HES-SO MASTER

EthiScan - L'App pour les Consomm'acteurs

Vendredi, 14 Mai 2024

Professeur: Pascal Bruegger & Aïcha Rizzotti

Olivier D'Ancona, Clarisse Fleurimont, Yannis Chamot

Contents

Abstract/Résumé

EthiScan est une application mobile destinée à transformer l'expérience de consommation en permettant aux utilisateurs de scanner des produits pour obtenir des informations détaillées alignées avec leurs valeurs personnelles. Elle vise à promouvoir une consommation responsable en fournissant des données sur les labels environnementaux et nutritionnels, l'évolution des prix, l'impact carbone, et d'autres critères pertinents. En intégrant une technologie de pointe et une conception centrée sur l'utilisateur, EthiScan offre une plateforme fiable et intuitive pour faire des choix de consommation éclairés et responsables.

1 Introduction

1.1 Motivation

Dans un monde où les consommateurs sont de plus en plus conscients des implications éthiques, environnementales et sanitaires de leurs achats, il est crucial de fournir des outils qui permettent de faire des choix éclairés. L'application Ethiscan a été conçue pour répondre à ce besoin croissant. En scannant des produits, les utilisateurs peuvent obtenir des métadonnées détaillées et pertinentes sur ceux-ci. Ces informations permettent de savoir si un produit est local, équitable, biologique, ou répond à d'autres critères spécifiques que l'utilisateur juge importants. Ethiscan offre une solution moderne et pratique pour aligner les choix de consommation avec les valeurs personnelles des individus.

1.2 Concept

L'application Ethiscan se distingue par son approche innovante et personnalisée des métadonnées de produits. Contrairement à des applications existantes comme Yuka [11], qui se concentrent uniquement sur les métadonnées alimentaires, Ethiscan vise à créer une plateforme versatile où les utilisateurs peuvent s'abonner aux types de métadonnées qui leur importent le plus. Des mainteneurs spécialisés testent les produits et certifient les informations, assurant ainsi la fiabilité des données fournies. Cette plateforme unique ambitionne de rassembler une large base d'utilisateurs en leur offrant une solution intégrée et complète pour suivre les produits en fonction de leurs préférences et convictions personnelles. En facilitant l'accès à des informations détaillées et certifiées, Ethiscan se positionne comme un outil indispensable pour les consommateurs modernes et responsables.

1.3 Vue d'ensemble des technologies

Dans le développement de notre application mobile, nous avons adopté plusieurs technologies modernes pour garantir efficacité, performance et maintenabilité. Cette section présente une vue d'ensemble des technologies utilisées, organisées par front-end et back-end, ainsi que les outils de développement et de déploiement employés.

- **Framework principale :** Flutter
- **Versionnement et CI/CD :** GitHub, GitHub Actions
- **Authentification :** Firebase Authentication
- **Base de données :** Firebase Firestore
- **Scan de codes QR :** Librairie Google
- **Architecture :** Clean Architecture, Bloc
- **Automatisation de code :** JSON Serializable, Freezed
- **Localisation :** Localizations de base de Flutter

1.4 Front-end

Pour le développement de l'interface utilisateur, nous avons choisi Flutter comme framework principal. Flutter permet de créer des applications cross-platform, ce qui nous permet de cibler à la fois les utilisateurs Android et iOS avec une seule base de code. Voici quelques-unes des technologies et pratiques clés que nous avons utilisées pour le front-end :

- **Flutter :** Framework pour le développement d'interfaces utilisateur.
- **Bloc :** Utilisé pour la gestion de l'état, assurant une séparation claire des responsabilités et facilitant la testabilité.
- **I18n :** Pour supporter plusieurs langues et offrir une expérience utilisateur adaptée à différents marchés.

- **Google MlKit Barcode Scanning :** Intégrée pour offrir une fonctionnalité de scan rapide et précise.

1.5 Technologies de Stockage

Nous avons considéré différentes technologies pour stocker nos données avant de se tourner vers Firebase. Le tableau 1 récapitule les différentes technologies avec leurs caractéristiques clés :

Technologie	Type	Scalabilité	Expérience	Services Complémentaires	Coûts	Cloud
SQFLite	SQL	Limitée	Faible	Non	Gratuit	Non
Drift	SQL	Limitée	Faible	Non	Gratuit	Non
Firebase Firestore	NoSQL	Haute	Élevée	Authentification, Analytics	Gratuit/Tier	Oui
Supabase	NoSQL	Haute	Moyenne	Authentification, API	Variable	Oui
Flutter Storage	NoSQL	Limitée	Moyenne	Non	Gratuit	Non
Hive	NoSQL	Limitée	Faible	Non	Gratuit	Non
ObjectBox	NoSQL	Haute	Faible	Sync, GraphQL API	Variable	Oui
Shared Preferences	NoSQL	Limitée	Moyenne	Non	Gratuit	Non

Table 1: Comparatif des technologies de stockage pour Flutter

1.5.1 Choix de Technologie

Nous avons choisi Firebase Firestore pour plusieurs raisons clés. En tant que base de données NoSQL, elle élimine l'impedance mismatch, simplifiant ainsi le développement. Firebase est gratuit dans son tier initial, ce qui est idéal pour démarrer sans coûts initiaux. Étant basé dans le cloud, il permet une synchronisation en temps réel des données. De plus, Firebase offre des services intégrés tels que l'authentification et les analytics, réduisant ainsi le besoin de solutions tierces. L'expérience préalable d'un membre de notre équipe avec cette technologie a également été un atout majeur. En somme, Firebase Firestore s'est avéré être la solution idéale, offrant une intégration simplifiée, des coûts initiaux réduits, des services complets, et une accélération du développement sans nécessiter de backend complexe.

1.6 Méthodologie de Développement

1.6.1 Kanban

Les tâches sont organisées en issues avec les états suivants :

- **To Do** : Tâches dont l'implémentation n'est pas encore commencée. Elles peuvent cependant déjà être attribuées.
- **In Progress** : Tâches en cours d'implémentation.
- **To Review** : Une fois l'implémentation terminée, le développeur ouvre une pull request et place la story dans cette colonne. Un autre développeur vérifie l'implémentation, peut laisser des commentaires, et s'assigne la story en tant que reviewer. Si l'implémentation nécessite des modifications, il la déplace dans la colonne "To Do" avec des commentaires explicatifs. Sinon, il ferme la pull request et déplace l'issue dans "Done".
- **Done** : Tâches terminées.

1.6.2 Outils de développement

Pour assurer une gestion efficace du projet et maintenir une haute qualité de code, nous avons utilisé les outils suivants :

- **GitHub :** Plateforme de gestion de code source, facilitant la collaboration entre les membres de l'équipe.
- **GitHub Actions :** Utilisé pour automatiser le processus de compilation à chaque push de code, et pour exécuter des linters afin de maintenir la cohérence et la qualité du code.
- **JSON Serializable et Freezed :** Paquets utilisés pour l'automatisation de la sérialisation des objets JSON, réduisant les erreurs manuelles et accélérant le développement.

1.7 Architecture

Pour garantir la robustesse et la maintenabilité de notre application Flutter, nous avons adopté le principe de la Clean Architecture [7]. Ce paradigme architectural permet une séparation claire des responsabilités, divisant le projet en plusieurs couches distinctes :

- **Domain Layer** : Cette couche contient la logique métier et les entités de l'application. Elle est indépendante des détails d'implémentation et des frameworks, ce qui facilite les tests unitaires et la réutilisation du code.
- **Data Layer** : Responsable de la gestion des sources de données (API, bases de données locales, etc.), cette couche implémente les interfaces définies dans la couche Domain. Elle assure également la conversion des données entre les formats utilisés par les sources externes et les entités du domaine.
- **Repository Layer** : Les repositories servent d'intermédiaires entre la couche Domain et la couche Data. Ils encapsulent la logique de récupération et de persistance des données, offrant une interface simplifiée pour les use cases de l'application.
- **Presentation Layer** : Cette couche gère l'interface utilisateur et les interactions. Elle utilise les données fournies par les use cases pour mettre à jour l'état de l'interface.

En plus de la Clean Architecture, nous avons implémenté le pattern Bloc (Business Logic Component) pour la gestion des états [1]. Le pattern Bloc facilite la séparation entre la logique de présentation et la logique métier, en utilisant des flux de données réactifs (Streams). Chaque Bloc gère un état spécifique de l'application et réagit aux événements émis par l'interface utilisateur pour produire de nouveaux états.
La combinaison de la Clean Architecture et du pattern Bloc nous a permis de concevoir une architecture modulaire, testable et évolutive. Cette approche nous a permis de créer une application à la fois robuste et professionnelle, répondant aux exigences de qualité nécessaires pour un projet de cette envergure.

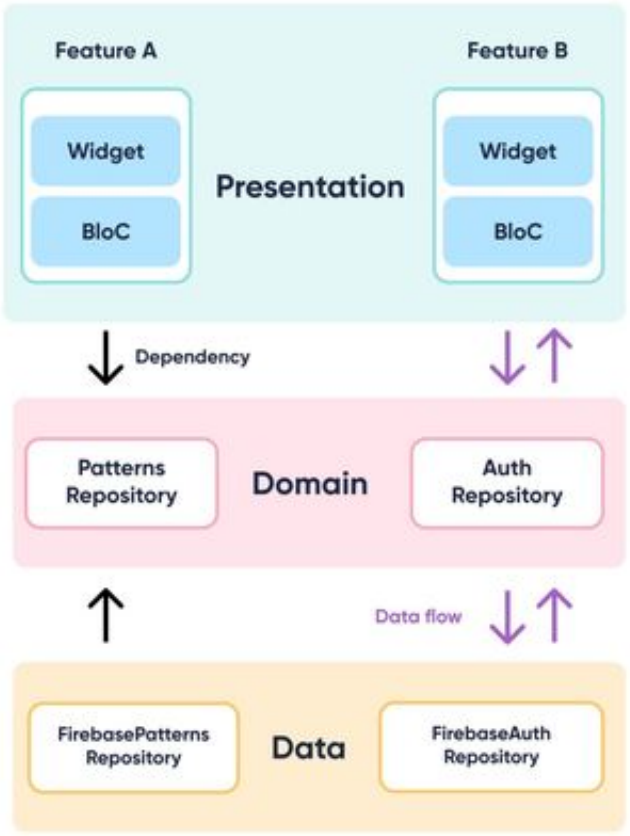

Figure 1: Clean Architecture et BLoC pour la structure de l'application. (image récupérée sur [9])

2 UX

2.1 User Stories

Pour le développement de notre application mobile, nous avons défini plusieurs user stories afin de nous assurer que les besoins des utilisateurs sont pris en compte de manière exhaustive. Voici les user stories que nous avons identifiées :

2.1.1 User Story 1

En tant qu'utilisat.eur.rice, je veux pouvoir scanner un produit pour obtenir des informations détaillées sur celui-ci.

- **Priorité** : Must Have
- **Difficulté** : Moyen
- **Critères d'acceptation** :
  - L'utilisat.eur.rice peut scanner un produit en utilisant la caméra de son téléphone.
  - L'application affiche les informations détaillées du produit scanné (Metadata).
- **Histoire** : Alice et Bob sont dans un supermarché et veulent acheter des produits alimentaires. Ils veulent pouvoir scanner les produits pour obtenir des informations détaillées sur ceux-ci. Par exemple, ils veulent savoir si le produit est bio, local, s'il contient des allergènes, etc.

2.1.2 User Story 2

En tant qu'utilisat.eur.rice, je veux pouvoir ajouter un produit à une liste de favoris pour un accès rapide.

- **Priorité** : Nice to Have
- **Difficulté** : Facile
- **Critères d'acceptation** :
  - L'utilisat.eur.rice peut ajouter un produit à une liste de favoris.
  - L'utilisat.eur.rice peut consulter sa liste de favoris.
- **Histoire** : Alice et Bob veulent pouvoir ajouter des produits à une liste de favoris pour un accès rapide. Par exemple, ils veulent pouvoir ajouter des produits qu'ils achètent régulièrement à leur liste de favoris.

2.1.3 User Story 3

En tant qu'utilisat.eur.rice, je veux pouvoir configurer mes préférences d'achat pour recevoir des informations personnalisées.

- **Priorité** : Must Have
- **Difficulté** : Moyen
- **Critères d'acceptation** :
  - L'utilisat.eur.rice peut configurer ses préférences d'achat (metadata) (local, bio, qualité, prix, impact carbone, durabilité de l'emballage, livrable par la poste).
  - L'application affiche des informations personnalisées en fonction des préférences de l'utilisat.eur.rice.
- **Histoire** : Alice a scanné un produit et est abonnée au metadata Labels. Elle cherche à trouver le label bio sur le produit scanné.

2.1.4 User Story 4

En tant qu'utilisat.eur.rice, je veux pouvoir consulter les labels et certifications des produits scannés.

- **Priorité** : Nice to Have
- **Difficulté** : Moyen
- **Critères d'acceptation** :
  - L'application affiche les labels et certifications des produits scannés.
- **Histoire** : Alice et Bob veulent pouvoir consulter les labels et certifications des produits scannés. Par exemple, ils veulent savoir si le produit est bio, s'il a des labels environnementaux, etc.

2.1.5 User Story 5

En tant qu'utilisat.eur.rice, je veux pouvoir consulter l'évolution du prix des produits scannés (metadata).

- **Priorité** : Nice to Have
- **Difficulté** : Moyen
- **Critères d'acceptation** :
  - L'application affiche l'évolution du prix des produits scannés chez différents fournisseurs.
- **Histoire** : Alice et Bob veulent pouvoir consulter l'évolution du prix des produits scannés. Par exemple, ils veulent savoir si le prix du produit a augmenté ou diminué récemment.

2.1.6 User Story 6

En tant qu'utilisat.eur.rice, je veux pouvoir consulter l'impact carbone des produits scannés (metadata).

- **Priorité** : Nice to Have
- **Difficulté** : Moyen
- **Critères d'acceptation** :
  - L'application affiche l'impact carbone des produits scannés.
- **Histoire** : Alice et Bob veulent pouvoir consulter l'impact carbone des produits scannés. Par exemple, ils veulent savoir si le produit a un impact carbone élevé ou faible.

2.2 Wireframe

Après avoir défini les user stories, nous avons créé un wireframe sur Figma pour visualiser l'interface utilisateur de notre application et assurer une cohérence visuelle. Ce wireframe 4 a permis de structurer et d'organiser les différents éléments de l'application, tels que les écrans de scan de produit, la configuration des préférences d'achat, et les listes de favoris. En adoptant un style uniforme, nous avons assuré une expérience utilisateur harmonieuse et intuitive. Ce prototype a servi de base pour le développement, facilitant la collaboration entre les concepteurs et les développeurs, et garantissant que tous les aspects fonctionnels et esthétiques de l'application sont alignés avec les attentes des utilisateurs.

2.3 Design & Experience

Dans le cadre du développement de notre application mobile, nous avons utilisé des composants customisés tout en respectant un design strict afin d'assurer une expérience utilisateur optimale. Nous avons défini une palette de couleurs cohérente et attrayante, qui reflète l'identité de notre application et facilite la navigation pour les utilisateurs. Cette palette de couleurs a été appliquée uniformément à travers tous les composants de l'application pour maintenir une cohérence visuelle et renforcer la reconnaissance de la marque.
Tous les composants de l'application, tels que les boutons, les formulaires, les cartes de produits, et les menus, ont été conçus de manière customisée pour s'adapter à nos besoins spécifiques. Nous avons veillé à ce que chaque composant soit non seulement esthétiquement plaisant mais aussi intuitif et facile à utiliser. Voici quelques-unes des bonnes pratiques en matière d'UX que nous avons implémentées :

- **Consistance Visuelle** : En utilisant une typographie cohérente et des icônes uniformes, nous avons assuré que l'interface reste claire et organisée, ce qui facilite la navigation pour les utilisateurs.
- **Accessibilité** : Nous avons pris soin d'intégrer des fonctionnalités d'accessibilité telles que des contrastes de couleurs suffisants (pour les daltoniens).
- **Navigation Intuitive** : La structure de navigation a été pensée pour être intuitive, avec un menu en 3 points qui a un certain avantage d'accessibilité.
- **Minimisation de la Charge Cognitive** : En évitant les informations superflues et en présentant les données de manière claire et concise, nous avons réduit la charge cognitive de l'utilisateur, facilitant ainsi la prise de décision rapide et informée.
- **Éléments Tactiles Optimisés** : Les zones tactiles pour les interactions ont été conçues de manière à être suffisamment grandes pour éviter les erreurs de manipulation, et les transitions entre les écrans sont fluides pour offrir une expérience utilisateur agréable, grâce aux push et swap de pages.
- **Simplicité et Clarté** : L'interface a été simplifiée pour que chaque écran ne présente que les informations nécessaires, sans surcharge, afin que les utilisateurs puissent se concentrer sur leur tâche principale sans distractions.
- **Internationalisation** : L'application est disponible en plusieurs langues, ce qui permet aux utilisateurs de choisir la langue qui leur convient le mieux. Le processus est décrit dans la section 4.3
- **Gestion des erreurs** : Les erreurs sont bienveillantes et informatives, elles aident l'utilisateur à comprendre ce qui s'est mal passé et à ne pas s'inquiéter. La section 4.1 détaille comment nous l'avons implémenté

Grâce à l'intégration des bonnes pratiques vu en cours, l'expérience utilisateur est plaisante et agréable.

3 Évaluations

3.1 Tests unitaires

Afin de garantir la fiabilité et la robustesse de notre application mobile, nous avons mis en place une série de tests unitaires. Les tests unitaires sont essentiels pour vérifier que chaque composant de l'application fonctionne correctement de manière isolée.
Notre application utilise Firebase pour la gestion des données et l'authentification des utilisateurs, spécifiquement Firestore pour la base de données et Firebase Authentication pour la gestion des utilisateurs 1.7. Étant donné que Firebase est une plateforme de services cloud, il est nécessaire d'utiliser des outils de mocking pour simuler les interactions avec ces services lors des tests unitaires ce qui a posé quelques difficultés 3.
Pour ce faire, nous avons intégré plusieurs librairies de mocking dans notre suite de tests :

- **Mockito** : Cette librairie nous permet de créer des objets mock et de définir le comportement attendu de ces objets lors des tests. Elle est utilisée pour simuler les interactions avec diverses dépendances de notre application.
- **MockFirebaseFirestore** : Cette librairie est utilisée pour simuler les opérations de la base de données Firestore. Elle permet de créer des collections et des documents fictifs, de simuler les requêtes et de vérifier les interactions avec Firestore sans nécessiter une connexion réelle à la base de données.
- **MockAuthentication** : Cette librairie est utilisée pour simuler les opérations d'authentification de Firebase. Elle permet de tester des scénarios d'inscription, de connexion et de déconnexion des utilisateurs sans interagir avec le service d'authentification réel.

3.2 Tests utilisateurs

Des tests utilisateurs ont été planifié mais n'ont pas pu être mis en oeuvre puisque nous avons préféré développer l'application entièrement avant de la faire tester. Il était prévu de réaliser une série de tests utilisateurs à des proches. Ces tests ont été conçus pour évaluer chaque user story et comprendre comment les utilisateurs interagissent avec notre application dans divers scénarios.

4 Développement Technique

4.1 Gestion des erreurs

Pour gérer les erreurs dans notre application, nous avons utilisé `either`, qui permet de retourner une erreur si une opération n'a pas fonctionné, et de renvoyer les données si elle a réussi. De cette manière, nous pouvons gérer facilement l'erreur au front-end. Ensuite, nous avons créé des classes avec des exceptions et des erreurs spécifiques pour pouvoir les afficher à l'utilisateur. Il est essentiel que ces erreurs soient bienveillantes et compréhensibles pour l'utilisateur, et qu'elles ne soient pas techniques. Ainsi, l'utilisateur peut facilement comprendre ce qui s'est passé et comment rectifier la situation, sans être inquiété par des messages d'erreur trop complexes ou techniques.

4.2 Peuplement de la base de données

Pour peupler la base de données de notre application, nous avons rassemblé des données disponibles librement et publiquement sur Internet, en particulier sur les sites de la Migros, de la Coop et de Denner. Ces sites fournissent des métadonnées des articles, incluant leurs prix et des images associées. Nous avons débuté par la création d'un script en Dart qui intègre les produits, certaines métadonnées, ainsi que quelques types de produits et utilisateurs, afin de tester l'application de manière adéquate. Ce script se connecte à Firestore et envoie les données sur Firebase, ce qui permet de peupler efficacement la base de données. Bien que ce script manuel soit initialement utilisé pour insérer les données nécessaires au test de l'application, nous envisageons à l'avenir de développer un scraper web. Ce scraper pourrait automatiser la récupération des données de tous les produits des sites de la Migros, de la Coop, etc., garantissant ainsi une base de données complète et constamment mise à jour. Grâce à cette automatisation, nous pourrions maintenir une base de données riche et précise, essentielle pour le bon fonctionnement de l'application.

4.3 Internationalisation

Nous avons localisé notre application avec le mécanisme d'internationalisation (i18n) de Flutter. Concrètement, nous avons créé un fichier utilitaire i18n qui nous permet de gérer les différentes traductions de l'application. Chaque langue est représentée par un fichier de traduction spécifique, dans lequel sont stockées les chaînes de caractères traduits. Le processus de traduction commence par l'identification des chaînes de texte à traduire dans l'application. Ensuite, ces chaînes sont externalisées dans les fichiers de traduction, où chaque langue dispose de ses propres traductions. Grâce à ce système, l'application peut afficher automatiquement le texte dans la langue choisie par l'utilisateur, en fonction des paramètres de son appareil. Bie que nous n'ayons pas eu le temps de traduire toutes les chaînes de texte de l'application, le principe de l'internationalisation est entièrement implémenté et fonctionnel. Ainsi, l'ajout de nouvelles traductions peut se faire rapidement et efficacement à mesure que de nouvelles langues sont prises en charge ou que de nouvelles chaînes de texte sont ajoutées à l'application.

4.4 Modélisation des Données

La structure de données d'EthiScan, illustrée dans la figure 2 a été conçue pour permettre une gestion efficace et évolutive des informations produits et des préférences utilisateurs.
La classe principale `EthiscanUser` représente un utilisateur de l'application, incluant ses produits favoris (`FavoriteProduct`), ses préférences (`UserPreferences`), et un Firebase `User` contenant des informations d'authentification comme l'identifiant, le nom et l'email. Chaque `FavoriteProduct` référence un `Product` avec une date d'ajout, permettant à l'utilisateur de suivre ses produits préférés.
La classe `Product` représente les produits scannés par les utilisateurs. Chaque produit peut avoir une certification (`Certification`) et est associé à une liste de métadonnées spécifiques (`ProductMetadata`).

Cette séparation entre le type de métadonnées (`MetadataType`) et les instances spécifiques des métadonnées permet une gestion flexible des différentes informations que les utilisateurs peuvent consulter. Les fournisseurs (`Supplier`) sont modélisés pour contenir une liste de produits vendus (`SoldProduct`), chacun ayant un prix spécifique. Cette approche permet de gérer les variations de prix d'un même produit chez différents fournisseurs.
Enfin, les préférences utilisateur (`UserPreferences`) sont représentées par une liste de types de métadonnées auxquels l'utilisateur est abonné. Ces abonnements permettent à l'utilisateur de recevoir des informations personnalisées et pertinentes en fonction de ses valeurs et de ses préférences de consommation.

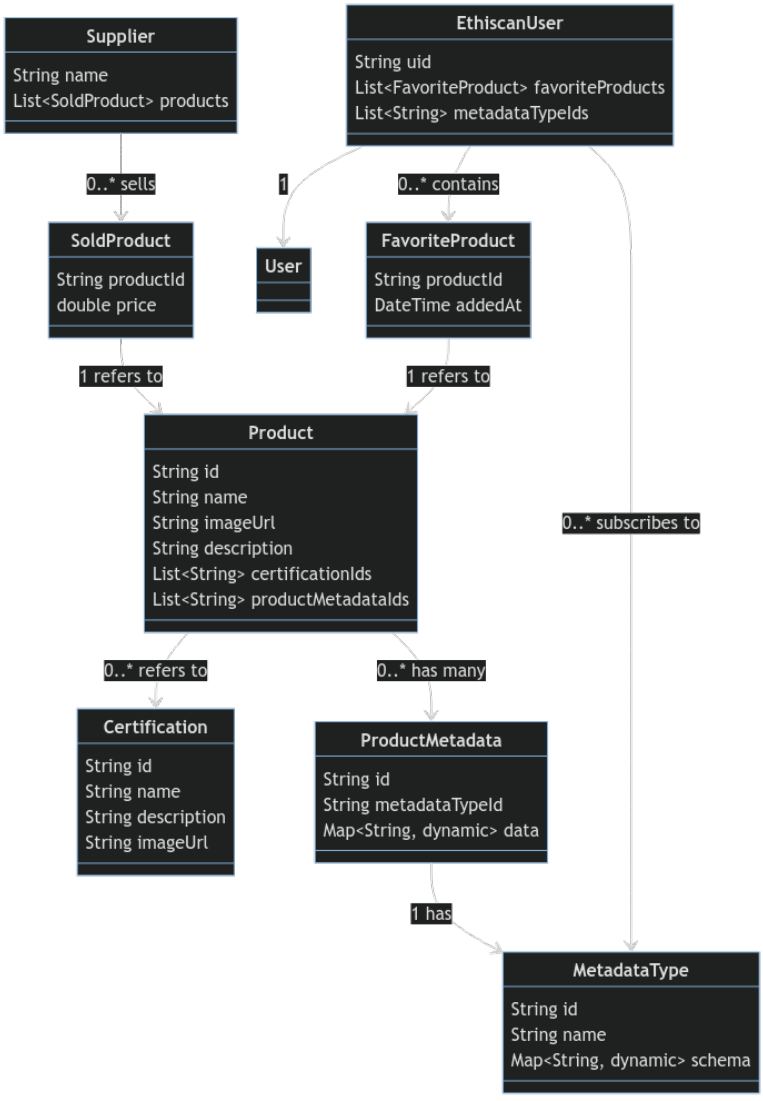

Figure 2: Diagramme de classe de l'application

4.5 Intégration Continue et Déploiement Continu (CI/CD)

Pour automatiser le processus de développement et de déploiement de notre application mobile, nous avons mis en place des GitHub Actions. Ces workflows permettent de s'assurer que chaque modification apportée au code est correctement testée et que les versions de l'application sont déployées de

manière fluide. Nous avons segmenté notre approche CI/CD en deux parties distinctes : l'intégration continue (CI) et le déploiement continu (CD).

4.5.1 Intégration Continue (CI)

L'intégration continue est déclenchée à chaque fois qu'un commit est effectué dans le dépôt GitHub. Le but principal de la CI est de garantir la qualité et la cohérence du codebase. Pour ce faire, nous avons configuré GitHub Actions pour exécuter les tests unitaires et le linter sur l'ensemble du code. En particulier, nous utilisons la commande `flutter analyze` pour analyser notre code Flutter. Cette analyse permet de détecter les erreurs et les incohérences dans le code, facilitant ainsi la collaboration entre les développeurs et assurant que le code est toujours dans un état prêt à être fusionné.

4.5.2 Déploiement Continu (CD)

Le déploiement continu est activé à chaque fois qu'un tag est poussé sur le dépôt. Ce processus comprend la construction de l'application et son déploiement. Nous avons configuré une action pour compiler l'application et générer les fichiers APK nécessaires. Une fois la construction terminée, les fichiers APK sont automatiquement envoyés sur notre groupe Telegram 3. Ceci permet à l'équipe de disposer immédiatement des nouvelles versions de l'application.

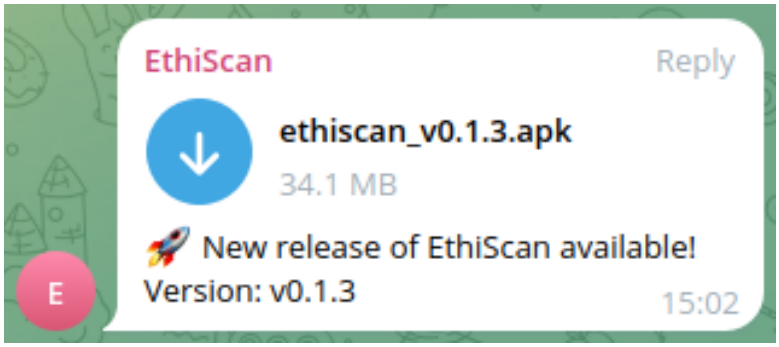

Figure 3: Bot Telegram pour le déploiement de l'application

4.6 Problèmes Rencontrés et Solutions

Version des dépendances Lors du développement, nous avons rencontré un problème majeur lié à la gestion des versions de Flutter utilisées par les différents membres de l'équipe. En effet, nous avons constaté qu'une personne utilisait la version 3.22 de Flutter, tandis que d'autres travaillaient avec la version 3.19. Cette disparité de versions a causé des conflits lors de l'intégration de nouvelles dépendances. Par exemple, certaines dépendances étaient compatibles avec une version de Flutter mais pas avec l'autre, ce qui a entraîné des erreurs de compilation. La synchronisation des versions de Flutter entre tous les membres de l'équipe s'est avérée difficile. Nous avions le choix entre downgrade Flutter ou de résoudre les conflits avec la dernière version. Nous avons opté pour la deuxième option, car elle nous permettait de bénéficier des dernières fonctionnalités et correctifs de Flutter. Cependant, cela a nécessité un effort supplémentaire pour résoudre les conflits et garantir la compatibilité des dépendances avec la version la plus récente de Flutter. Des bugs qui se sont avéré très chronophage ont aussi été rencontré avec nos outils de développement. Un membre de notre équipe a notamment eu des problèmes de dépendances qui n'a pu être résolut qu'en redémarrant VS-Code.

Structure de l'application Nous avons décidé de suivre les bonnes pratiques de la "clean architecture" ce qui a posé plusieurs défis. Comme chaque membre de l'équipe avait une expérience différente Flutter, nous avons tous développé des portions de code de manière indépendante, en utilisant des approches et des structures différentes, souvent inspirées d'exemples trouvés en ligne ou de projets précédents. L'intégration de ces morceaux de code disparates dans un cadre uniforme basé sur la clean architecture a nécessité des efforts significatifs. Nous avons dû refactorer et harmoniser les différentes méthodes de développement pour les aligner avec les principes de la clean architecture, ce qui a parfois causé des problèmes d'intégration et ralenti le développement de nouvelles features. Cette étape a souligné l'importance de définir dès le départ une architecture claire et partagée par toute l'équipe pour faciliter le développement collaboratif. Au final, nous avons utilisé le package "clean_architecture" pour structurer notre application.

Gestion des états Un autre défi a été la gestion des blocs, en particulier avec les pages de connexion et d'inscription. Initialement, nous avions un `mainUserBloc` chargé de l'authentification et de la gestion des données utilisateur dans toute l'application. Cependant, l'intégration des pages de connexion et d'inscription a posé problème. Ces pages, conçues pour fonctionner indépendamment, créaient chacune une nouvelle instance de `mainUserBloc`. Par conséquent, les événements déclenchés sur ces instances n'affectaient pas l'état de l'application principale dans app.dart, entraînant des conflits et des incohérences dans la gestion de l'état. Pour contourner ce problème, notre première solution a été d'intégrer la page de connexion directement dans app.dart, afin qu'elle utilise la même instance de `mainUserBloc`. Bien que cela ait résolu le problème de gestion d'état, cela a rendu le fichier app.dart encombrant et difficile à maintenir. La solution finale a été d'extraire la page de connexion et de lui passer en argument l'instance du `mainUserBloc` utilisée dans app.dart. Cela a permis de déclencher des événements sur la même instance de `mainUserBloc`, assurant ainsi une gestion cohérente de l'état tout en maintenant une structure de code propre et modulaire.

Versionnement du code Nous avons rencontré quelques problèmes avec l'utilisation de GitHub au sein de notre équipe. L'un des membres, ne maîtrisant pas GitHub, a effectué tout son travail sur une branche existante déjà mergée sur la principale (main), sans jamais fusionner ses modifications. Par conséquent, le reste de l'équipe n'a pas pu suivre l'avancement de son travail. Au moment de fusionner sa branche, nous avons découvert le problème. Cela a posé des problèmes d'intégration car le code sur main avait été largement restructuré entre-temps. Pour résoudre cette situation, nous avons résolu les conflits de versions progressivement jusqu'au succès. Puis, nous avons formé toute l'équipe à l'utilisation de GitHub. Désormais, chaque nouvelle tâche doit être accompagnée de la création d'une issue et d'une branche correspondante. Une fois la tâche terminée et approuvée, la branche est fusionnée dans la branche principale avec un code review. Cette approche, combinée à une meilleure communication entre les membres de l'équipe, a permis d'éviter les problèmes.

Déploiement Continu Le déploiement a posé quelques difficultés, principalement liées à la gestion sécurisée du fichier `google-service.json`, indispensable pour la construction et le déploiement de l'application sur notre groupe Telegram. Comme ce fichier contient des informations sensibles, nous devons le sécuriser et le stocker dans un endroit accessible par notre pipeline CI/CD. Pour résoudre ce problème, nous avons décidé de stocker `google-service.json` en tant qu'action secret dans GitHub. Cependant, un problème de formatage est apparu lors de la gestion de ce secret. Pour le surmonter, nous avons encodé le fichier en base64 avant de le stocker dans GitHub Secrets. Ce qui a résolu le problème de formatage et nous a permis de déployer l'application avec succès de manière sécurisée.

Développement Automatisé Pour accélérer le développement, nous avons intégré le package `Freezed` et encore le package `json_encode`, destiné à compiler et générer automatiquement certaines parties du

code. Cette automatisation vise à réduire le code répétitif, augmentant ainsi notre productivité et permettant de se concentrer sur des aspects plus critiques du développement. Cependant, l'introduction de ce package a posé des défis pour un membre de l'équipe non familier avec cette technologie. En effet, il rencontrait des difficultés à déboguer son code, n'ayant pas une maîtrise complète des processus automatisés par `Freezed`. Ce manque de compréhension a entraîné des erreurs de compilation et une perte de temps significative pour cette personne, contrebalançant ainsi les bénéfices attendus de l'automatisation. Pour résoudre ce problème, nous avons organisé une session de discussion afin de clarifier l'utilisation du script `Freezed` et de bien séparer les concepts. Après cette mise au point, il n'a plus rencontré de problèmes de compilation, et l'équipe a pu bénéficier pleinement des gains de productivité promis par l'automatisation. Cette expérience souligne l'importance de l'accompagnement et de la formation lors de l'intégration de nouvelles technologies dans un projet de développement.

Test Unitaires Nous avons rencontré quelques obstacles pour mettre en place les tests unitaires qui se sont avérés complexes à implémenter. Le principal défi a été de simuler Firebase, afin de pouvoir tester nos datasources. Nous avons initialement tenté d'utiliser Mockito, mais cela s'est révélé insuffisant car nous ne pouvions pas simuler les comportements Firebase de manière adéquate. En conséquence, nous avons dû installer un clone de Firebase, un mock Firebase, ainsi qu'un mock Firebase Authentication, ce qui a été un processus laborieux car le package `mock_firestore` empiétait sur une autre dépendance. Finalement, bien que toutes les fonctionnalités aient été développées, elles ont été testées manuellement, ce qui n'est pas conforme aux meilleures pratiques en ingénierie logicielle mais s'est avéré nécessaire pour achever le projet dans les délais.

4.7 Conclusion Technique

La combinaison de Flutter et Firebase a prouvé son efficacité pour développer une application mobile performante et réactive. L'architecture choisie a permis une mise en œuvre rapide des fonctionnalités tout en maintenant une haute qualité et fiabilité des données.

5 Auto-Critique du Code

5.1 Points Positifs

Flexibilité L'implémentation partielle de la clean architecture dans notre application nous offre une flexibilité en termes de gestion des providers de données. Par exemple, si nous décidons de remplacer `Firebase` par `Supabase` par exemple, il suffit de réimplémenter les datasources sans modifier le reste de l'application. Cette modularité nous rend résilients face aux évolutions technologiques et aux changements de fournisseurs de services, assurant ainsi la pérennité de notre application.

Modélisation robuste des données Notre application bénéficie d'une modélisation des données bien structurée et robuste. Cette modélisation permet non seulement de gérer efficacement les données actuelles, mais aussi de faire évoluer le système pour accueillir plusieurs centaines de milliers d'utilisateurs si nécessaire. Grâce à cette architecture scalable, notre application est prête à grandir et à s'adapter aux besoins croissants des utilisateurs, garantissant une performance et une fiabilité optimales à long terme.

Pratiques de développement logiciel Nous avons implémenté de nombreuses pratiques de développement logiciel qui ont grandement facilité notre travail. L'intégration continue (CI) et le déploiement continu (CD) nous permettent d'automatiser les processus de test et de déploiement, assurant une détection rapide des erreurs et des mises à jour fluides de l'application. De plus, en utilisant la clean archi-

tecture, nous avons structuré notre code de manière modulaire et maintenable. Toutes ces pratiques combinées a augmenté notre productivité et nous a aidé à maintenir une haute qualité de code.

5.2 Points à Améliorer

Pour améliorer le développement de notre application mobile, nous avons identifié plusieurs aspects nécessitant des améliorations.

Augmentation de la couverture des tests unitaires Actuellement, la couverture de code par nos tests unitaires est insuffisante. Les tests unitaires sont très limités et ne couvrent pas l'ensemble des fonctionnalités critiques de l'application. Pour garantir une meilleure qualité et fiabilité du code, il est impératif d'augmenter la couverture des tests unitaires. Cela inclut l'ajout de tests pour les cas limites, les scénarios d'erreur, ainsi que les interactions complexes entre les différents composants de l'application.

Renforcement de la sécurité des données Pour des raisons de développement, les règles de sécurité de notre base de données Firestore sont actuellement complètement ouvertes. Cette configuration est inadmissible dans un environnement de production réel. Il est crucial de mettre en place des règles de sécurité strictes afin de gérer correctement les autorisations des utilisateurs. Chaque utilisateur doit uniquement pouvoir accéder et modifier ses propres données. Par exemple, les règles de sécurité doivent garantir qu'un utilisateur ne puisse pas supprimer ou accéder aux produits favoris d'un autre utilisateur. Le renforcement de ces mesures de sécurité est impératif pour la mise en production de l'application.

Amélioration de la clean architecture Actuellement, notre implémentation de la clean architecture est partielle. Les blocs gèrent directement les use cases, ce qui va à l'encontre des principes de séparation des préoccupations et rend les blocs plus difficiles à tester et complexes à maintenir. Pour se rapprocher davantage de la clean architecture, il serait judicieux de séparer les use cases des blocs. Les use cases doivent être implémentés de manière indépendante et injectés dans les blocs, ce qui facilitera les tests unitaires et simplifiera le code.

6 Annexes

6.1 Planning Actualisé Avant/Après

Pour notre projet, nous avons opté pour une organisation méthodique en utilisant un tableau Kanban et une répartition des tâches sur GitHub, plutôt que de suivre un planning formel. Le tableau Kanban nous a permis de structurer le travail en plusieurs colonnes : "À faire", "En cours", "À valider" et "Terminé", offrant ainsi une vue d'ensemble claire de l'état d'avancement du projet et facilitant la coordination. Sur GitHub, chaque tâche était associée à une issue, assignée à des membres spécifiques de l'équipe, ce qui a permis un suivi précis des responsabilités et de l'avancement. Cette méthode de travail nous a permis de prioriser les tâches critiques, de tenir des réunions régulières pour synchroniser nos efforts et de rester flexibles face aux imprévus, assurant ainsi une gestion efficace et adaptative du projet, même sans planning détaillé.

6.2 Liste des Bugs Connus

- Overflow sur le login
- Impossible de lancer la caméra sur Linux ou Chrome
- Si un product n'a pas le même id que son docId, il ne peut pas être supprimé

Dépendances

6.3 Dépendances

6.4 Dépendances de Développement

6.5 Aides Extérieures

- Documentation officielle de Flutter : [6]
- Documentation officielle de Firebase: [5]
- Documentation de FlutterFire pour l'intégration de Firebase: [8]
- Clean Architecture: [7]
- Figma pour la conception de l'interface utilisateur: [4]
- Design Pattern du Bloc: [1]
- Site de la Migros: [10]
- Site de la Coop: [2]
- Site de Denner: [3]
- Architecture : [9]

6.6 Cahier des charges original

Cahier des Charges : EthiScan

1. Introduction

EthiScan est une application mobile conçue pour permettre aux utilisateurs de scanner des produits et de recevoir des informations détaillées alignées avec leurs valeurs personnelles de consommation. Elle vise à promouvoir une consommation responsable en fournissant des données telles que l'évolution du prix, les labels environnementaux et nutritionnels, l'impact carbone, et plus encore.

2. Objectifs du Projet

L'application a pour but de :

- Aider les utilisateurs à faire des choix de consommation éclairés et responsables.
- Fournir des informations détaillées et fiables sur les produits scannés.
- Promouvoir les achats alignés avec les valeurs personnelles des utilisateurs, comme le bio, le local, la qualité, le prix, l'impact carbone, la durabilité de l'emballage, et la possibilité de livraison par la poste.

3. Fonctionnalités Principales

3.1 Scan de Produit [MH]

- Permettre le scan de codes-barres pour identifier rapidement les produits.

3.2 Liste des Produits Favoris [NTH]

- Possibilité d'ajouter des produits à une liste de favoris pour un accès rapide.

3.3 S'abonner aux Metadatas [MH]

- Configuration de préférences d'achat personnalisées : Local, Bio, Qualité, Prix, Impact carbone, Durabilité de l'emballage, Livrable par la poste.

3.4 Sections Détaillées des Métadonnées [MH]

- **Labels** : Affichage des labels et certifications (éco-labels, bio, etc.).
- **Évolution du Prix** : Visualisation de l'évolution du prix chez différents fournisseurs.
- **Impact Carbone** : Information sur l'empreinte carbone du produit.
- **Metadata** : Informations générales (nom du produit, lien vers plus d'infos).

4. Stack Technologique

- **Front-end** : Flutter pour une expérience utilisateur cohérente sur iOS, Android et le Web.
- **Back-end** : Firebase pour l'authentification, le stockage des données, et les fonctions backend.

5. Spécifications Techniques

5.1 Exigences Fonctionnelles

- Authentification sécurisée des utilisateurs.
- Interface intuitive pour le scan de produits et l'affichage des informations.
- Système de favoris et de préférences personnalisables.
- Intégration d'APIs externes pour la récupération des données produits.

5.2 Exigences Non-Fonctionnelles

- Performances : Temps de réponse rapide pour le scan et l'affichage des données.
- Affichage : L'interface utilisateur doit respecter certaines règles (styles, couleurs, ...)
- Accessibilité : Conception inclusive pour une utilisation facile par tous.

6. Deadlines

- Formation groupes et choix du sujet du mini-projet – Semaine 1
- Descriptif du projet (mini cahier de charges) – A remettre avant le cours de la semaine 2
- Validation du projet – semaine 3 – en classe
- Présentations du mini-projet (avec démo) – Semaines 14-15
- Livraison d’un prototype fonctionnel et la rédaction d’un rapport (~15-20 pages). A rendre le lundi avant la dernière séance

7. Conclusion

EthiScan ambitionne de devenir une référence pour les consommateurs souhaitant aligner leurs achats avec leurs valeurs personnelles. Par la transparence et la fourniture d'informations détaillées, l'application vise à promouvoir une consommation plus responsable et éclairée.

Package	Version
flutter	sdk: flutter
injectable	>2.4.0
flutter_localizations	sdk: flutter
cupertino_icons	>1.0.6
flutter_clean_architecture	>5.0.4
firebase_core	>2.25.5
flutter_svg	>2.0.4
syncfusion_flutter_charts	>21.2.4
shared_preferences	>2.2.2
jwt_decoder	>2.0.1
flutter_dotenv	>5.1.0
flutter_launcher_icons	>0.13.1
flutter_i18n	>0.35.1
get_it	>7.1.3
firebase_auth	>4.19.2
cloud_firestore	>4.17.3
provider	>6.1.2
bloc	>8.1.2
flutter_bloc	>8.1.3
path_provider	>2.0.2
auto_size_text	>3.0.0
freezed_annotation	>2.2.0
camera	>0.11.0
google_mlkit_barcode_scanning	>0.12.0
http	>1.2.1
intl	>0.19.0
json_annotation	>4.9.0
uuid	>4.4.0
phone_number_controller	>1.0.4

Table 2: Dépendances du projet

Package	Version
injectable_generator	>2.1.5
build_runner	>2.0.5
json_serializable	>6.1.4
freezed	>2.4.7
flutter_test	sdk: flutter
flutter_lints	>4.0.0

Table 3: Dépendances de développement du projet

6.7 Wireframe

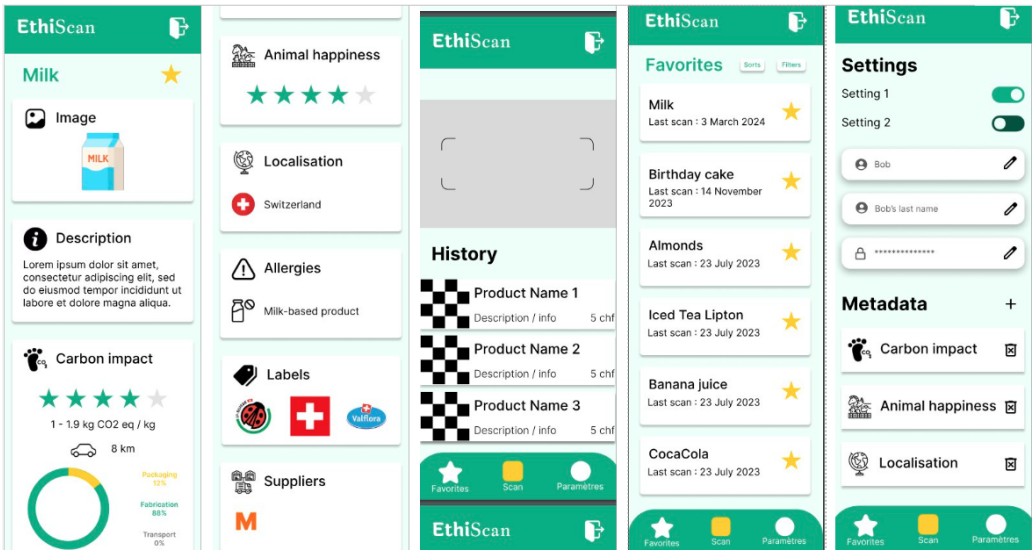

Figure 4: Wireframe de l'application

References

[1] Bloc Library. Design pattern du bloc. https://bloclibrary.dev. Accessed: 2024-05-20.

[2] Coop. Site de la coop. https://www.coop.ch. Accessed: 2024-05-20.

[3] Denner. Site de denner. https://www.denner.ch. Accessed: 2024-05-20.

[4] Figma. Figma pour la conception de l'interface utilisateur. https://www.figma.com/. Accessed: 2024-05-20.

[5] Firebase. Documentation officielle de firebase. https://firebase.google.com/docs. Accessed: 2024-05-20.

[6] Flutter. Documentation officielle de flutter. https://flutter.dev/docs. Accessed: 2024-05-20.

[7] Flutter Clean Architecture. Clean architecture. https://pub.dev/packages/flutter_clean_architecture. Accessed: 2024-05-20.

[8] FlutterFire. Documentation de flutterfire pour l'intégration de firebase. https://firebase.flutter.dev/docs/overview. Accessed: 2024-05-20.

[9] Google Cloud Community. Creating beautiful mobile applications using flutter and firebase. https://www.googlecloudcommunity.com/gc/Community-Blogs/Creating-beautiful-mobile-applications-using-Flutter-and/ba-p/495476. Accessed: 2024-05-20.

[10] Migros. Site de la migros. https://migipedia.migros.ch. Accessed: 2024-05-20.

[11] Yuka. Yuka - food & cosmetic product scanner, 2024. Accessed: 20-May-2024.