UNIVERSITÉ DE LORRAINE | IUT nancy Charlemagne Informatique

Rapport de projet tutoré

CharleMi'App

ARNOUX Guillaume
BRASLEY Maxime
STEINER Noé
VIGNERON Steven

Tuteurs de projet :

M. Alain LONGHAIS
Mme Bénédicte CLEMENT

Responsable pédagogique :

M. Dominique LECHAUDEL

Année 2021-2022

Remerciements

Le projet qui va être présenté et détaillé par ce document est le fruit de recherches et de travaux collectifs.

Nous tenons particulièrement à adresser nos remerciements à toutes les personnes qui nous ont permis de construire ce projet.

Nos remerciements vont d'abord à Monsieur Dominique Lechaudel, qui nous a accompagnés durant ces mois de recherches, de découvertes ainsi que de travail intensif. Nous avons pu, grâce à lui, rester sur la bonne voie et mettre à profit des méthodes agiles dans notre projet afin de toujours avancer dans la bonne direction. Le suivi régulier apporté a été précieux et nous permet, aujourd'hui, de vous présenter un projet dont nous sommes fiers.

Nous adressons également nos plus sincères remerciements au personnel de la cafétéria « Charlemiam », sans qui nous n'aurions pu mener à bien ce projet. Bénédicte et Esméralda ont pu répondre à toutes nos demandes techniques et nous détailler le fonctionnement de la cafétéria, sous forme de journées types par exemple.

Enfin, nous remercions M. Alain Longhais, à l'initiative du sujet, qui a suivi notre travail depuis son commencement et nous a aidés à le perfectionner. Les mises au point effectuées depuis novembre nous ont énormément aidées, surtout pour poser les bases de la réflexion de ce projet.

Ce travail fut, pour nous, d'un intérêt considérable. Nous avons pris plaisir à explorer le sujet. Ainsi, nous souhaitons remercier tous les acteurs ayant apporté leurs connaissances, leur motivation et leur savoir-faire au sein de ce projet.

Table des matières

1. Introduction

1.1 Objet du document

Le présent document a pour vocation de détailler l'intégralité de notre projet. Il doit être compréhensible pour toute personne le lisant.

Dans ce document, nous détaillerons les phases d'analyse, de conception et de réalisation du projet. Nous étudierons ensuite certaines parties techniques telles que les contraintes.

1.2 Présentation du projet

Le projet CharleMi'App résulte en la création d'une application mobile de type Click and Collect pour la cafétéria de l'IUT Nancy-Charlemagne, le Charlemiam.

L'application met à disposition de l'utilisateur tous les produits de la cafétéria. Ils pourront ensuite valider leurs commandes et les envoyer dans le but de venir les chercher à l'heure prévue.

1.3 Présentation de l'équipe

Notre équipe est composée de 4 membres.

- Guillaume ARNOUX, en charge de la création de l'API et du serveur Backoffice
- Maxime BRASLEY, en charge de la création d'un panier local permettant l'envoi de commande
- Noé STEINER, chargé de la réalisation et de la conception de l'interface de l'application
- Steven VIGNERON, chargé de la création et de la mise en place de la base de données

1.4 Déroulement du projet

Le projet a été effectué sous la méthode SCRUM, en trois itérations.

Pendant la première itération, nous nous sommes intéressés aux Frameworks et langages que nous allions utiliser pour la réalisation du projet, et nous avons retenu :

- Flutter et Dart pour développer l'application mobile
- Le service Firestore de Google pour notre base de données
- Une API en JavaScript pour le backoffice

Puis, lors de la deuxième itération, nous avons réalisé une première version de la base de données et de l'application comportant les principaux panels de navigation :

- L'écran des produits qui les affiche par catégories et sur une grille. (L'utilisateur peut ensuite les ajouter à son panier)
- L'écran du panier ou l'utilisateur peut envoyer la commande qu'il a créée s'il est connecté et a les fonds nécessaires sur son compte
- L'écran de connexion/profil qui permet la création et l'authentification d'un utilisateur ainsi que l'ajout de fonds

Pendant la troisième itération, nous nous sommes concentrés sur la finalisation de fonctionnalités majeures comme la validation de commande et de leur gestion sur une interface dédiée au personnel, permettant aussi la gestion des stocks et l'ajout de nouveaux produits.

Par la suite, nous avons ajouté des fonctionnalités mineures mais nécessaires au bon fonctionnement de l'application comme :

- Le déclenchement de notifications lors de la modification du statut d'une commande
- La gestion des débordements liés aux widgets composant l'interface
- La réinitialisation du mot de passe et la suppression d'un compte

Nous avons conclu avec des mises à jour esthétiques et mineures pour rendre l'interface plus intuitive pour les utilisateurs, notamment grâce :

- au grisage des produits indisponibles en cafétéria
- au thème clair de l'application

2. Analyse

Dans cette partie du rapport, nous nous intéresserons à l'analyse du projet. Nous allons, dans un premier temps étudier le découpage fonctionnel du projet avec les différentes parties qui le composent puis, nous étudierons certains modèles UML et enfin, nous contrasterons les évolutions entre notre étude préalable et le présent document.

2.1 Découpage fonctionnel

Nous allons à présent nous concentrer sur le découpage fonctionnel de notre projet. Celui-ci fonctionne grâce à trois parties qui sont interdépendantes.

2.1.1 Application mobile

Tout d'abord, le système se compose d'une application mobile. Elle constitue la base du projet. Accessible pour tous les utilisateurs, cette dernière permet aux clients de transmettre leurs commandes au Charlemiam en choisissant les produits qu'ils désirent, la quantité souhaitée, mais aussi les instructions de retrait relatives à la commande.

L'application, point central du système, permet à un utilisateur d'interagir avec l'ensemble des ressources mises à sa disposition.

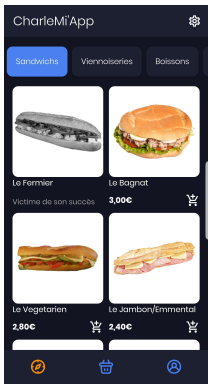

FIG. 2.1 – Écran principal de l'application

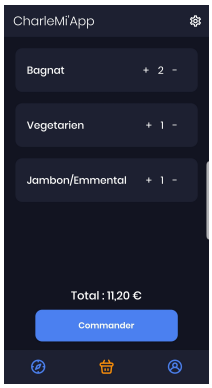

FIG. 2.2 – Écran relatif au panier

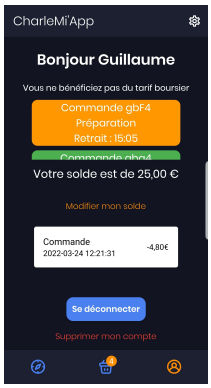

FIG. 2.3 – Écran relatif à l'utilisateur

Pour pouvoir créer une commande, le client doit être connecté (donc authentifié) à son compte et doit avoir un solde lui permettant de valider sa commande.

Ainsi, l'application mobile est conçue pour fonctionner avec les autres services et ne peut pas échanger sans une garantie de fonctionnement de l'ensemble du système.

2.1.2 Serveur Backoffice

Ce système est, comme son nom l'indique, la plateforme de gestion réservée au personnel du Charlemiam. Comme le client mobile, il permet d'interagir avec le serveur. Ce dernier est une application web nécessitant une authentification pour y accéder. Une fois connectés, les acteurs ont accès à des opérations de gestion, prioritaires par rapport au client mobile.

Les différentes opérations proposées sont :

- Gestion des horaires d'ouverture
- Gestion des stocks
- Gestion des commandes
- Gestion du statut boursier d'un utilisateur

Autrement dit, le personnel reçoit les différentes commandes et peut modifier leur état au cours du temps. Ils ont aussi la possibilité de modifier, sur leur espace réservé, les stocks des différents produits afin de toujours avoir le contrôle sur le stock mis à disposition des utilisateurs. L'ensemble des interactions effectuées par le personnel met à jour les éléments correspondant dans la base de données afin que les données affichées sur l'application mobile soient à jour en permanence.

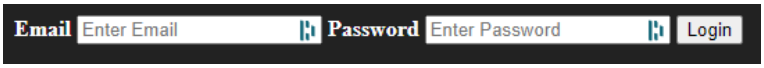

FIG. 2.4 – Authentification sur la plateforme

FIG. 2.5 – Gestion des commandes

FIG. 2.6 – Gestion des stocks

2.1.3 Interface de programmation (A.P.I.)

Afin que les clients (systèmes) puissent communiquer avec le serveur, il a fallu créer une API qui va avoir pour rôle de récupérer les informations de la base de données et de les retransmettre dans l'application. L'API est le noyau central de la transmission d'informations entre le client et le serveur.


```
{
    "success": true,
    "list": [
        {
            "calories": null,
            "boursier": false,
            "description": null,
            "price": 0.8,
            "diminutif": "Pain chocolat",
            "category": "Viennoiseries",
            "name": "Pain au chocolat",
            "stock": 13,
            "id": "P0001"
        },
        {
            "stock": 0,
            "calories": null,
            "price": 0.7,
            "id": "P0002",
            "category": "Viennoiseries",
            "description": null,
            "boursier": false,
            "name": "Croissant",
            "diminutif": "Croissant"
        },
        {
```


FIG. 2.7 – Réponse type provenant de l'API

Toutes les requêtes provenant du client mobile transitent vers l'API. Son rôle est de traiter et calculer un résultat grâce aux données fournies à la place de l'appareil mobile. Elle permet aussi de déléguer certaines tâches et d'effectuer des opérations à la place du serveur.

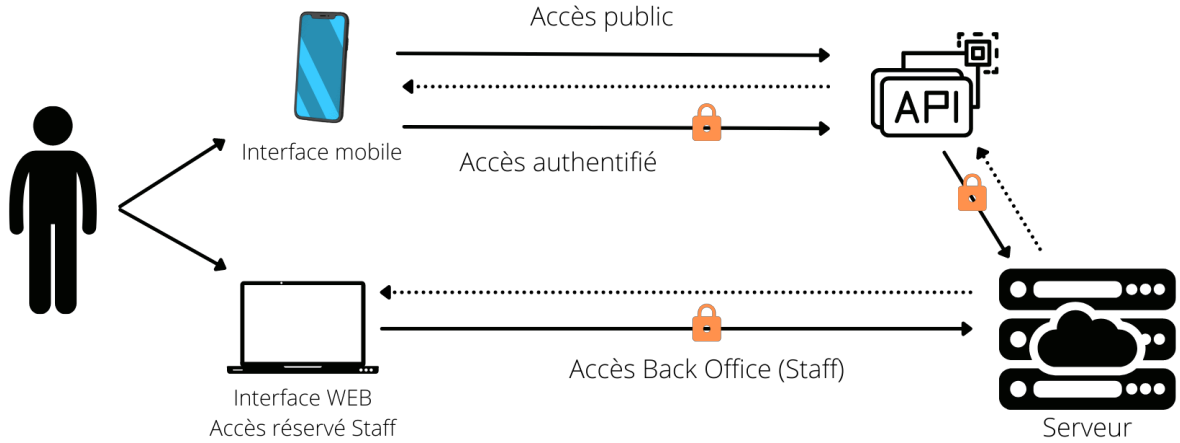

FIG. 2.8 – Schéma du fonctionnement global des systèmes

2.1.4 Serveur

Pour que tous les systèmes puissent communiquer entre eux, il a fallu concevoir une plateforme centrale capable de faire dialoguer des systèmes conçus différemment. Cette plateforme, nous l'avons mise en place grâce à Google.

Cette plateforme se compose de 4 éléments principaux :

- Serveur d'authentification
- Base de données
- Serveur de messagerie
- Serveur exécutant l'API

2.1.4.1 Authentification

L'authentification est rendue possible grâce au service Firebase Authentification. Le choix d'utiliser un service provenant de Google assure une réelle sécurité des données, ainsi qu'une rapidité d'exécution très appréciée.

Aussi, cela délègue une partie des actions afin que le personnel de la cafétéria n'ait pas de contraintes techniques à gérer.

2.1.4.2 Base de données

La base de données fonctionne grâce au service Cloud FireStore, fourni par Google. Les données sont enregistrées de manière dynamique (NoSQL), permettant une rapidité accrue et une simplicité optimale.

FIG. 2.9 – Représentation d'un utilisateur

FIG. 2.10 – Représentation d'une commande

FIG. 2.11 – Représentation d'un produit

2.1.4.3 Serveur de Messagerie

Cette partie du système a pour objectif de dialoguer avec les appareils mobiles des clients par le biais de notifications. Le service, fourni par Google, se nomme Firebase Messaging.

Il permet, par le biais d'un token FCM, d'envoyer des notifications sur un appareil distant.

La principale utilité est d'informer un utilisateur lorsque sa commande est prête ou annulée.

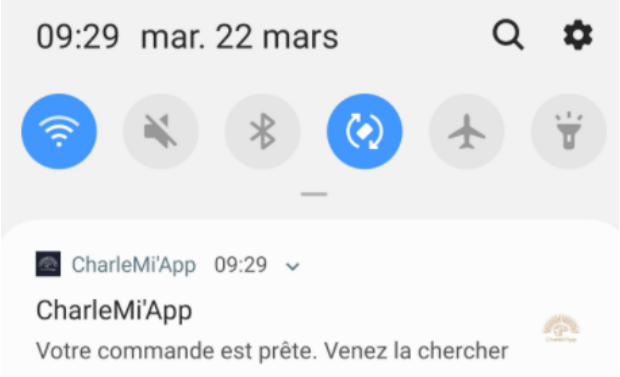

Fig. 2.12 – Affichage d'une notification sur un appareil distant

2.1.4.4 Serveur exécutant l'API

Également proposé par Google, Cloud Functions est une solution FaaS. Un serveur est mis à notre disposition afin d'exécuter des scripts, notre API dans le cas présent.

La particularité de ce service est sa modularité. Le service est dimensionné sur mesure, c'est-à-dire que si les requêtes provenant sur les serveurs augmentaient de manière considérable, le serveur se redimensionnerait tout seul afin de répondre à notre besoin et ce sans aucune opération requise de notre part.

L'intérêt d'utiliser Google pour la plupart des services est multiple. Il apporte notamment une vitesse d'exécution très rapide, une proximité des services et une fiabilité bien plus haute que si nous avions développé nos propres solutions en entier.

2.2 Modélisation et conception

Nous allons maintenant étudier nos systèmes grâce aux modèles UML ayant permis leur construction.

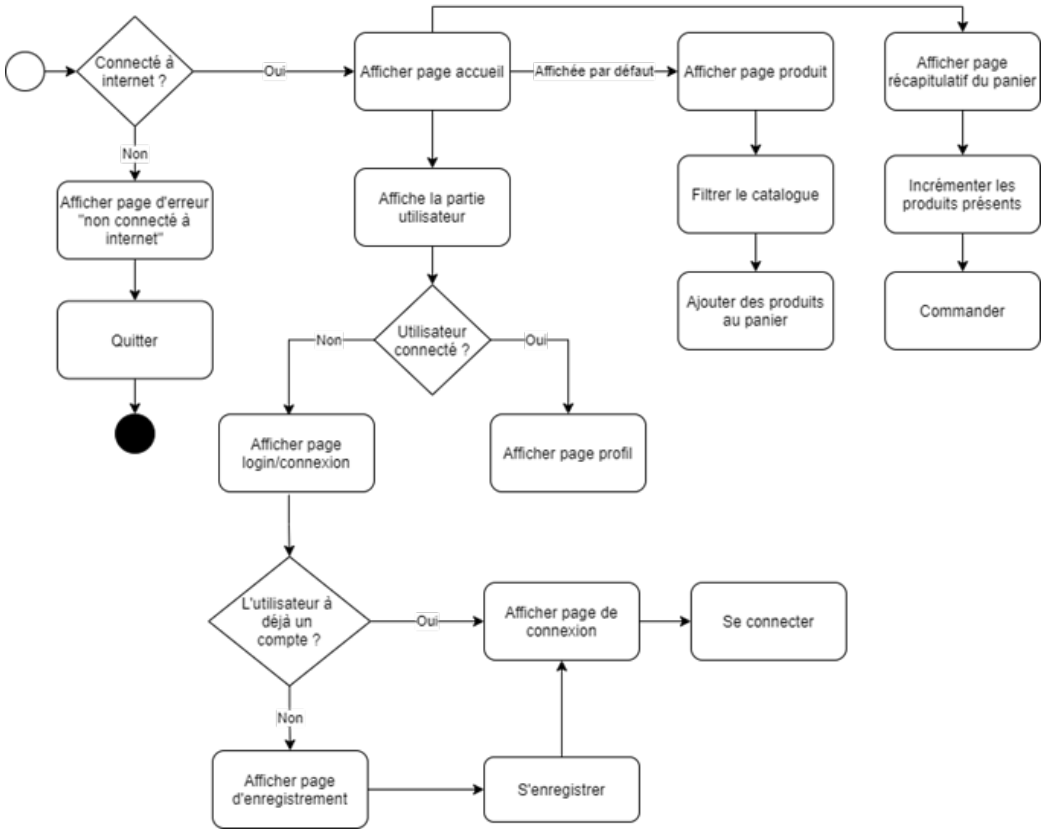

FIG. 2.13 – Diagramme d'activité concernant le rendu de l'application

Ce diagramme d'activité illustre le fonctionnement de l'affichage de l'application mobile en fonction des différentes possibilités. Il permet de comprendre le fonctionnement global de l'application.

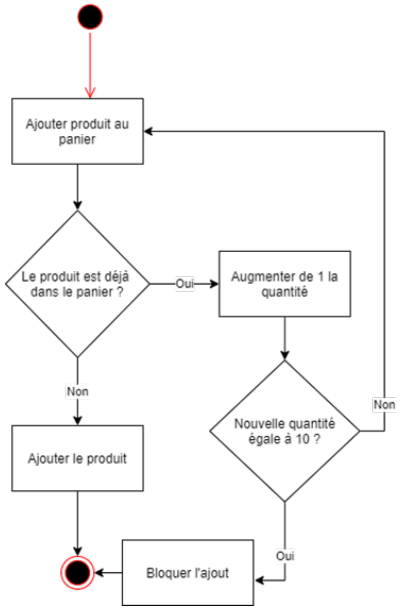

FIG. 2.14 – Diagramme d'activité concernant le fonctionnement du panier local

Ce diagramme d'activité permet d'illustrer le fonctionnement de l'ajout d'un produit au panier. On constate par exemple qu'un produit ne peut pas être ajouté plus de 10 fois pour éviter les abus.

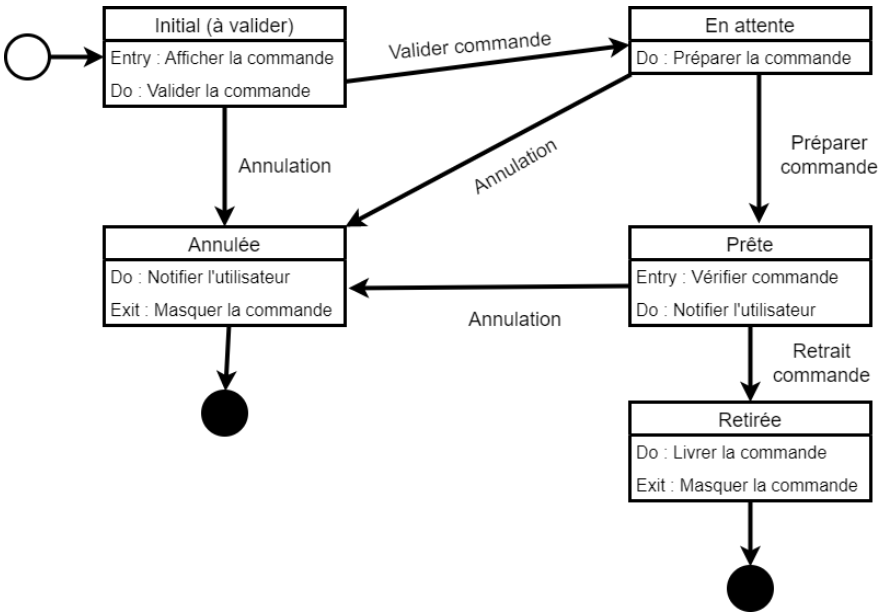

Fig. 2.15 – Diagramme d'état relatif au changement de statut d'une commande

Ce diagramme d'état détaille les différentes transitions entre les états d'une commande. On peut donc connaître tous les états d'une commande, les motifs de passage d'un état à un autre et les actions encourues.

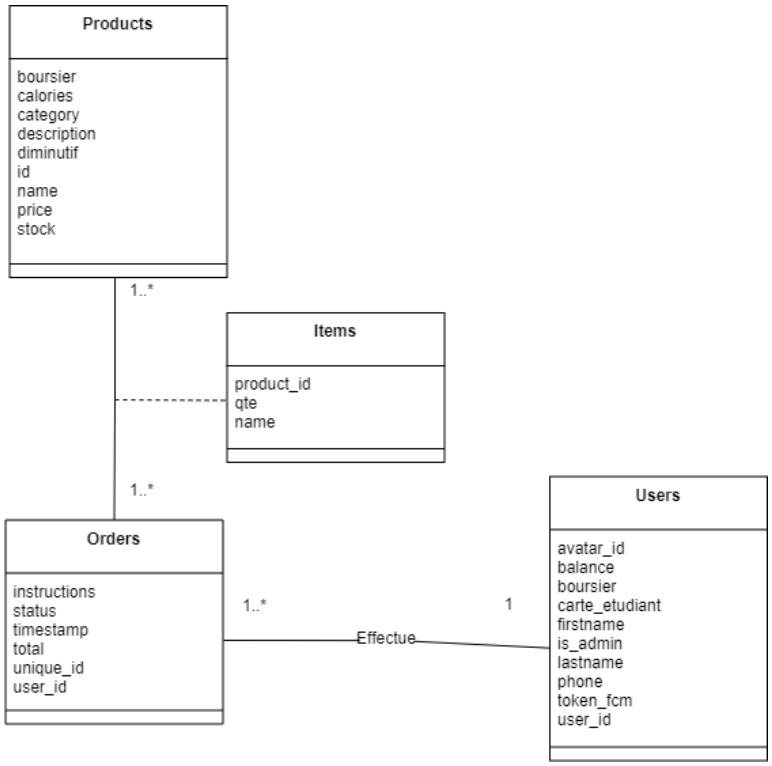

Fig. 2.16 – Modèle relationnel des données

Ce modèle ULM correspond au schéma relationnel de notre application. Il vient compléter l'analyse fonctionnelle faite plus haut.

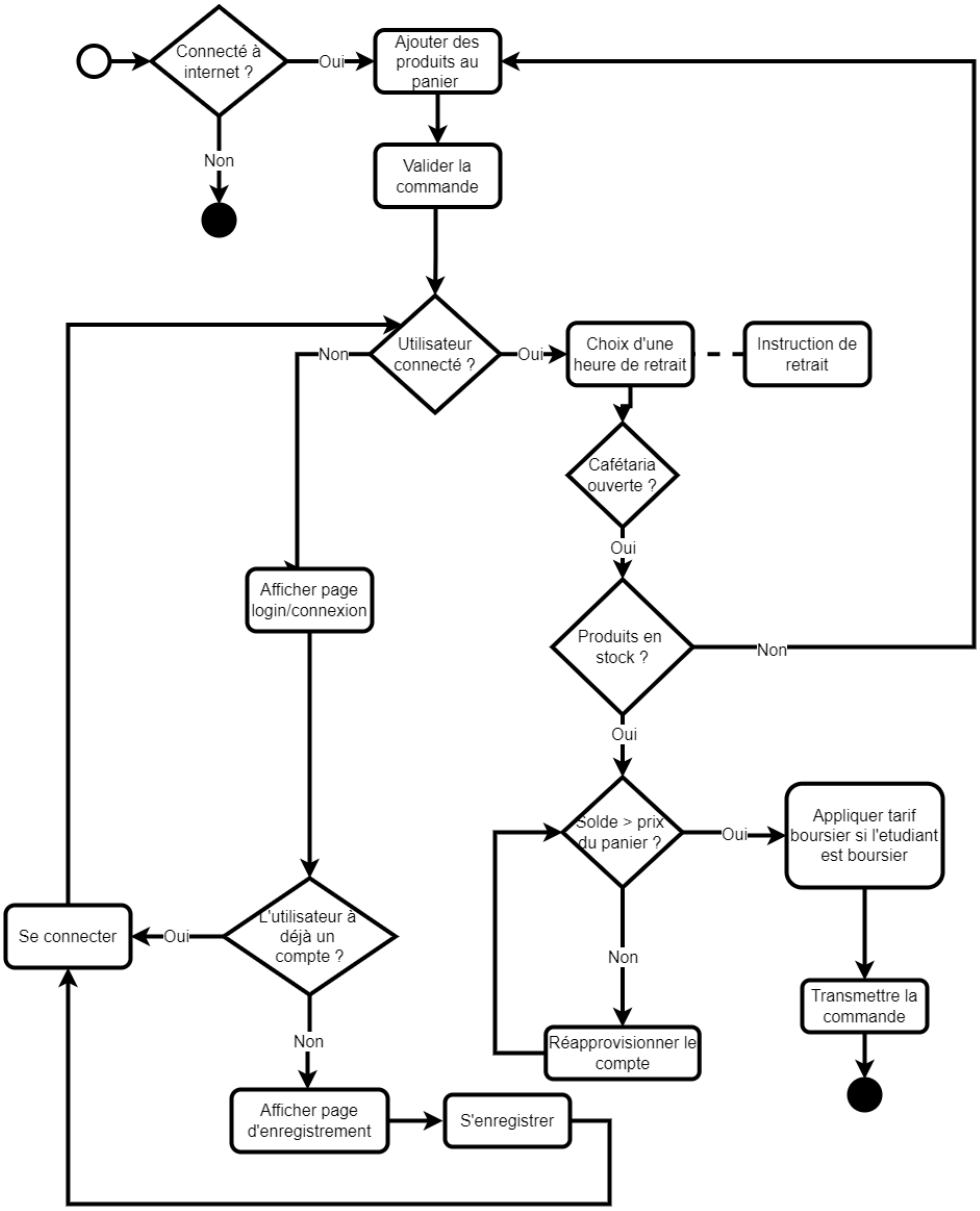

FIG. 2.17 – Diagramme d'activité concernant le passage d'une commande

Ici, nous pouvons étudier le diagramme d'activité correspondant au passage d'une commande par un utilisateur depuis l'application mobile. On constate les différentes vérifications effectuées avant de transmettre la commande.

2.3 Évolution du projet

À présent, nous allons évoquer les évolutions entre l'étude préalable et le projet aujourd'hui.

Nous souhaitions, au début du projet, mettre en place un système d'ouverture du Click and Collect en fonction de créneaux horaires fixes. Aujourd'hui, nous avons un fonctionnement similaire, mais l'ouverture du Click and Collect se fait lorsque le Charlemiam le souhaite grâce à un bouton dans le back-office.

L'ensemble des fonctionnalités que nous souhaitions mettre ont été implantées, que ce soit du côté staff ou du côté client. Cependant, nous aurions voulu avoir le moyen de paiement fourni par Izly afin de pouvoir rendre l'application réellement opérationnelle, toutefois cela n'a pas été possible à la suite du refus du Crous.

Les technologies envisagées lors de l'étude préalable sont restées inchangées.

3. Réalisation

Maintenant, nous allons étudier la réalisation de ce projet.

Pour ce faire, nous présenterons dans un premier temps les outils et les technologies que nous avons utilisés. Puis, l'architecture au niveau de la programmation de nos applications. Dans une troisième partie, nous étudierons les nombreuses difficultés que nous avons rencontrées tout au long des itérations, ainsi que la manière dont nous les avons surmontées. Enfin, nous parlerons des moyens de validation de nos fonctionnalités.

3.1 Outils et technologies utilisés

Tout au long de ce projet, nous avons été amenés à utiliser diverses technologies et divers outils afin de faciliter la conception, l'organisation, la communication et le développement de notre projet tutoré.

Afin de conceptualiser celui-ci, nous avons utilisé plusieurs logiciels, le principal étant l'application web Draw.io, qui nous a permis de réaliser les différents diagrammes et modèles UML. De plus, pour la partie UX et UI de notre application mobile, nous avons eu recours aux logiciels de conception Figma et Adobe Experience Design. Ceux-ci ont mené à la réalisation des maquettes graphiques de notre application, afin d'avoir un plan clair de développement front-end de celle-ci.

Nous avons utilisé les applications web Notion, Trello ainsi que Discord, afin de nous organiser et de communiquer entre nous. Trello a été le point central de toutes nos opérations. C'est grâce à cet outil que nous avons défini les tâches de chacun. Notre page Trello regroupe l'intégralité de toutes les activités relatives à notre projet, représentées sous forme de cartes interactives. Notion, quant à lui, nous a permis d'étendre ce que nous proposait Trello, c'est-à-dire que chacun a pu se construire un carnet de notes, un calendrier et certains documents. Enfin, nous communiquons par le biais de Discord, via son système de messagerie instantanée, mais également avec ses salons vocaux où nous pouvons partager nos écrans en temps réel afin que nos collègues puissent nous aider ou simplement nous expliquer quelque chose à propos du projet.

Au niveau du développement, nous avons utilisé plusieurs IDE, outils de développement et langages. En premier lieu, nous avons eu recours à Visual Studio Code comme éditeur de texte avec les extensions nécessaires pour traiter tous les langages présents dans notre projet. Autre logiciel retenu : IntelliJ Idea Ultimate pour la programmation avec Dart et le Framework Flutter, utilisé dans l'application mobile, accompagné d'émulateurs Android et iOS pour effectuer des rendus de l'application. Pour la base de données, nous avons fait le choix de nous appuyer sur le service de Google Firestore. Celui-ci propose une solution rapide et sécurisée pour stocker nos données.

L'API, passerelle de communication entre les services, a été réalisée en JavaScript grâce à NodeJS. Nous nous sommes servis de librairies telles qu'Express.js. Nous avons fait le choix d'utiliser l'IDE Webstorm afin de la créer. Pour l'hébergement de celle-ci, le choix s'est porté vers Google, qui héberge en même temps notre base de données et notre API. Nous avons également opté pour PHP, CSS et JavaScript pour la réalisation du serveur backoffice et il nous a paru intuitif d'utiliser l'IDE PHPStorm, étant donné que tous nos autres IDE sont édités par JetBrains. Enfin, nous avons eu recours à PowerShell et Zsh afin de contrôler les différentes actions, telles que le lancement de l'API.

Tout notre projet a été versionné avec Git, qui nous a permis d'effectuer des sauvegardes, de séparer et de regrouper notre travail.

Afin de concevoir un système viable et sécurisé, nous avons longuement réfléchi, au cours de la première itération, à la manière dont nous allions concevoir et structurer notre projet.

3.1.1 Client mobile

Pour le client mobile, nous avons fait le choix d'utiliser Flutter, framework du langage Dart, créé par Google. Flutter permet la réalisation d'interfaces graphiques natives et modernes. Il se caractérise par la création d'un arbre d'éléments appelés Widgets. Un arbre est également un widget. Il s'agit donc d'une conception de l'interface de type décorateur, un widget pouvant contenir un ou plusieurs widgets.

Pour la structure globale de l'application, nous avons fait le choix d'utiliser le patron de conception BLoC, qui signifie Business Logic Components. Ceci permet d'avoir une version évoluée de MVVM (Modèle vue vue modèle) ou MVC (Modèle vue contrôleur) pour une application type Flutter.

Le principe de ce pattern est que tout, dans l'application, doit être représenté comme un flux d'événements. Le widget reproduit des évènements : interaction avec l'interface graphique, évènement système, changement d'orientation, etc. Le BLoC se trouve derrière le widget et il a la charge d'écouter et de réagir aux évènements produits par le widget. L'objectif de BLoC est en quelque sorte de gérer le cycle de vie d'un widget. Cette structure permet donc d'avoir une application mobile bien conçue, fiable et viable dans le temps pour le plus grand confort des utilisateurs.

3.1.2 API

Pour notre API web, nous avons opté pour une application NodeJS (JavaScript) accompagnée de la librairie Express.js, qui permet notamment d'intercepter des requêtes HTTP et de pouvoir les traiter à la volée. Nous avons ainsi fait le choix de réaliser une API de type RESTful.

L'architecture REST, extrêmement flexible, permet de gérer différents types d'appel et de renvoyer différents types de données (contrairement au type SOAP qui est limité à l'XML).

Comme dit ci-dessus, notre API respecte 6 contraintes, la rendant RESTful.

Voici la liste des contraintes que nous respectons :

- Client-serveur : cette contrainte repose sur le concept selon lequel le client et le serveur doivent être séparés l'un de l'autre et autorisés à évoluer individuellement.
- Sans état : les API REST sont sans état (stateless), ce qui signifie que les appels peuvent être réalisés indépendamment les uns des autres et que chaque appel contient toutes les données nécessaires pour être mené à bien.
- Cache : étant donné qu'une API sans état peut augmenter la surcharge des requêtes en traitant de grandes quantités d'appels entrants et sortants, une API REST doit être conçue de manière à encourager le stockage des données pouvant être mises en cache.
- Interface uniforme : la clé du découplage entre le client et le serveur réside dans une interface uniforme qui permet une évolution indépendante de l'application sans que les services, les modèles ou les actions de l'application soient étroitement liés à la couche d'API.
- Système multicouches : les différentes couches de l'architecture des API REST travaillent ensemble pour construire une hiérarchie qui favorise la création d'une application plus évolutive et modulaire.
- Code à la demande : le code à la demande permet de transmettre du code ou des applets via l'API pour les utiliser dans l'application. Ainsi, notre application mobile émet des requêtes HTTP interceptées par notre API RESTful, qui sont ensuite traitées.

3.1.3 Serveur Backoffice

Enfin, s'agissant de l'interface dédiée au personnel de la cafétéria, notre choix s'est tourné vers PHP accompagné du micro-framework Slim.

L'architecture globale du back office est de type MVC (Modèle Vue Contrôleur). Le traitement des données avec Firestore se fait de manière rapide et sécurisée grâce aux librairies de Google prévues à cet effet.

Slim, quant à lui, nous permet de traiter les différentes routes de l'application. C'est ce système qui va traiter les requêtes HTTP adressées à l'application afin de pouvoir lancer les méthodes de nos contrôleurs en fonction des routes interceptées et éventuellement de leur passer des paramètres récupérés à l'aide des arguments. Il se charge ensuite de renvoyer au client le texte HTML correspondant à la vue que le contrôleur aura appelée. Cela permet de construire facilement des interfaces dynamiques.

3.2 Contraintes

À présent, nous allons décrire les différentes contraintes que nous avons rencontrées tout au long des itérations et comment nous sommes arrivés à les résoudre.

3.2.1 Paiement

La première contrainte que nous avons rencontrée est intervenue lors de la conception de notre système : le lien avec le système de paiement du Crous, Izly.

Notre application, pour être fonctionnelle, devait être pourvue d'un système de paiement. Nous avons alors entrepris de multiples démarches auprès du Crous afin d'avoir accès à leurs services. Malheureusement, nos démarches se sont toutes révélées vaines.

Une contrainte majeure est à ce moment entrée en scène : quelle technologie de paiement utiliser ?

Ne pouvant utiliser Izly, nous nous sommes documentés sur les différentes options à notre disposition. Cependant, toutes les solutions potentielles trouvées posaient un souci légal ou d'autorisations.

Nous avons par conséquent fait le choix, en accord avec nos tuteurs, de mettre en place un système de paiement fictif, qui pourra éventuellement être remplacé dans le cadre d'un futur projet tutoré ou d'une adoption de notre projet.

3.2.2 Accessibilité des services

Cette difficulté résolue, il fallait désormais savoir comment implanter notre système dans l'environnement existant.

Nous avons posé de nombreuses questions à nos tuteurs, surtout à Mme Clément, afin de savoir quelles étaient les solutions qui lui conviendraient le mieux.

Pour les utilisateurs, nous avons fait le choix d'une application mobile disponible pour les appareils avec les systèmes d'exploitation iOS et Android, qui permet de couvrir un très grand nombre d'appareils et garantit une compatibilité maximale.

3.2.3 Langages et technologies utilisés

Après avoir terminé la conception et la réflexion autour de la réalisation de notre système, une autre contrainte est apparue : l'apprentissage de nouveaux langages, frameworks et architectures qui nous étaient encore inconnus.

Nous avons fait en sorte d'apprendre le plus efficacement possible le framework Flutter (associé au langage Dart), solution qui nous était à présent inconnue. Nous nous sommes familiarisés avec la syntaxe de ce langage, les différents patrons que nous pouvions utiliser pour bien architecturer notre application et enfin avec les bonnes pratiques à adopter.

Ensuite, nous avons ouvert des projets open source existants dans le but de comprendre la logique à adopter, afin de pouvoir l'appliquer sur notre application. Nous avons alors suivi la même démarche pour l'apprentissage de NodeJS et des librairies associées.

3.2.3.1 Contraintes liées au framework Flutter

Lors de la programmation de l'application mobile, 3 contraintes se sont dessinées.

La première était de savoir comment produire un affichage dynamique des données dans le but d'avoir un contrôle total sur ces dernières.

Pour parvenir à la résolution de cette problématique, nous avons élaboré une version plus évoluée de la maquette graphique. Ceci nous a amené à revoir certains éléments-clés, dont nous n'avions pas forcément conscience lors de la première itération. Cette nouvelle maquette a apporté une vision plus précise et complète avec un affichage dynamique, garantissant le contrôle total sur les données.

Cette nouvelle interface créée, la seconde problématique fut de savoir comment gérer l'affichage au sein de l'application afin d'avoir un rendu homogène sur n'importe quel dispositif mobile.

Au début de notre conception, nous n'avons utilisé qu'un seul type d'émulateur. Nous avions donc en permanence la même taille d'écran, le même positionnement des boutons.

Cependant, même si Flutter gère très bien la portabilité sur des appareils avec des aspects différents, il va de soi que certains éléments ne s'affichaient plus du tout, ou pas correctement sur certains appareils. Nous avons par conséquent réadapté certaines stratégies dans la programmation de l'interface, notamment en utilisant uniquement des valeurs calculées et non «hard-codées» (par exemple, mettre directement la taille en pixels au lieu d'utiliser des pourcentages) et en testant les cas de débordement, afin d'adapter l'affichage des éléments.

Grâce à cela, notre application s'affiche correctement sur tous les dispositifs, peu importe leur taille et leur système d'exploitation, aussi bien sur un format mobile que sur un format tablette. La rotation est également gérée.

La dernière contrainte est intervenue lors de la finalisation de notre système :

Comment émettre des notifications relatives aux commandes passées par l'utilisateur ?

Cette fonctionnalité a été très contraignante, étant donné que le système d'émission de notification nous était totalement inconnu, et surtout, nous avions beaucoup de mal à le comprendre. Nous nous sommes immédiatement documentés sur le fonctionnement des notifications sur les systèmes Android et iOS, et également sur le déclenchement de celles-ci avec Google Firebase. Nous avons lu les documentations de Flutter, Firebase, Android et iOS traitant des notifications, accompagnées de tutoriels trouvés sur YouTube dans le but de rendre la compréhension plus simple.

3.3 Validation

Dans l'objectif de valider nos différentes fonctionnalités, nous nous sommes basés sur des critères précis avant de passer à d'autres éléments à créer. Notre premier critère était les validations de nos tuteurs.

Tout au long de notre projet, nous nous sommes très régulièrement réunis avec M. Longhais et Mme Clement. Nous préparions à chaque réunion un résumé de notre état d'avancement ainsi que les contraintes que nous rencontrions. Nous avions ensuite un retour de leur part nous permettant de savoir quoi améliorer, quoi changer et également ce qu'il était possible d'ajouter. Nous avons également dévoilé des aperçus à certains de nos proches afin de connaitre leur ressenti, ces derniers étant extérieurs à ce milieu.

Deuxièmement, à chaque fonctionnalité créée, une phase de tests a été imposée. Ceux-ci visaient à tester la fonctionnalité en local, vérifier la cohérence des données, puis de l'essayer dans des conditions réelles.

C'est grâce à tout ce cheminement et ces stratégies que nous avons pu mener à bien et jusqu'au bout notre projet.

4. Conclusion

Ce qui fait qu'un projet est réussi, c'est d'abord la volonté qu'on met dans celui-ci. Ce projet, nous avons pris plaisir à l'explorer, à le développer, à le faire. En plus de la résolution des problématiques initiales, nous avons vu ce projet comme un véritable défi, car, en tant qu'étudiants de l'IUT, nous sommes en même temps concepteurs et clients de l'application.

Malgré les contraintes ayant parsemé notre projet, nous avons toujours su faire face aux problèmes et trouver d'autres solutions à temps. L'esprit d'équipe a été très important au sein de ce projet, car il a permis de mettre en commun beaucoup de connaissances et d'idées.

Ainsi, nous pensons avoir exploré entièrement le sujet. En tout cas, nous avons fait le maximum.

En ce qui concerne la reprise de ce projet, c'est sans doute une réponse positive des établissements Crous Lorraine qui pourrait le faire sortir de sa bulle "fictive". Ainsi, sans changements dans ce domaine et sans autorisations légales de la part d'autres plateformes, nous ne pensons pas qu'il soit possible de développer ce projet, si ce n'est qu'en faisant de l'ajout de fonctionnalités secondaires.

5. Glossaire

A.P.I. (Application Programming Interface) : Interface de programmation servant de façade par laquelle un logiciel offre des services à d'autres logiciels. Elle fait office de passerelle entre plusieurs services.

Backoffice : Interface à accès restreint regroupant des activités d'administration d'une plateforme. Il s'agit de la partie "invisible" d'un site web ou d'une application.

BLoC, Decorator, MVC, MVVM : Patrons de conception permettant de structurer l'architecture d'une application. Tous ont des rôles différentes.

FaaS (Function-as-a-Service) : Solution évolutive de cloud computing qui fournit une plate-forme permettant aux clients de développer, d'exécuter et de gérer des fonctionnalités d'application sans serveur.

Framework : Infrastructure logicielle regroupant un ensemble cohérent de composants servants à créer les fondations ainsi que les grandes lignes de tout ou partie d'un logiciel. Il s'agit de bibliothèques évoluées.

I.D.E. (Integrated Development Environment) : Environnement de développement composé d'un ensemble d'outils permettant d'augmenter la productivité des programmeurs qui développent des logiciels.

NodeJS : Plateforme logicielle libre en JavaScript, orientée vers les applications réseau évènementielles.

NoSQL : Bases de données non relationnelles. Les données sont dites dynamiques.

REST - RESTful (REpresentational State Transfer) : Une application (A.P.I.) est dite REST si cette dernière respecte des contraintes d'architecture bien définies. REST est un ensemble de contraintes architecturales. Il ne s'agit ni d'un protocole, ni d'une norme. RESTful détermine les services développés avec l'architecture REST.

SCRUM : Cadre de développement évolutif regroupant des techniques de travail. Ces techniques sont liées à un objectif de production. *Ex : Méthodes agiles*

Token FCM : Clé d'identification d'un appareil permettant de dialoguer avec ce dernier par le biais de notifications, gérées par le système d'exploitation et non l'application en question.

U.M.L. (Unified Modeling Language) : Langage de Modélisation Unifié à base de pictogrammes conçu comme une méthode normalisée de visualisation dans les domaines du développement logiciel et en conception orientée objet.