’ai changé mon moyen d’ajout de produit. Grâce à la base de données, au lieu de simplement permettre à l’utilisateur de choisir des produits prédéfinis, il peut désormais en ajouter des personnalisés (ajouter, modifier, supprimer). J’ai donc dû créer des pages rendant possible cela :



|  |  |
| --- | --- |
| **Page ajout de produit** | **Page de modification** |



Pour ajouter un magasin et une catégorie dans le magasin, c’est le même principe et template.

Les modifications effectuées ont été décidées pendant le projet afin de garder une bonne ergonomie d’application ou dans le cas des cellules, de coder de manière plus « propre ».

## Description des tests effectués

Les preuves de chaque tests se trouvent dans la vidéo de démonstration de l’application :

Afin de simplifier la lecture du tableau des tests effectués, voici les symboles que seront utilisés avec leurs descriptif

|  |  |
| --- | --- |
| **Symbole** | **Déscriptif** |

|  |  |
| --- | --- |
| 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 |
| nada | Symbole indiquant qu’aucuns tests n’a été éffectué |

|  |  |  |  |
| --- | --- | --- | --- |
| Description | Conséquence | Résultat | Date du test |
| Tester si le bouton « invisible » de la page accueil permet de passer à la page « Ma liste de courss » | Page de launchscreen sans logo appuyable avec un chargement suivant l’appui. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\check.png | 04.06.2021 |
| 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 » | Bouton « ajout de produit » sur la page « Ma liste de courses » fonctionnel, permet de changer de page et ensuite d’ajouter un produit. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\check.png | 04.06.2021 |
| Tester si le bouton « choix de modèle » de la page « Ma liste de courses » permet de passer à la page « Liste es magasins » | Bouton « magasin » sur la page « Ma liste de courses » fonctionnel, permet de changer de page. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\check.png | 04.06.2021 |
| Tester le visuel de ma liste de courses, voir si les produits ajoutés et supprimés sont correctement présents ou non | Cellules des produits bien synchronisées à l’ajout ou la suppression de produit. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\check.png | 04.06.2021 |
| Tester la page « Liste de produits » pour vérifier que les produits sont bien classés | Page non existante au vu du changement du mode d’ajout de produit. Amélioration de version. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\nada.png | - |
| Tester le bouton permettant d’ajouter un produit dans la liste des courses. Possibilité d’incrémenter ce nombre en re appuyant dessus | Incrémentation ou décrémentation directement via le clavier dans la page « ajout de produit » Amélioration. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\check.png | 04.06.2021 |
| 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 | Incrémentation ou décrémentation directement via le clavier dans la page « ajout de produit » Amélioration. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\check.png | 04.06.2021 |
| Tester le bouton permettant de choisir un modèle de magasin prédéfini. Le magasin prédéfini correspond au magasin séléctionné. | Le choix du magasin correspond au résultat du magasin. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\check.png | 04.06.2021 |
| Tester le bouton permettant de choisir un modèle de magasin personnalisé. Le magasin correspond au magasin séléctionné. | Une fois, un magasin crée il apparaît dans la page « choix de magasin ». Le résultat correspond au magasin choisi. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\check.png | 04.06.2021 |
| Tester si le bouton « ajout de magasin personnalisé » permet de passer à la page « création de magasin » | Bouton « ajout de magasin » sur la page « choix magasin » fonctionnel, permet de changer de page. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\check.png | 04.06.2021 |
| Permet grâce au boutou « validation » sur la page  « création magasin » de donner à un nom et un lieu au magasin. | Un magasin nécessite d’avoir au moins un nom et un lieu. Une fois crée les détails du magasin correspondant. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\check.png | 04.06.2021 |
| Permet grâce au bouton « validation » sur la page « création magasin » de donner un ordre de catégorie grâce à des combobox | Le bouton « valider » permet de trier l’ordre selon les catégories. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\check.png | 04.06.2021 |
| 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 » | En cas de produit non dans l’ordre, un NA apparaît, l’utilisateur a donc le choix de modifier l’ordre de ce produit. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\check.png | 04.06.2021 |
| Tester si grâce au bouton « ajouter » l’utilisateur peut traiter manuellement un produit non ordré | L’utilisateur peut ajouter la catégorie et l’ordre afin d’ajouter le produit dans le tri. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\check.png | 04.06.2021 |
| Tester le visuel de la page ordre des courses « trie » correctement les produits selon le magasin choisi | La page « ordre de course » trie bien les produits selon le modèle de magasin choisi. | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\check.png | 04.06.2021 |
| 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 » | La liste des courses et effectivement supprimé et l’utilisateur est redirigé vers la page | C:\Users\lucbohlen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\check.png | 04.06.2021 |

## Liste des documents fournis

* Rapport de projet version 21
* Manuel d’installation version 1 (en annexe)
* Manuel d’utilisation version 1(en annexe)
* Journal de travail version 26 (en annexe et fichier)
* Planification initiale/détaillée version 26(en annexe et fichier)
* Modèle de magasin version 1
* Demonstration vidéo version 1
* Code source version 4

# Conclusions

## Améliorations

Pendant le projet, j’ai déjà réalisé des améliorations qui ne se sont pas mentionnées dans le cahier des charges :

* Connexion et inscription
* Ajout de produit personnalisé
* Amélioration graphique cellules

Toutefois, un projet n’est jamais vraiment fini. Par exemple, je ne me suis pas vraiment inquiéter du design de l’application. En effet, je n’ai ajouté que les objets par défauts proposé par Xcode. Les améliorations possibles :

* Amélioration design personnalisé, identité de l’application
* Possibilité d’ajouter des catégories
* Possibilité de « cocher » un produit lors des courses
* Ajouter un plan image d’un magasin
* Diction vocale pour ajouter des produits
* Version Android de l’application

## Bilan des fonctionnalités demandées

100 % des fonctionnalités demandées sont fonctionnelles. J’ai aussi ajouté des fonctionnalités non-demandées qui m’ont permis d’améliorer l’application. Voici un graphique des différentes fonctionnalités réalisées :

## Bilan de la planification

Le plus gros point faible de ce projet a été la planification. En effet, j’ai largement sous-estimé la période de rapport. J’ai planifié 13 h 30 de rapport, mais en réalité ,j’ai presque passé le double dessus. J’ai sur-estimé la documentation et la recherche, je pensais que j’aurai eu plus de difficultés pendant la réalisation. Au niveau des fonctionnalités, je ne pouvais pas avancer sur les dernières fonctionnalités, cependant une fois les premières fonctionnalités telles que la liste de course et la liste des magasins, toutes les autres ont été complétées rapidement. En bref, j’ai sous-estimé la documentation du projet et j’ai mal évalué les fonctionnalités.

## Bilan personnel

Le projet était intéressant de bout en bout avec quelques difficultés et nouvelles « technologie » qui ont pimenté l’expérience. Je suis content du rendu de l’application que j’ai réalisé. L’entièreté des fonctionnalités sont présentes Concernant mon organisation, j’ai largement sous-estimé les périodes de rapport. J’aurai pu m’organiser autrement lors de mon pré-tpi. Malgré tout, j’ai pu délivrer un projet dont je suis fier et la communication avec mon chef de projet et les experts, c’est selon moi bien passée. J’ai pu lors de ce projet acquérir des connaissances technique comme firebase, mais aussi des connaissances d’organisation de projet, qui selon moi seront utiles lors de mon travail dans le futur.

## Bilan graphique comparaison

# Annexes

## Webographie

Tableau des sources ou des sites que j’ai utilisé sur Internet.

* **Tpivd.ch**, consulté pendant la période de projet, critères d’évaluation : [www.tpivd.ch](http://www.tpivd.ch)
* Trello image, consulté le 06.05.2021, exemple tableau :  
  <https://trello.com/fr/tour>
* Googleplay, consulté le 07.05.2021, exemple design application shopping :

<https://play.google.com/store/apps/details?id=com.l&hl=fr_CH&gl=US>

<https://play.google.com/store/apps/details?id=ch.publisheria.bring&hl=fr_CH&gl=US>

<https://play.google.com/store/apps/details?id=de.markusliebschner.thatlist&hl=fr_CH&gl=US>

* Moqups, consulté le 07.05.2021, site de création de maquette :

<https://moqups.com/>

* Doc coredata, consulté le 07.05.2021, documentation coredata :

<https://www.raywenderlich.com/7569-getting-started-with-core-data-tutorial>

* Doc database, consulté le 07.05.2021, documentation base de données :

<https://qastack.fr/programming/28628225/how-to-save-local-data-in-a-swift-app>

* Openclassroom, consulté le 12.05.2021, documentation cocoapods :

<https://openclassrooms.com/fr/courses/2582746-developpez-votre-premiere-application-pour-ios/3854446-tirez-parti-des-frameworks-bibliotheques-et-add-ons>

* DeveloperApple, consulté le 17.05.021, documentation UIAppDelegate :

<https://developer.apple.com/documentation/uikit/uiapplicationdelegate>

* DeveloperApple, consulté le 17.05.2021, documentation content view :

<https://developer.apple.com/documentation/uikit/uitableviewcell/1623229-contentview>

* Youtube, consulté le 19.05.2021, documentation installation cocoapods :

<https://www.youtube.com/watch?v=TWrtw75dMQs>

* Raywenderlich, consulté le 19.05.2021, documentation UICollectionView :

<https://www.raywenderlich.com/18895088-uicollectionview-tutorial-getting-started>

* GoogleFirebase, consulté le 19.05.2021, documentation firebase :

<https://firebase.google.com/>

* GoogleFirebase, consulté le 20.05.2021, documentation firebase :

<https://firebase.google.com/docs/ios/setup>

* GoogleFirebase, consulté le 20.05.2021, documentation firebase :

<https://firebase.google.com/docs/ios/setup>

* Youtube, consulté le 20.05.2021, problème module firebase :

<https://www.youtube.com/watch?v=VObEHfbjcn0>

* StackOverFlow, consulté le 20.05.2021, problème module firebase :

<https://stackoverflow.com/questions/37850498/no-such-module-firebase-when-add-firebase-into-a-library>

* Console firebase, consulté depuis le 21.05.2021, console firebase :

<https://console.firebase.google.com/?pli=1>

* DeveloperApple, consulté le 21.05.2021, problème bouton crash :

<https://developer.apple.com/forums/thread/102811>

* Openclassroom, consulté le 21.05.2021, ajout liste application :

<https://openclassrooms.com/fr/courses/4504866-ajoutez-des-listes-dans-vos-applications-ios/5337840-remplissez-la-table-view>

* StackOverFlow, consulté le 26.05.2021, problème area storyborad :

<https://stackoverflow.com/questions/48388844/storyboard-safe-area-wrong-fixed-width-for-any-device>

* Google Firebase, consulté le 28.05.2021, documentation connexion :

<https://firebase.google.com/docs/auth/web/firebaseui>

* Grafikart, consulté le 28.05.2021, documentation noSQL :

<https://grafikart.fr/blog/sql-nosql>

* Openclassroom, consulté le 28.05.2021, documentation noSQL firebase :

<https://openclassrooms.com/fr/courses/4872916-creez-un-backend-scalable-et-performant-sur-firebase/4982769-creez-votre-premiere-structure-de-donnees#:~:text=Sur%20Firebase%2C%20il%20existe%20deux,aux%20bases%20de%20donn%C3%A9es%20NoSQL>.

* Looping, consulté le 28.05.2021, site création MCD :

<https://www.looping-mcd.fr/>

* Cacoo, consulté le 02.06.2021, site création organigramme :

<https://cacoo.com/>

## Manuel d’utilisation

Vidéo démonstrative du projet ci-dessous :

[..\5. Autres\5.5 Demonstration\MM-TPI-LucaBohlen-Demonstration.mp4](../5.%20Autres/5.5%20Demonstration/MM-TPI-LucaBohlen-Demonstration.mp4)

## Archives du projet

Toutes les versions du projet sont sur un dépôt github en public :

<https://github.com/LucaBohlen/TPA-ToBuy>

## Résumé rapport

**Situation de départ**

Pour mettre en place mon application iOS développée en langage de programmation swift, je disposais d’un mac prêté par l’école. Avant de commencer mon TPI, pendant la période de pré-tpi, j’ai eu une période d’analyse pour avoir des données réelles et cohérentes.

**Mise en œuvre**

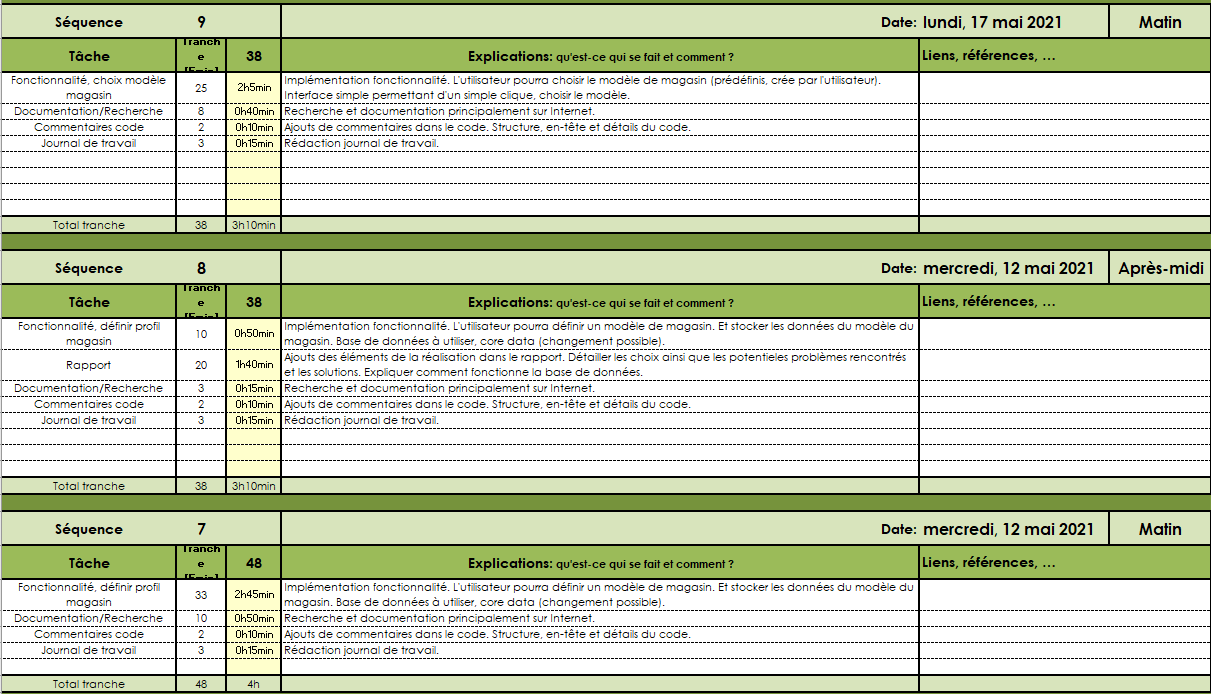
Je suis passé d’une phase de documentation d’analyse et de conception. Les mock-ups du projet ont été réalisés dans les temps et on permit de suivre une ligne directrice pour l’interface de l’application. Ensuite, un changement de base de données a été effectué pour avoir une database plus cohérente au projet. Après cela, une documentation poussée de l’utilisation de la base de données firebase a été effectuée sans modifier pour autant la planification initiale. J’ai réalisé pendant la réalisation des tests pour validé les fonctionnalités récemment intégrées. Le format NoSQL de la base de données m’a permis d’avoir des données constamment synchronisées dans les fichiers.json. Pour simplifier l’intégration de l’interface graphique dans le story-board, j’ai installé les framework de la bibliothèque cocoapods. L’utilisation de l’architecture logicielle « modèle-vue-controller » a été dans ce projet judicieux. Mes vues étant les pages dans le story-board intégré avec swiftUI. Mes modèles sont des scripts répertoriant les requêtes des tables « t\_shop, t\_shoppingList, t\_users ». Et pour finir, les controllers sont des scripts gérant les erreurs et les utilités des objets présents sur les pages. Pendant la réalisation du projet, les interfaces ont été modifiées afin de correspondre au mieux aux fonctionnalités, tout en restant le plus possible fidèle aux mock-up. Les modifications sont expliquées en détails dans le rapport. Pour finir, j’ai réalisé une période de test en fin de projet afin de pouvoir valider une dernière fois les fonctionnalités.

**Résultats**

Les résultats attendus sont satisfaisants, le projet manque possiblement d’une meilleure interface plus userfriendly. Cependant, toutes les fonctionnalités ont été développées et sont fonctionnelles. J’ai rajouté une inscription ainsi qu’une connexion afin d’avoir des listes de courses uniques à chaque utilisateur. Des améliorations tel que le graphisme ou par exemple une version Android de mon application pourrait être réalisé avec deux semaines de plus. La documentation me semble correcte pour moi avec les informations nécessaires au projet. Le point négatif avec ce projet a été la planification, j’ai largement sous-estimé la charge de travail lié à la documentation. Cependant, je n’ai pas eu de problème pour finir mon application, aucune période de travail n’a été effectué hors des heures planifié. Au niveau de la communication avec le chef de projet ainsi qu’avec les experts, cela s’est bien passé de mon côté.

## Planification Initiale

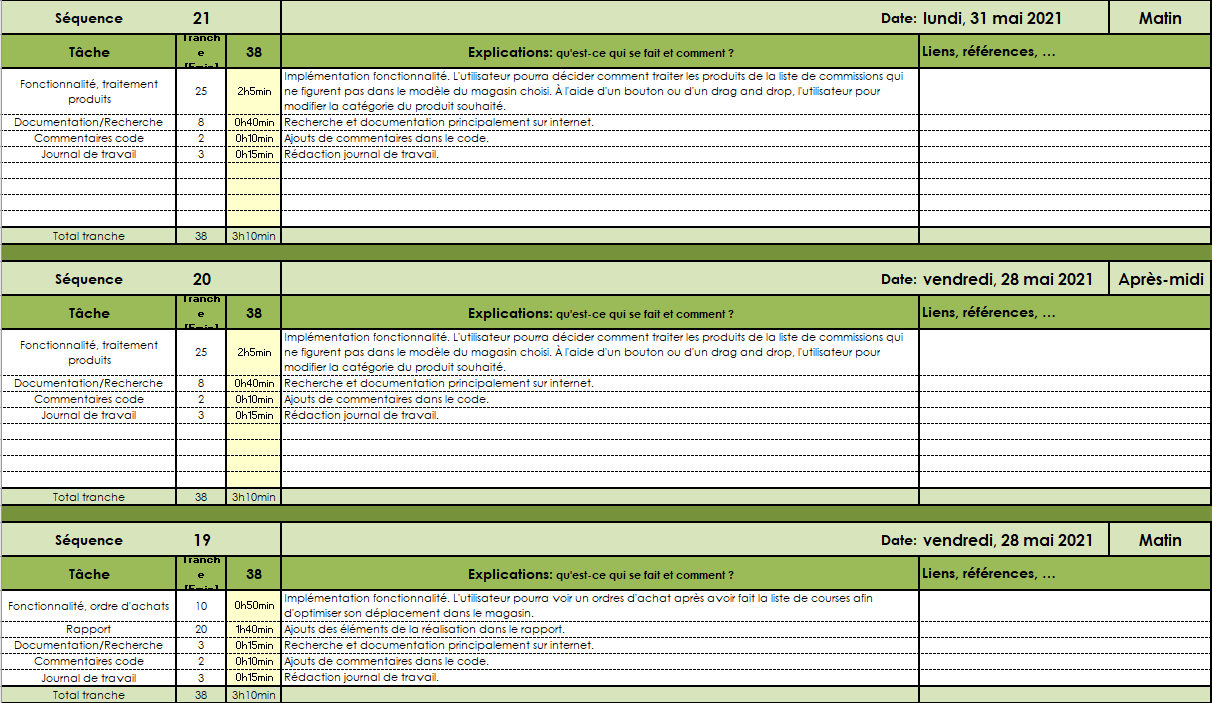


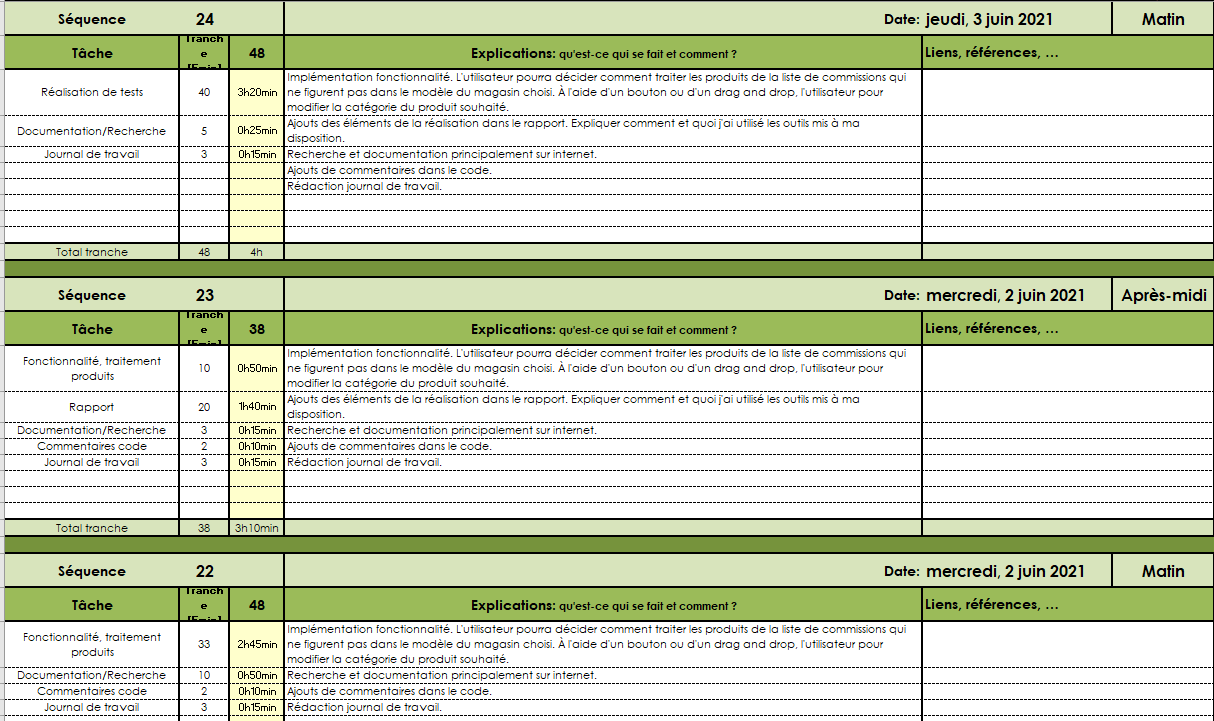


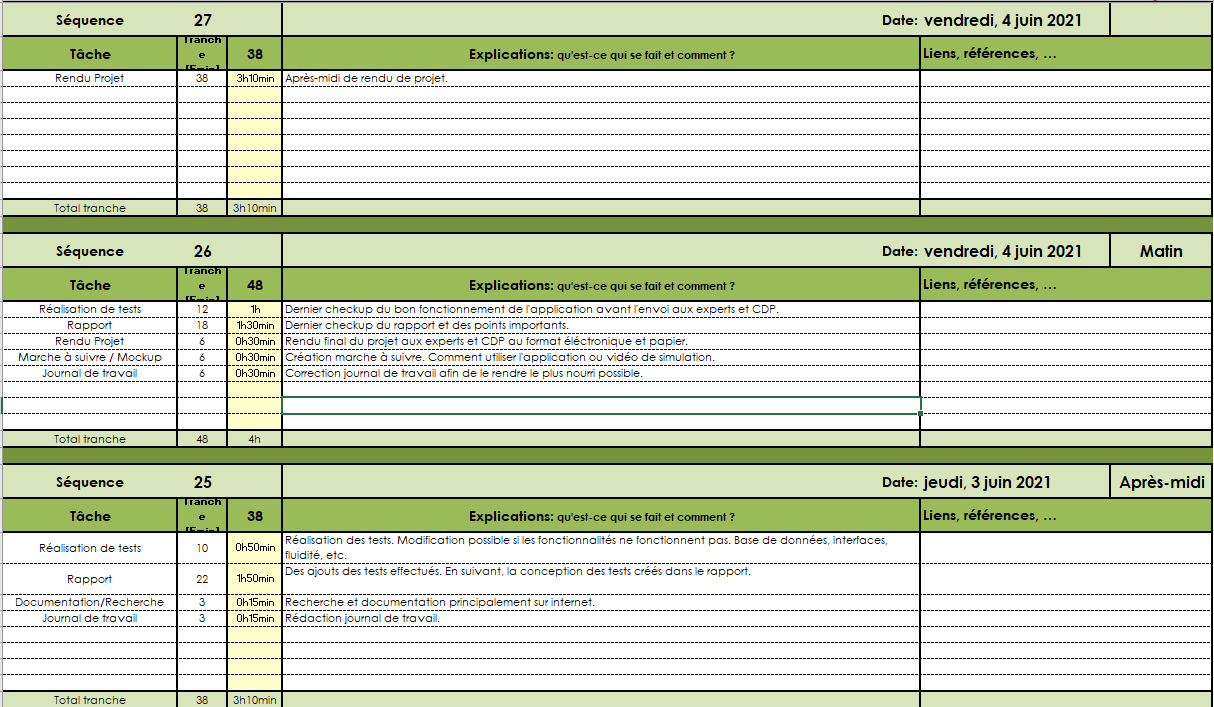




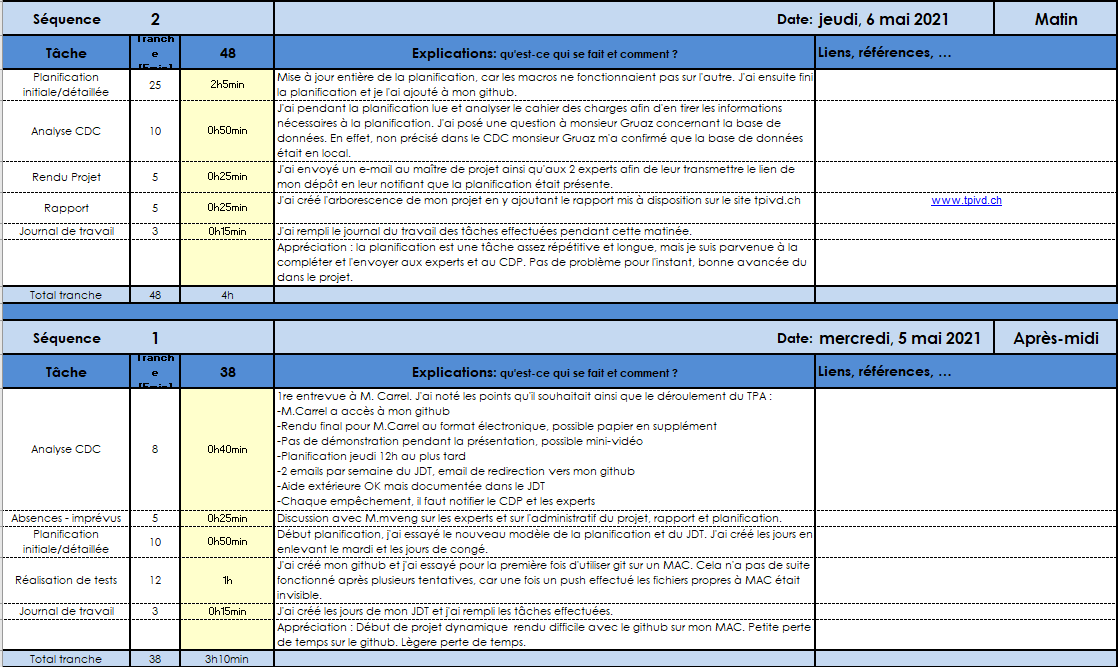


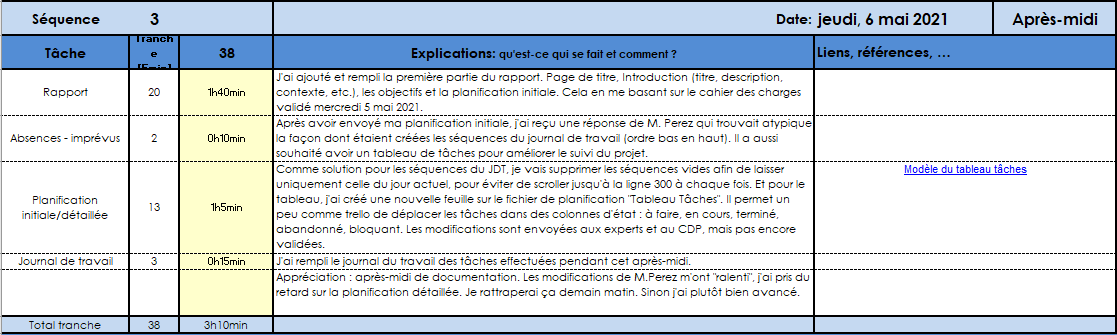
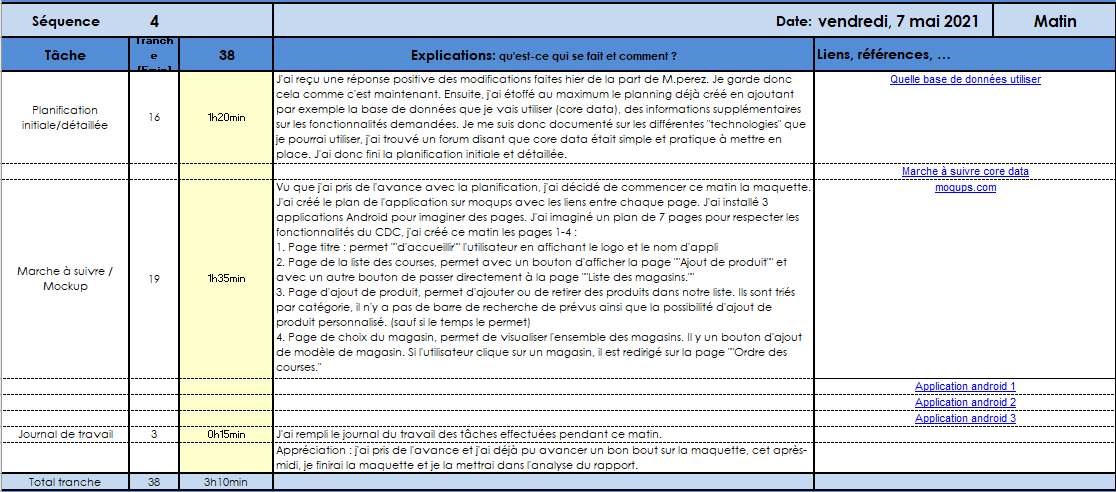


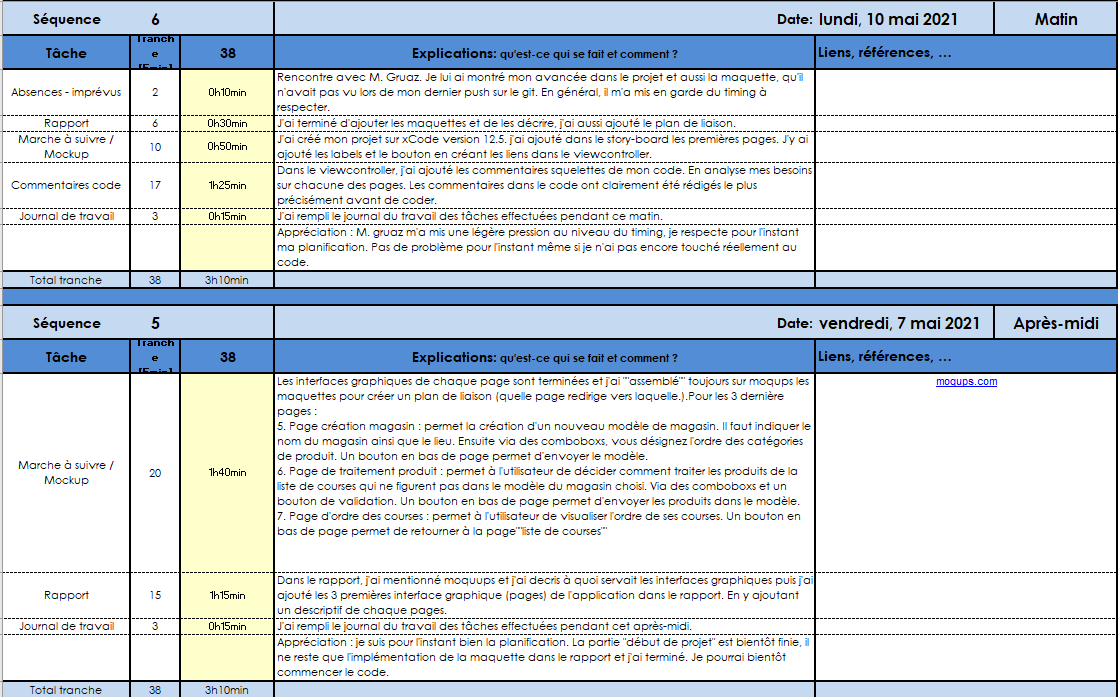




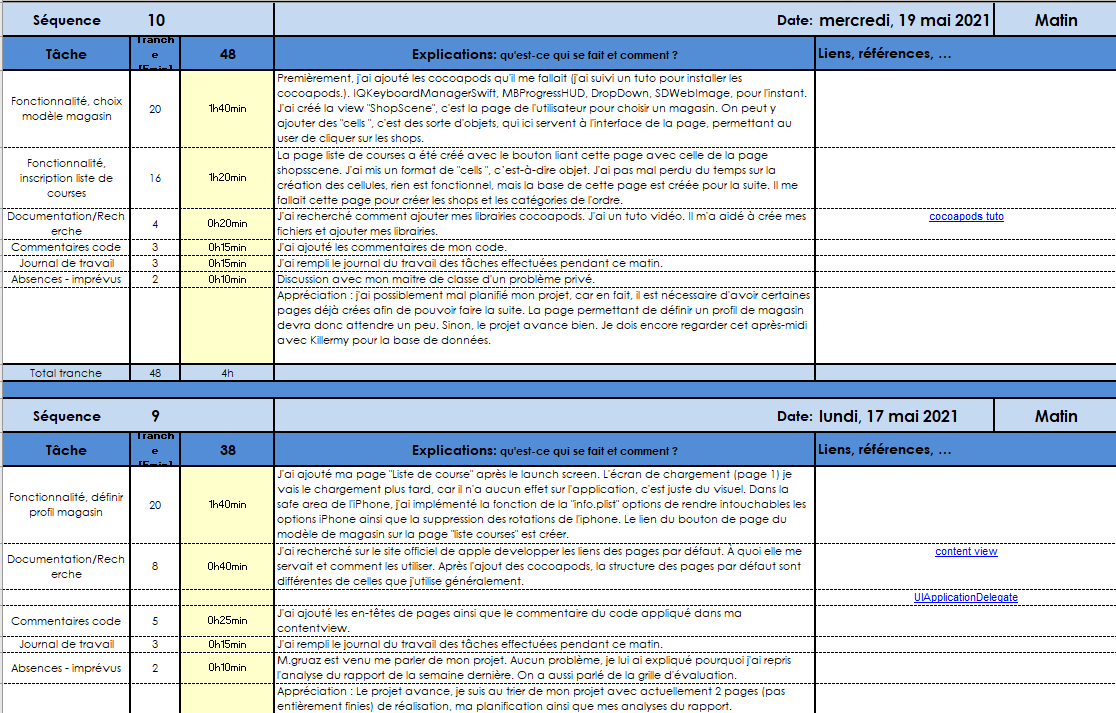
## Journal de travail

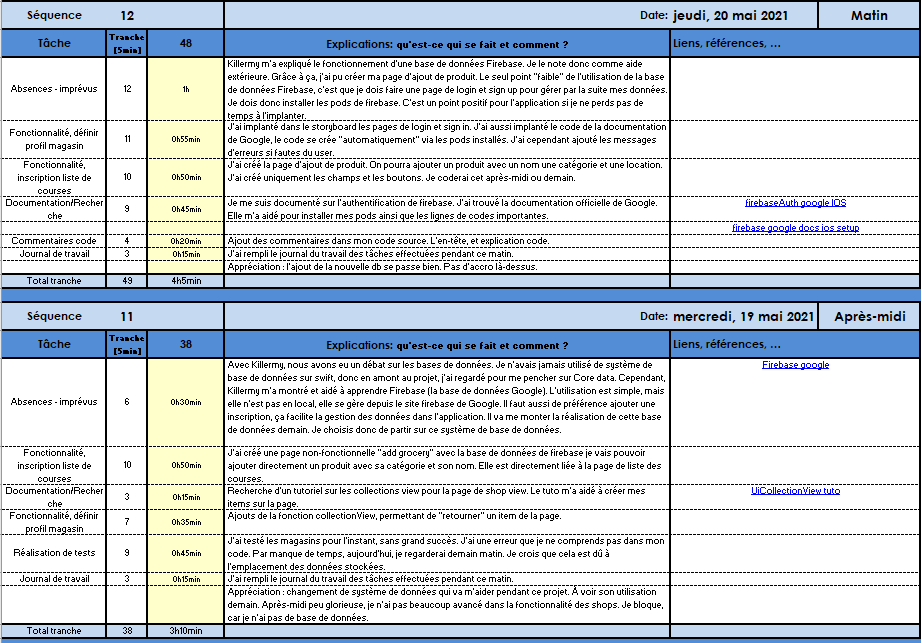


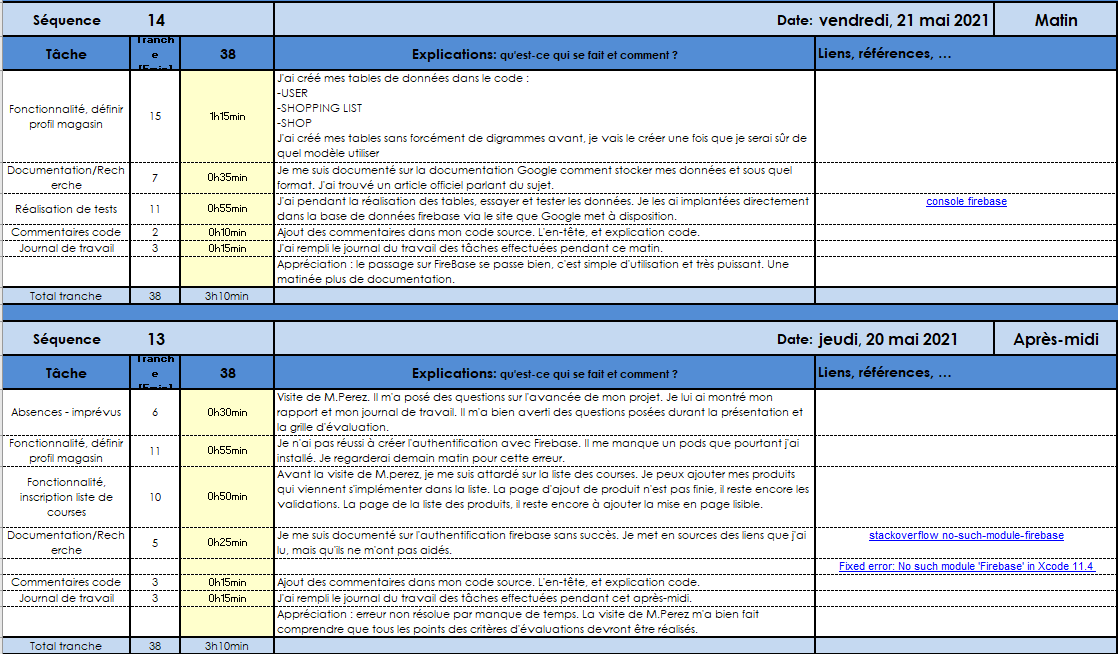


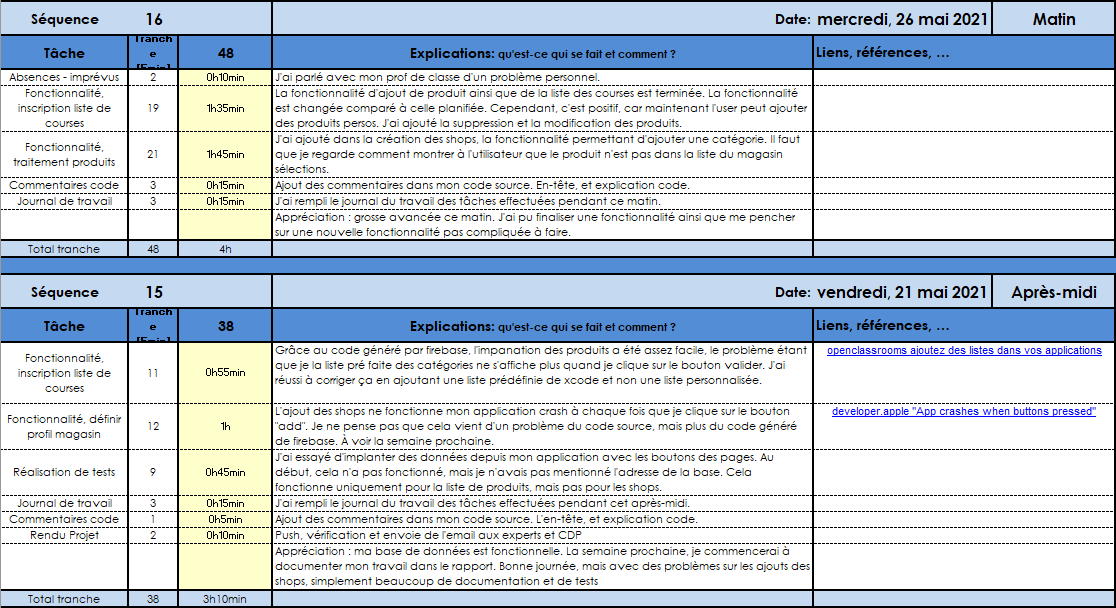


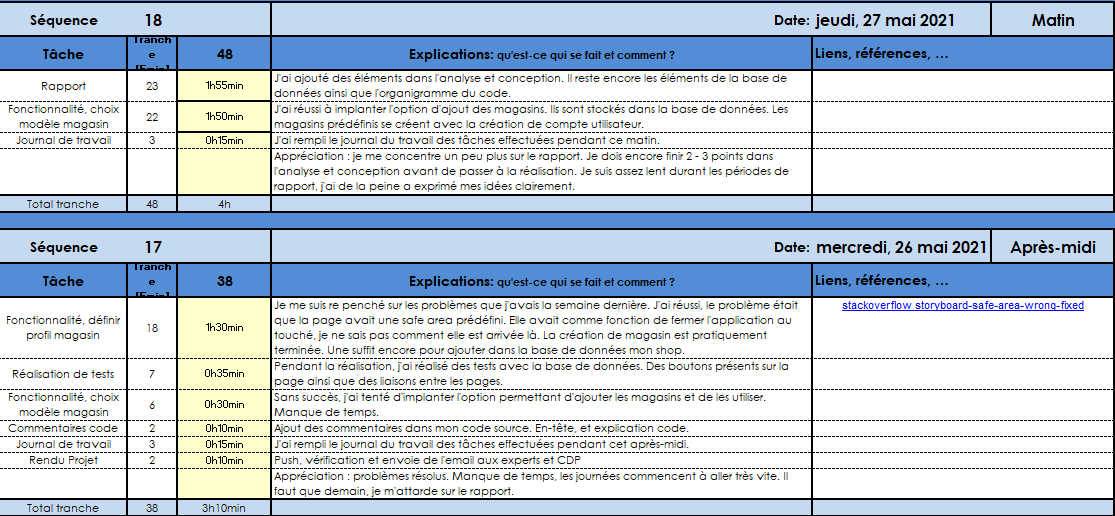






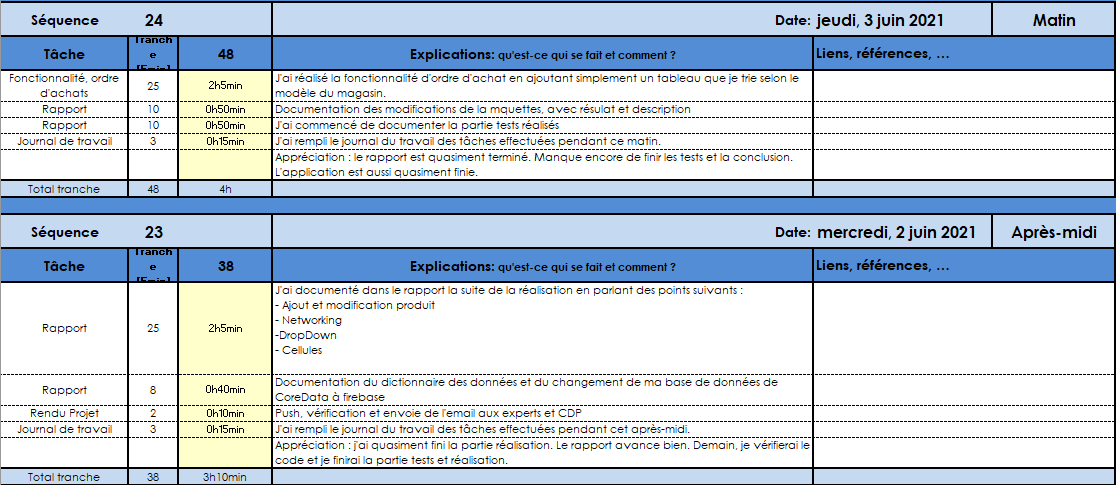


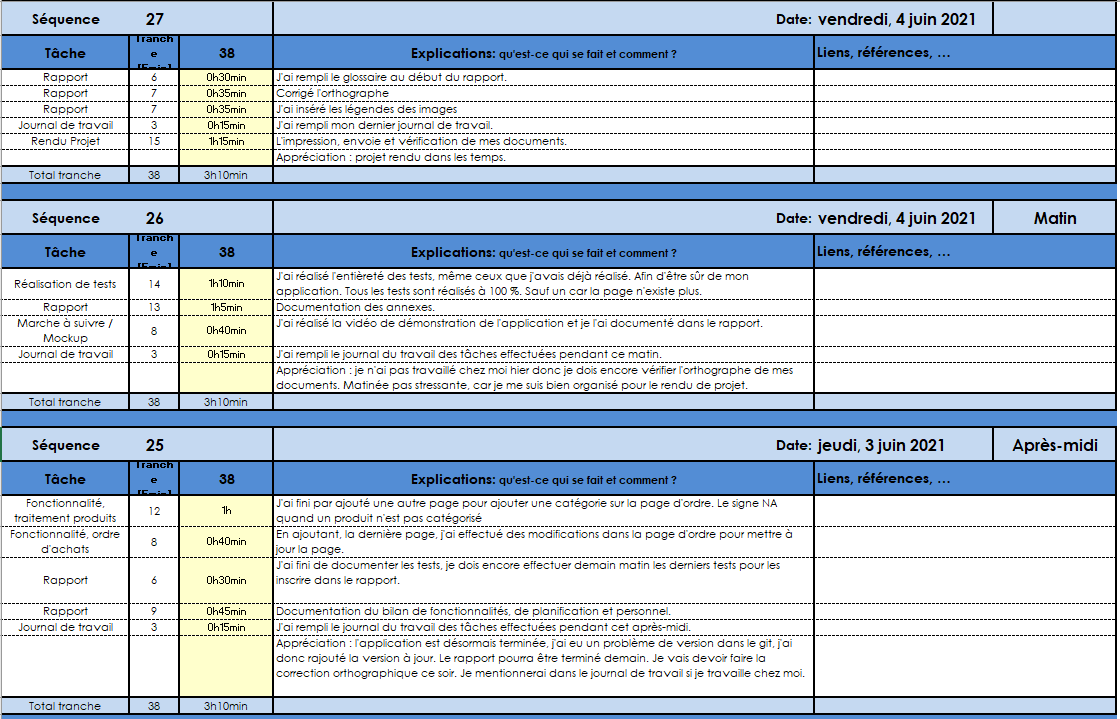




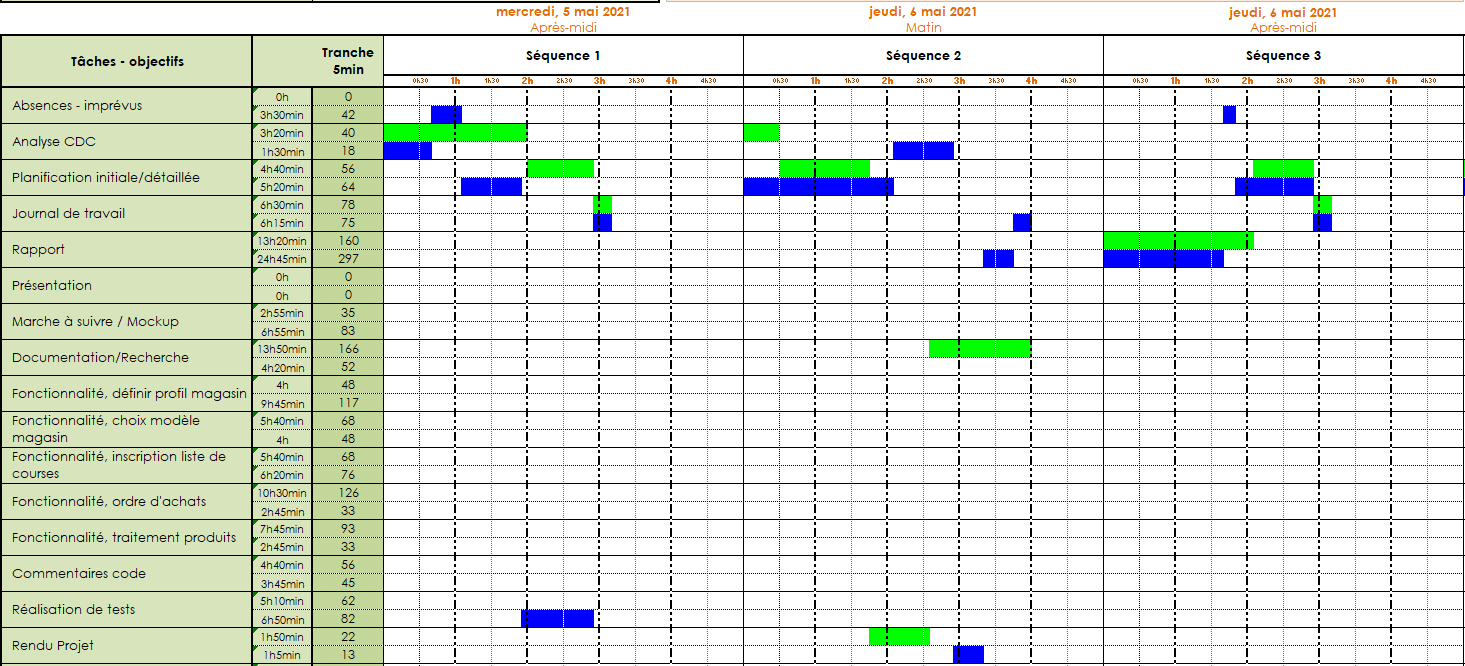


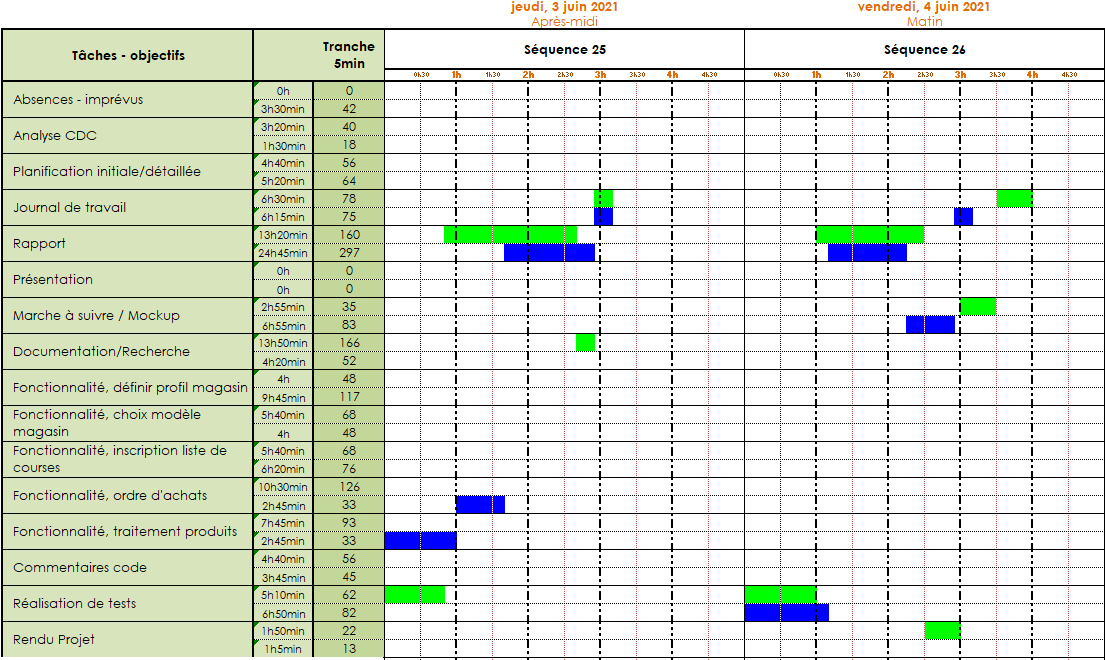
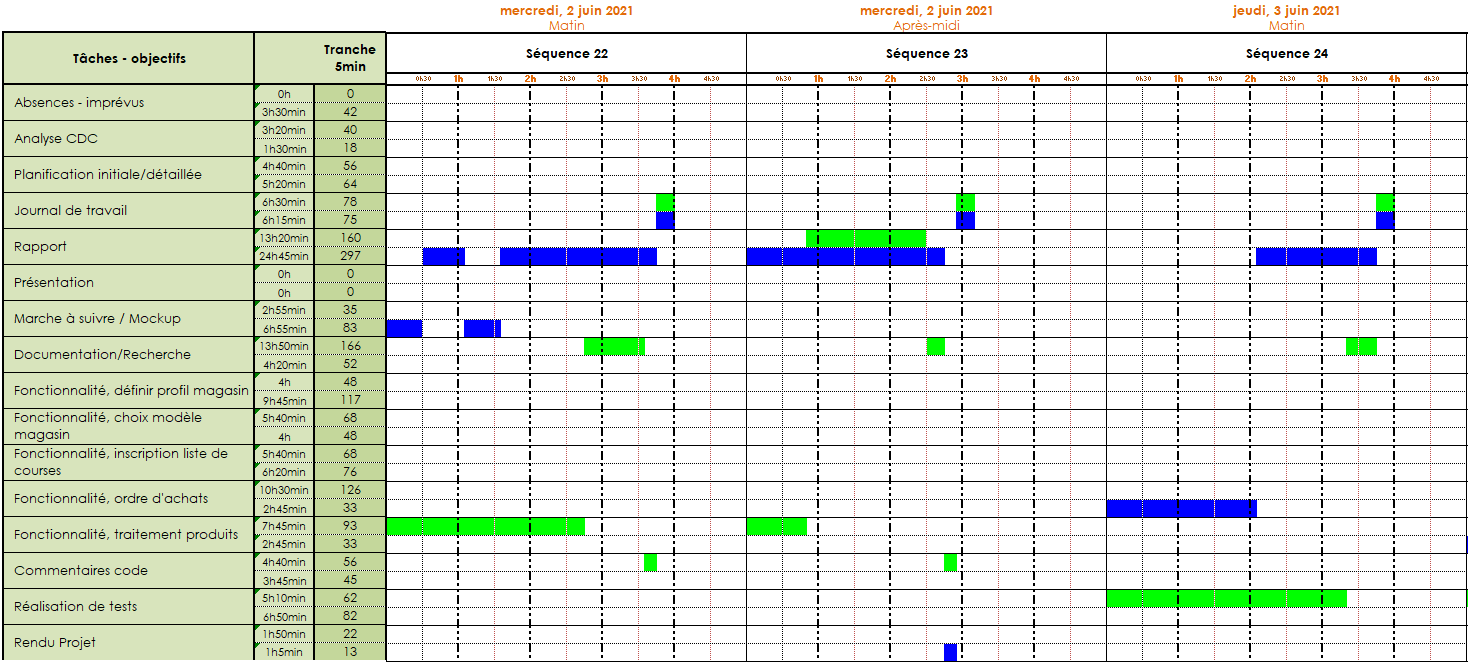
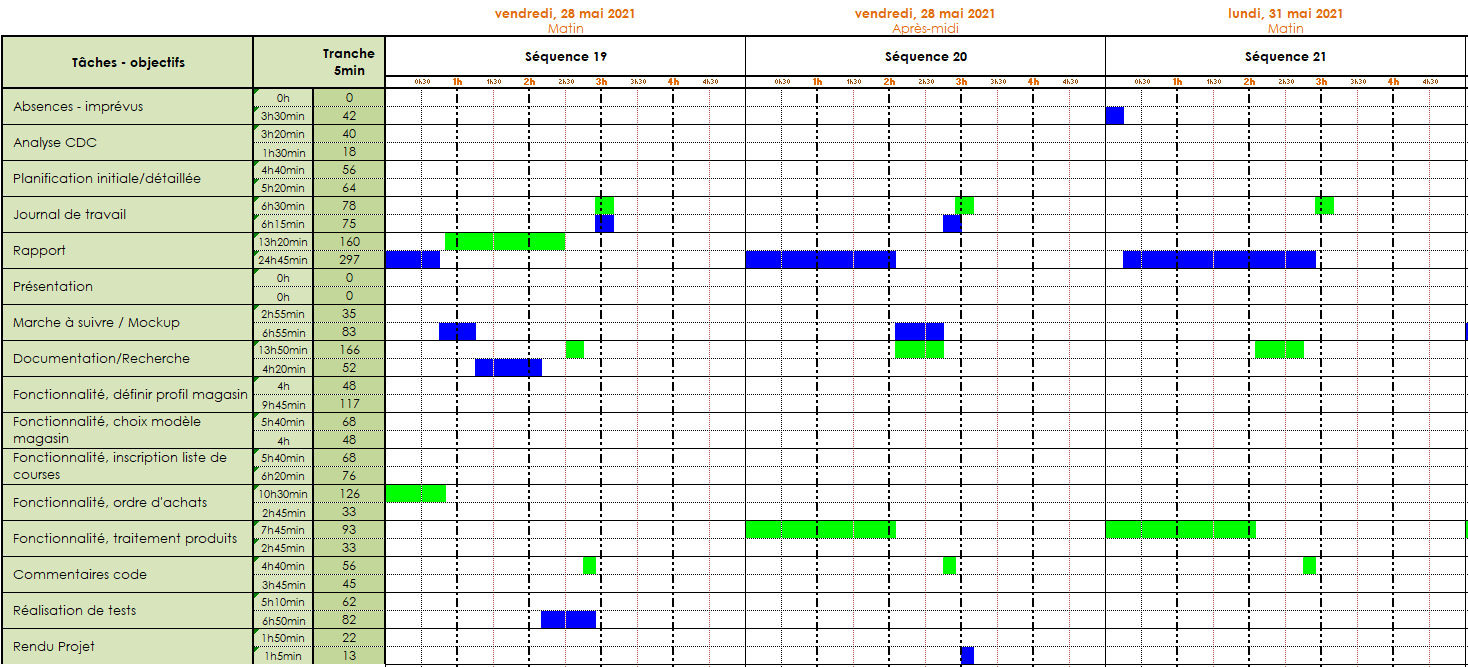
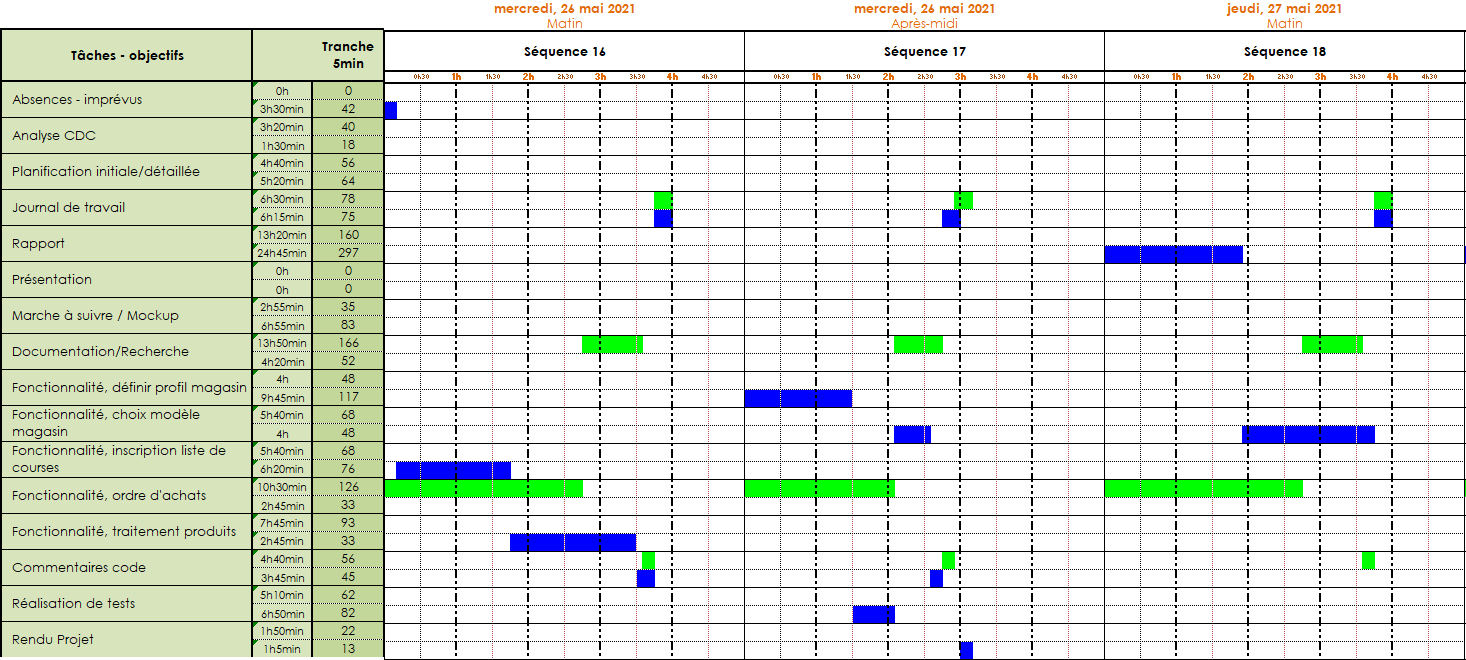
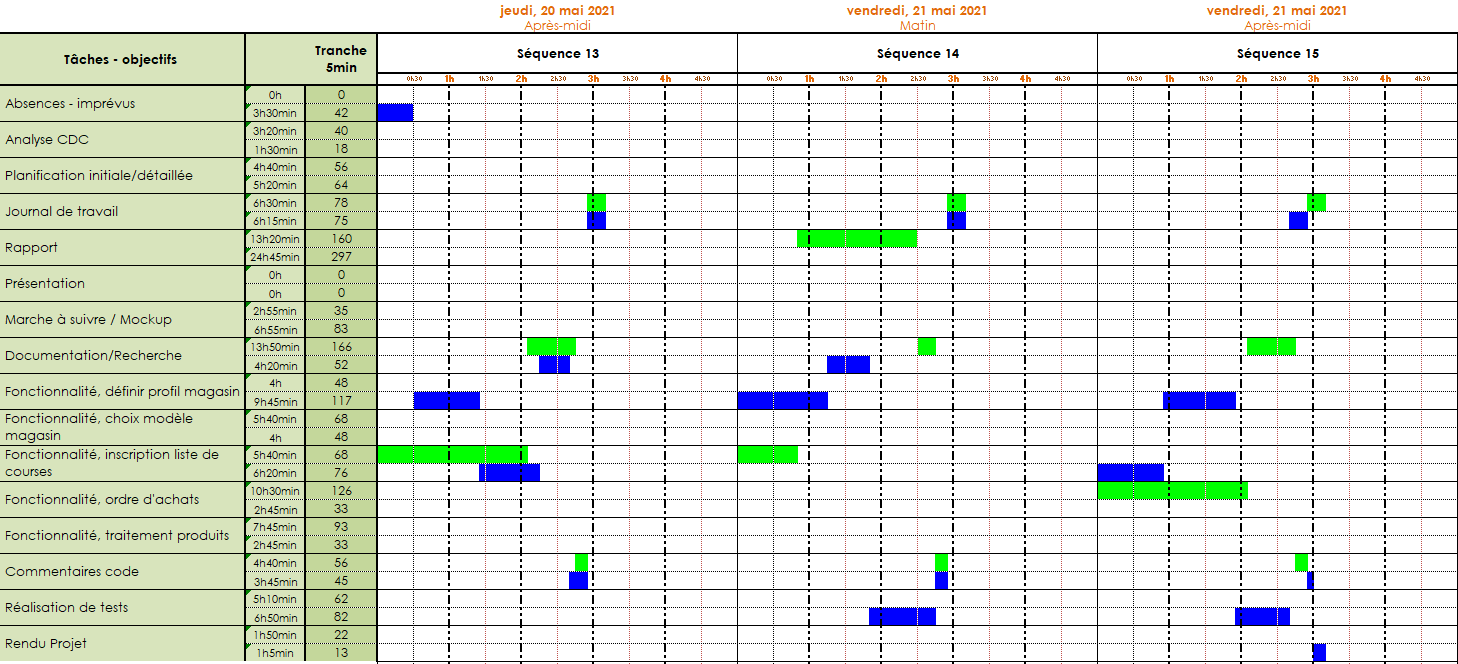
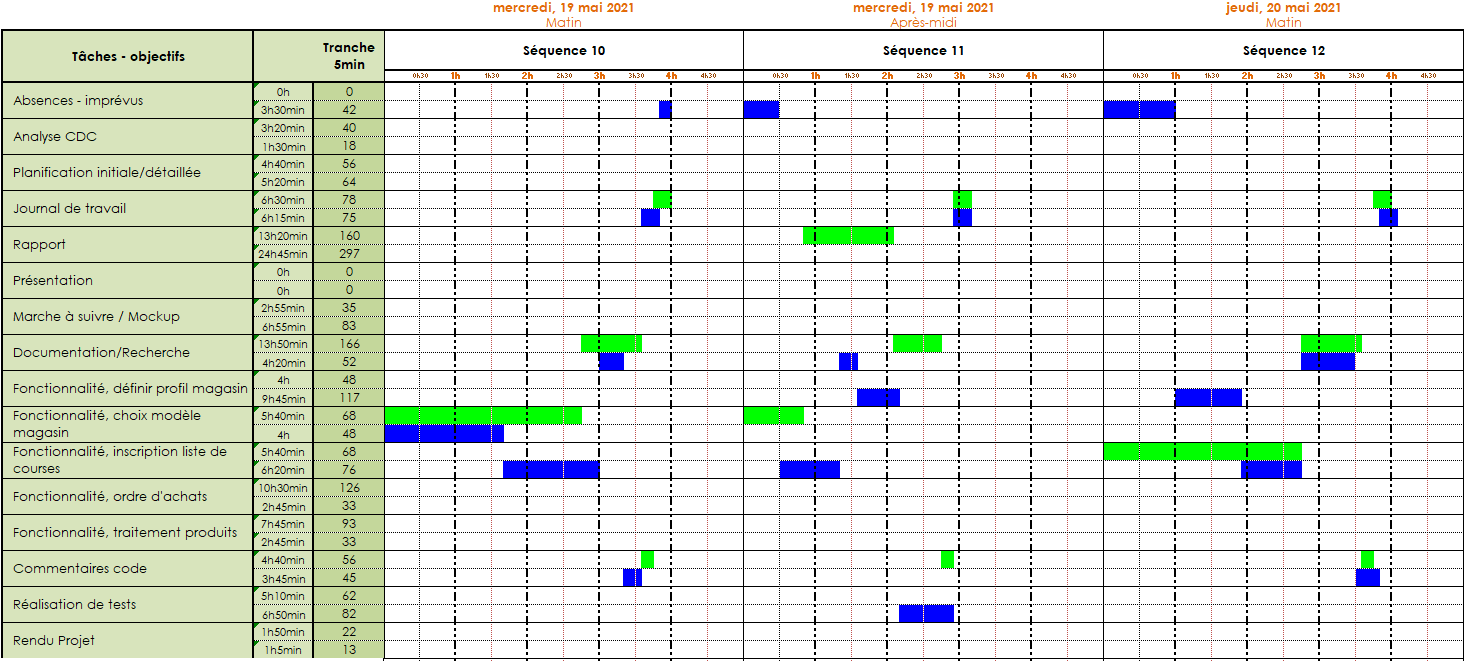
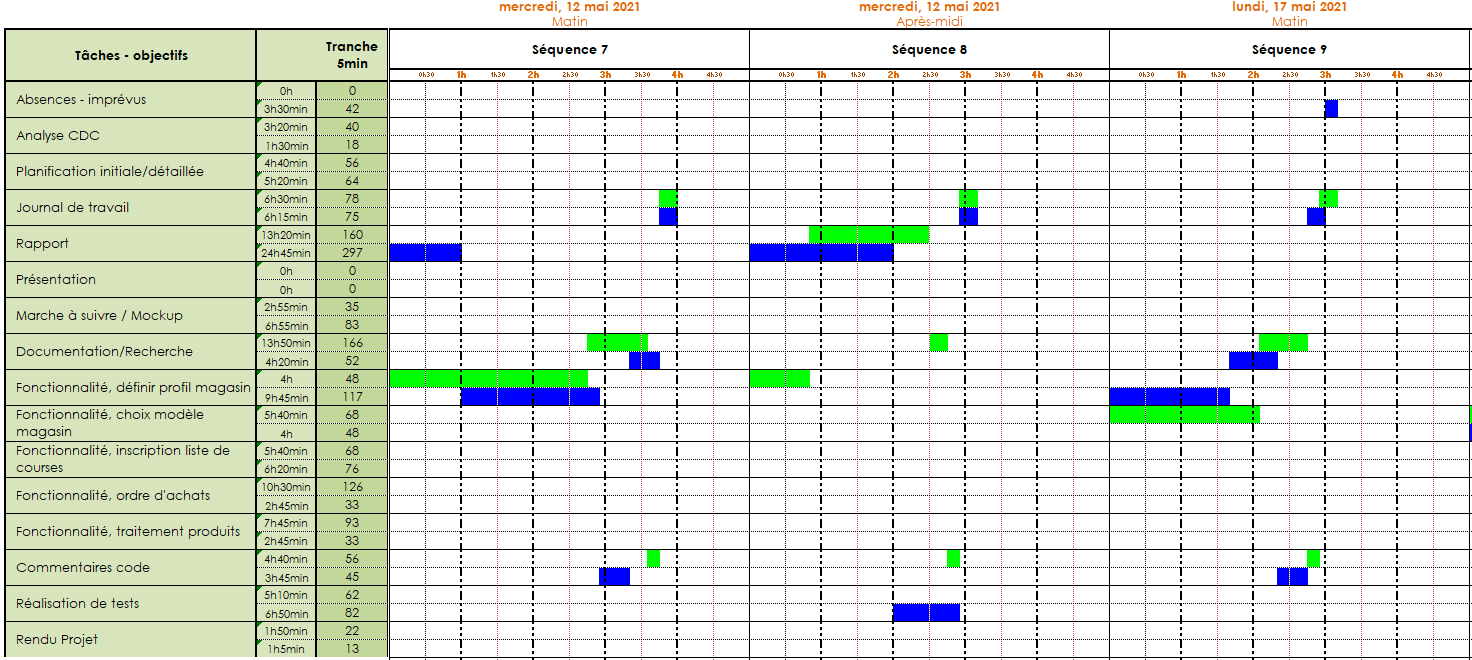
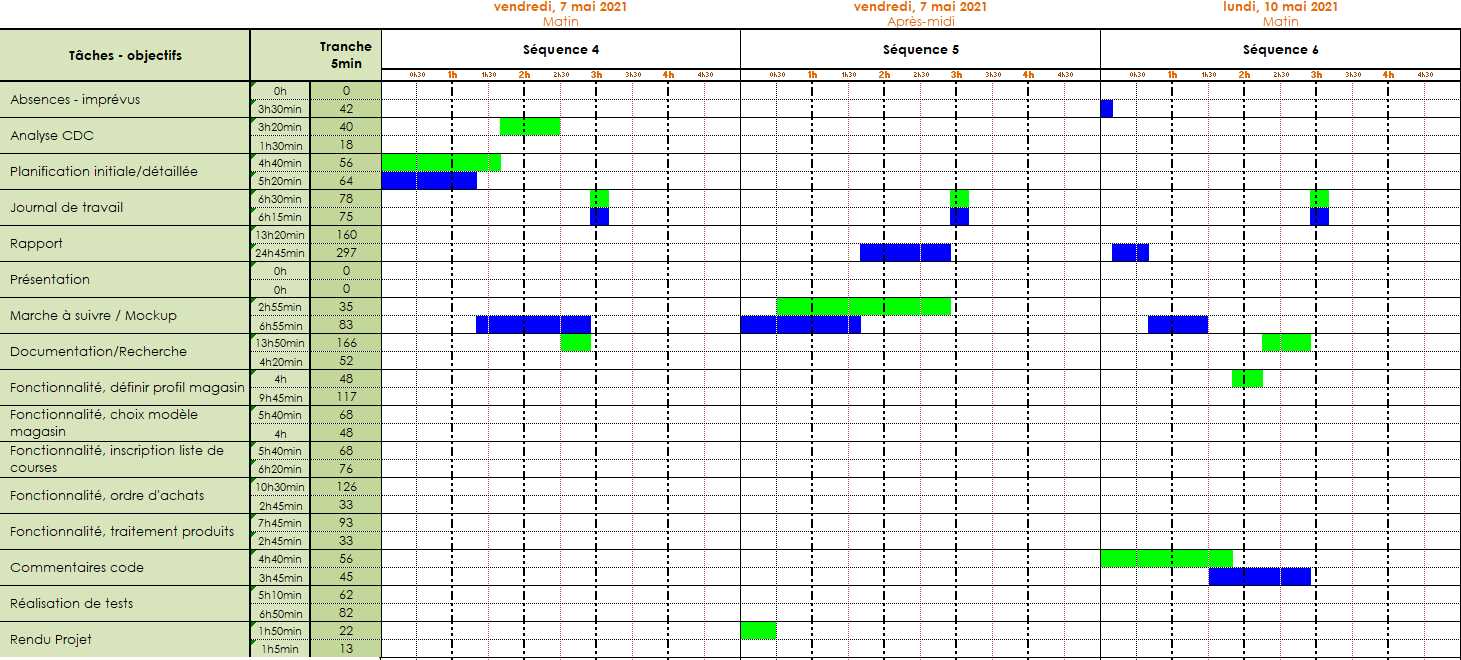






## Bilan graphique planification





## Tableau modèle magasin

## Instructions de mise en œuvre

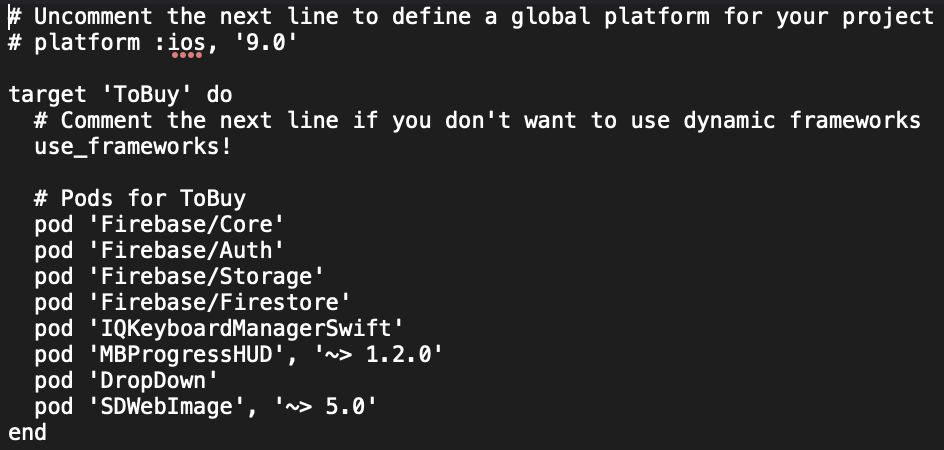
Tout d’abord installer la dernière version du code source sur le dépôt github :

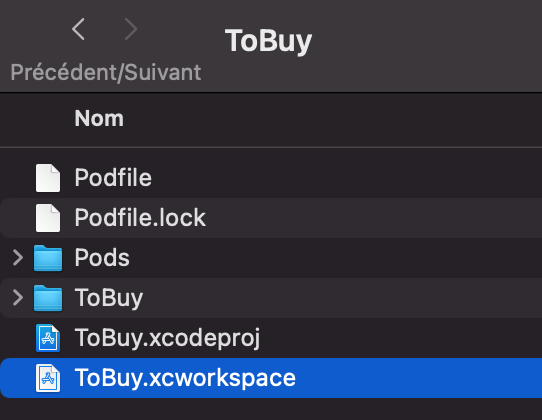
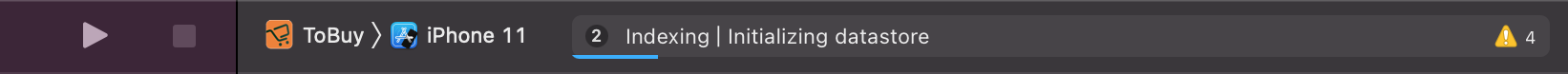
<https://github.com/LucaBohlen/TPA-ToBuy>

### Mac

Il faut installer les pods nécessaire à l’application, pour cela suivre ce tutoriel vidéo ci-dessous : <https://www.youtube.com/watch?v=iv3qMYaHzi0>

Pods à installer :



une fois installé, lancer le code de l’application avec le fichier « ToBuy.xcworkspace »  et build l’application:

### Mobile iOS

Il suffit de brancher un appareil mobile iOS et de build directemenr depuis cet appreil au lieu du simulateur.

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)