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

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.

1.1 Vue d'ensemble des technologies

- **Framework principal :** 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

1.2 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.
- **Localizations :** Pour supporter plusieurs langues et offrir une expérience utilisateur adaptée à différents marchés.
- **Librairie Google pour QR code :** Intégrée pour offrir une fonctionnalité de scan rapide et précise.

1.3 Back-end

Pour gérer les données et l'authentification, nous avons intégré les services de Firebase, connus pour leur fiabilité et leur facilité d'intégration avec les applications Flutter. Voici les principaux services utilisés :

- **Firebase Authentication :** Solution robuste et sécurisée pour la gestion des utilisateurs et leur connexion à l'application.
- **Firebase Firestore :** Base de données NoSQL flexible et évolutive, idéale pour nos besoins dynamiques.

1.4 Outils de développement et de déploiement

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.

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 (Certification = Metadata).

- **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.

2.1.7 User Story 7

En tant qu'utilisat.eur.rice je veux pouvoir avoir accès aux données qui m'intéressent pour pouvoir faire mes choix de produits.

- **Priorité** : Must Have
- **Difficulté** : Moyen
- **Histoire** : Alice va faire des courses pour l'anniversaire de Bob. Elle est devant le rayon des gâteaux. Elle scanne un gâteau qui lui fait envie. Elle veut connaître les informations suivantes : présence ou non de cacahuètes, d'où viennent les composants, s'il existe des produits équivalents qui consomment moins de CO2.

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 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.
- **Feedback Immédiat** : Chaque action de l'utilisateur, comme cliquer sur un bouton ou scanner un produit, est accompagnée d'un feedback visuel et/ou sonore immédiat pour confirmer que l'action a été enregistrée et est en cours de traitement.
- **Accessibilité** : Nous avons pris soin d'intégrer des fonctionnalités d'accessibilité telles que des contrastes de couleurs suffisants (pour les daltoniens), des polices de caractères ajustables.
- **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.

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

3 Évaluations

3.1 Évaluations Implémentées

L'évaluation de l'application s'est concentrée sur la performance, l'exactitude des données, et la satisfaction utilisateur. Des tests de performance ont été réalisés pour s'assurer que l'application répond rapidement aux requêtes de scan. L'exactitude des informations fournies a été vérifiée contre des sources externes pour garantir la fiabilité des données. Enfin, des enquêtes de satisfaction utilisateur ont aidé à recueillir des retours sur l'expérience d'utilisation, permettant d'identifier les domaines d'amélioration.

4 Développement Technique

4.1 Modélisation des Données

La structure de données d'EthiScan, illustrée dans la figure 1 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.

4.2 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).

4.2.1 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.

4.2.2 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. Ceci permet à l'équipe de disposer immédiatement des nouvelles versions de l'application.

4.3 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

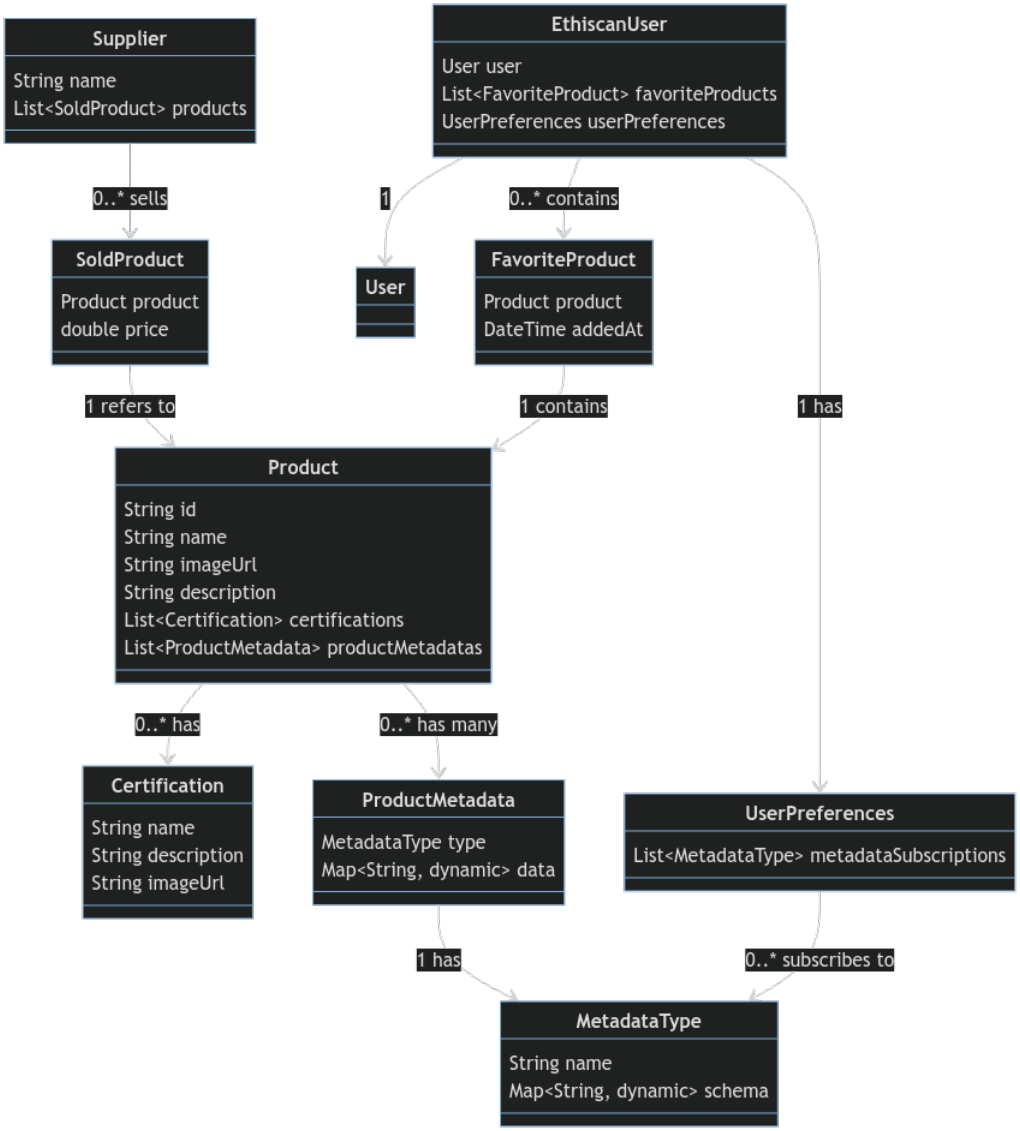

Figure 1: Diagramme de classe de l'application

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.

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, ne s'intégraient pas bien dans la structure de l'application en raison de problèmes liés aux locales de traduction. Pour contourner ce problème, nous avons décidé de placer ces pages directement dans le fichier app.dart, en dehors de la structure principale de l'application. Cette décision a introduit des complications avec le main user bloc, entraînant des conflits entre les blocs de gestion d'état. Bien que nous ayons finalement trouvé une solution. TODO Quelle solution ? Comment on a fix ce truc ?

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.

4.4 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.

4.5 Auto-Critique du Code

4.5.1 Points Positifs

- Code bien structuré et commenté, facilitant la maintenance et les mises à jour.
- Utilisation efficace des patterns de conception pour une architecture solide.

4.5.2 Points à Améliorer

- Couverture des tests unitaires à augmenter pour assurer une meilleure stabilité.
- Optimisation possible de certaines requêtes de données pour accélérer les temps de réponse.

5 Annexes

5.1 Cahier des Charges Original

- **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.
- **Objectifs du Projet**: 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.
- **Fonctionnalités Principales**:
  - **Scan de Produit**: Permettre le scan de codes-barres pour identifier rapidement les produits.
  - **Liste des Produits Favoris**: Possibilité d'ajouter des produits à une liste de favoris pour un accès rapide.
  - **S'abonner aux Metadatas**: Configuration de préférences d'achat personnalisées : Local, Bio, Qualité, Prix, Impact carbone, Durabilité de l'emballage, Livrable par la poste.
  - **Sections Détaillées des Métadonnées**:
    * **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).
- **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.
- **Spécifications Techniques**:
  - **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.
  - **Exigences Non-Fonctionnelles**: Performances: Temps de réponse rapide pour le scan et l'affichage des données. Accessibilité: Conception inclusive pour une utilisation facile par tous.
- **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
- **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.

5.2 Planning Actualisé Avant/Après

- Avant: Formation des groupes et choix du sujet du mini-projet - Semaine 1

- Après: Livraison d'un prototype fonctionnel et la rédaction d'un rapport - Lundi avant la dernière séance

5.3 Liste des Bugs Connus

- Erreur de connexion intermittente au service de scan de code-barres.
- Problèmes d'affichage sur certaines versions d'Android.
- Latence lors du chargement des données produits pour les articles récemment ajoutés.

5.4 Dépendances

- Flutter
- Firebase
- API externe pour les données produits

5.5 Planning

Nous avons désolidarisé la gestion du temps de notre liste des tâches et avons utilisé un Kanban pour suivre notre progression à la place.

5.6 Liste des bugs connus

- bug1
- bug2

5.7 Dépendances

- bug1
- bug2

5.8 Aides Extérieures

- Documentation officielle de Flutter :(`https://flutter.dev/docs`)
- Documentation officielle de Firebase: (`https://firebase.google.com/docs`)
- Documentation de FlutterFire pour l'intégration de Firebase: (`https://firebase.flutter.dev/docs/overview`)
- Clean Architecture: (`https://pub.dev/packages/flutter_clean_architecture`)
- Figma pour la conception de l'interface utilisateur: (`https://www.figma.com/`)
- Design Pattern du Bloc: (`https://bloclibrary.dev`)