# Informations générales

**Noms des coéquipiers** :

**URL du référentiel sur GitHub** : (optionnel si vous travaillez seul)

## Description du prototype

(Copiez la description de votre prototype ici)

## Table des matières

[Semaine #1 : 9 octobre au 15 octobre 2](#_Toc148011688)

[Semaine #2 : 16 octobre au 22 octobre 4](#_Toc148011689)

[Semaine #3 : 23 octobre au 30 octobre 5](#_Toc148011690)

# Semaine #1 : 9 octobre au 15 octobre

## Objectifs

Objectif 1: Introduction à Django : Comprendre les bases de Django et son architecture.

Objectif 2: Installation de Django : Installer Django et configurer l'environnement de développement.

Objectif 3: Url Routing et Django Apps : Apprendre à créer des routes URL et à organiser le projet en différentes applications Django.

Objectif 4: Django Template Language : Se familiariser avec le langage de template de Django pour afficher les données dans les vues.

Objectif 5: Envoi de Données au Fichier Template : Apprendre à passer des données du backend au frontend.

Objectif 6: Comprendre Get vs Post dans Django : Comprendre la différence entre les méthodes GET et POST et comment les utiliser dans Django.

Objectif 7: Fichiers Statiques dans Django : Apprendre à gérer les fichiers statiques dans Django.

Objectif 8: Introduction aux Modèles Django : Apprendre à créer des modèles pour la base de données.

Objectif 9: Panneau d'Administration Django et Manipulation de la Base de Données : Apprendre à utiliser le panneau d'administration de Django pour gérer les données.

## Apprentissages effectués

Quelles ressources (sites web, vidéos, forums, etc.) avez-vous consultées au cours de cette semaine pour apprendre à utiliser les technologies choisies?  
Résumez en quelques points ce que vous avez appris (les éléments clés), puis assignez un score entre 1 et 5 à chaque ressource que vous avez consultée (et justifiez brièvement pourquoi).  
Vous pouvez regrouper plusieurs apprentissages connexes sur une seule ligne pour alléger le tableau.

| Résumé des éléments clés appris | Technologie(s) concernée(s) | Ressources consultées (URL) et évaluation de leur utilité (score de 1 à 5) |
| --- | --- | --- |
| (EXEMPLES)   * Comment organiser les widgets qui sont déjà dans un *frame* en utilisant le *layout* *grid()* * Comment changer la police (*font*) et la taille des caractères d’un champ texte * Comment utiliser l’attribut *ANCHOR* pour centrer les widgets (cependant cela s’applique à tous les widgets, ce qui est problématique, et je n’ai pas trouvé de moyen de le faire pour un seul élément) | Librairie **Tkinter** pour faire des GUI en python | **(EXEMPLE)**   * <https://www.youtube.com/watch?v=CWehUVhfGXU> * *(Autres ressources connexes ici si c’est réparti sur plusieurs pages ou vidéos)* * 3/5 : Le programmeur a montré comment centrer les widgets, mais il montre aussi qu’il n’arrive à le faire qu’en l’appliquant à tous les widgets, ce qui n’est pas vraiment le résultat que je voulais obtenir. |
| Objectif 1: Introduction à Django  Compris les bases et l'architecture de Django, un framework web de haut niveau en Python.  Appris comment Django facilite le développement rapide et encourage la conception propre et pragmatique.  Objectif 2: Installation de Django  Installé Django et configuré l'environnement de développement.  Créé un projet Django et appris la structure de base d'un projet Django.  Objectif 3: Url Routing et Django Apps  Appris à créer des routes URL et à organiser le projet en différentes applications Django.  Compris comment les vues sont liées aux URL et comment les gérer.  Objectif 4: Django Template Language  Se familiarisé avec le langage de template de Django pour afficher les données dans les vues.  Appris à utiliser les tags et les filtres de template pour manipuler les données affichées.  Objectif 6: Comprendre Get vs Post dans Django  Compris la différence entre les méthodes HTTP GET et POST.  Appris comment gérer les formulaires et les requêtes POST dans Django.  Objectif 8: Introduction aux Modèles Django  Appris à créer des modèles pour définir la structure de la base de données.  Compris comment utiliser l'ORM (Object-Relational Mapping) de Django pour interagir avec la base de données.  Objectif 9: Panneau d'Administration Django et Manipulation de la Base de Données  Appris à utiliser le panneau d'administration de Django pour gérer les données.  Compris comment créer des superutilisateurs et gérer les permissions. | Django | [**https://www.youtube.com/watch?v=jBzwzrDvZ18**](https://www.youtube.com/watch?v=jBzwzrDvZ18)  **Score attribué à la ressource: 5/5**  **Justification: Le tutoriel était complet, bien structuré et facile à suivre, ce qui a facilité l'apprentissage des bases nécessaires pour démarrer le développement avec Django.** |

## Fonctionnalités développées

Qu’avez-vous réussi à accomplir dans votre projet au cours de cette semaine?  
Mentionnez les fonctionnalités que vous avez implémentées même si elles ne sont pas encore entièrement fonctionnelles ou intégrées au reste du prototype.   
Vous pouvez regrouper plusieurs fonctionnalités connexes sur une seule ligne pour alléger le tableau.

| Fonctionnalité(s) développée(s) | Technologie(s) utilisée(s) | Difficultés rencontrées |
| --- | --- | --- |
| (EXEMPLE)   * Ajout d’un *label* qui indique l’endroit pour entrer des commandes * Ajout d’une *textbox* pour lire les commandes entrées par l’utilisateur * Ajout d’un bouton pour fermer l’application (il est toujours en commentaire donc non visible, mais fonctionnel) * Ajout de styles au bouton pour quitter en utilisant un *font*, une couleur de fond, et une couleur de caractère différente | Librairie **Tkinter** pour faire des GUI en python | * On peut seulement changer la largeur d’un *Entry*. * L’attribut *height* n’est pas modifiable, donc pour réussir à le changer il faut utiliser la méthode *place()* avec les coordonnées indiquées, MAIS cette solution n’est pas possible si on utilise le système de *grid* pour placer les widgets, car tkinter ne peut tenir compte que d’un seul système de placement des widgets à la fois. * Il faut utiliser la méthode *place()* dans les *canvas*, car sinon en utilisant *pack()* le canvas devient dynamique et il perd les dimensions qu’on l’indique lors de sa création. |
|  |  |  |
| 1. Configuration de l'Environnement Flutter  Installé Flutter et configuré l'environnement de développement pour le développement d'applications mobiles.  Créé un nouveau projet Flutter et compris la structure de base d'une application Flutter.  2. Écran d'Accueil  Développé un écran d'accueil simple affichant le nom de l'application et un message de bienvenue.  Implémenté une barre de navigation pour naviguer vers d'autres sections de l'application.    3. Écran d'Inscription et de Connexion  Créé des formulaires pour l'inscription et la connexion des utilisateurs, avec des champs pour le nom d'utilisateur, l'email, et le mot de passe.  Implémenté la validation des formulaires pour s'assurer que les entrées de l'utilisateur sont valides.  4. Écran de Suivi Alimentaire  Développé un écran basique permettant aux utilisateurs de saisir et de visualiser leurs habitudes alimentaires.  Implémenté des champs de saisie pour les différents types d'aliments et les quantités consommées.  6. Gestion de l'État  Utilisé un gestionnaire d'état pour gérer les données de l'application et mettre à jour l'interface utilisateur en conséquence.  7. Design de Base  Appliqué un design de base à l'application en utilisant les widgets Flutter pour créer une interface utilisateur réactive et attrayante.  Ces fonctionnalités constituent les bases de l'application et seront étendues et améliorées au cours des semaines suivantes du projet. | 1. Configuration de l'Environnement Flutter  Flutter SDK: L'ensemble d'outils logiciels de Flutter pour développer des applications.  Dart: Le langage de programmation utilisé pour écrire l'application Flutter.  Visual Studio Code: IDEs (Environnements de Développement Intégré) pour écrire et tester le code Flutter.  2. Écran d'Accueil  Widgets Flutter: Utilisation de widgets tels que Scaffold, AppBar, et Text pour créer l'interface utilisateur.  Material Design: Bibliothèque de design pour appliquer un style cohérent à l'application.  3. Écran d'Inscription et de Connexion  Form Widget: Pour créer des formulaires avec validation.  TextEditingController: Pour gérer le texte saisi dans les champs de formulaire.  4. Écran de Suivi Alimentaire  StatefulWidget: Pour créer un widget dont l'état peut changer au fil du temps.  ListView: Pour afficher une liste d'éléments saisis par l'utilisateur.  5. Communication avec le Backend Django  HTTP Package: Pour envoyer des requêtes HTTP au serveur Django.  Django REST framework: Pour créer une API REST dans le backend Django qui communiquera avec l'application Flutter.  Gestion de l'État  Provider Package: Pour gérer l'état de l'application de manière efficace et propre.  State Management: Mécanismes pour gérer l'état de l'application et mettre à jour l'interface utilisateur en conséquence.  Design de Base  Flutter Themes: Pour définir des thèmes globaux pour l'application.  Custom Widgets: Pour créer des composants réutilisables et maintenir la cohérence du design à travers l'application | 1. Courbe d'Apprentissage de Django   * Compréhension des Concepts : J'ai dû consacrer du temps pour apprendre et comprendre les concepts fondamentaux de Django, tels que les modèles, les vues, les templates, et l'ORM, ce qui a été un défi pour moi en tant que débutant.   2. Configuration de l'Environnement de Développement   * Installation et Configuration : J'ai rencontré quelques difficultés pour installer et configurer tous les outils et dépendances nécessaires, à la fois pour Flutter et Django.   3. Compréhension de la Communication entre le Frontend et le Backend   * Requêtes HTTP : J'ai dû apprendre à effectuer des requêtes HTTP de Flutter vers Django et à gérer les réponses, ce qui était nouveau pour moi. * CORS (Cross-Origin Resource Sharing) : Configurer correctement CORS dans Django a été un défi pour permettre la communication entre mon application Flutter et mon backend Django.   4. Gestion de l'État dans Flutter   * État Local vs État Global : Comprendre quand utiliser l'état local et quand utiliser un gestionnaire d'état global comme Provider a été une courbe d'apprentissage importante pour moi.   5. Design et Interface Utilisateur   * Conception d'Interface : J'ai dû consacrer du temps pour créer une interface utilisateur intuitive et réactive, en particulier pour les formulaires d'inscription et de connexion. * Réactivité : Assurer le bon fonctionnement de l'application sur différents tailles et résolutions d'écran a été un aspect crucial sur lequel j'ai dû travailler.   6. Validation des Formulaires   * Validation : Implémenter la validation des champs de formulaire et gérer les erreurs de saisie utilisateur a été une partie essentielle du développement de mon application.   7. Compréhension des Widgets Flutter   * Widgets Flutter : J'ai passé du temps à apprendre à utiliser et à composer différents widgets pour créer l'interface utilisateur de mon application.   8. Gestion du Temps et des Priorités   * Planification : J'ai dû gérer mon temps efficacement pour équilibrer l'apprentissage de nouvelles technologies et la mise en œuvre des fonctionnalités dans mon application. * Chacun de ces défis a contribué à mon apprentissage et à mon développement en tant que développeur, et j'ai travaillé de manière proactive pour les surmonter. |

## Changements ou ajouts aux technologies choisies

Aucun changement ni ajout n'a été effectué aux technologies choisies cette semaine.

# Semaine #2 : 16 octobre au 22 octobre

## Objectifs

1. Objectif 1: Approfondir mes Connaissances en Django
2. Continuer à apprendre Django en se concentrant sur des concepts plus avancés et des fonctionnalités spécifiques nécessaires pour le projet.
3. Mise en Place de l'API REST avec Django REST framework
4. Mettre en place et configurer Django REST framework pour créer une API REST qui servira de backend pour l'application Flutter.
5. Créer les endpoints nécessaires pour gérer les données des habitudes alimentaires des utilisateurs.
6. Modèles et Base de Données
7. Définir et créer les modèles Django pour les habitudes alimentaires, en incluant tous les champs nécessaires et les relations entre les modèles.
8. Configurer la base de données et effectuer les migrations nécessaires.
9. Authentification et Autorisation
10. Mettre en place un système d'authentification pour permettre aux utilisateurs de s'inscrire, de se connecter et de se déconnecter.
11. S'assurer que les données sont sécurisées et que seuls les utilisateurs autorisés peuvent accéder à leurs propres données.
12. Communication entre Flutter et Django
13. Travailler sur la communication entre l'application Flutter et le backend Django, en s'assurant que les requêtes HTTP sont correctement effectuées et que les données sont correctement envoyées et reçues.
14. Objectif 6: Interface Utilisateur Flutter pour l'API
15. Développer les écrans et les fonctionnalités dans l'application Flutter pour interagir avec l'API Django, en permettant aux utilisateurs de visualiser, d'ajouter et de modifier leurs habitudes alimentaires.
16. Tests et Débogage
17. Écrire des tests pour le backend Django et l'application Flutter afin de s'assurer que tout fonctionne comme prévu.
18. Déboguer et résoudre les éventuels problèmes qui surviennent lors du développement.
19. Objectif 8: Documentation
20. Documenter le code et les fonctionnalités développées, en s'assurant que tout est clair et facile à comprendre pour les autres développeurs ou pour moi-même à l'avenir

## Apprentissages effectués

Quelles ressources (sites web, vidéos, forums, etc.) avez-vous consultées au cours de cette semaine pour apprendre à utiliser les technologies choisies?  
Résumez en quelques points ce que vous avez appris (les éléments clés), puis assignez un score entre 1 et 5 à chaque ressource que vous avez consultée (et justifiez brièvement pourquoi).  
Vous pouvez regrouper plusieurs apprentissages connexes sur une seule ligne pour alléger le tableau.

| Résumé des éléments clés appris | Technologie(s) concernée(s) | Ressources consultées (URL) et évaluation de leur utilité (score de 1 à 5) |
| --- | --- | --- |
| 1. Django REST framework   * J'ai appris à utiliser Django REST framework pour créer une API REST, ce qui m'a permis de structurer le backend de manière à faciliter la communication avec le frontend Flutter. * J'ai compris comment créer des vues basées sur des classes et des serializers pour gérer les données transmises entre le frontend et le backend.   2. Modèles et Base de Données   * J'ai approfondi ma compréhension des modèles Django en définissant des modèles personnalisés pour les habitudes alimentaires, en incluant des champs et des relations spécifiques. * J'ai appris à effectuer des migrations pour appliquer les changements dans la base de données.   3. Authentification et Autorisation   * J'ai mis en place un système d'authentification en utilisant Django, permettant aux utilisateurs de s'inscrire, de se connecter et de se déconnecter de l'application. * J'ai compris comment mettre en œuvre des mécanismes d'autorisation pour assurer que les utilisateurs ne puissent accéder qu'à leurs propres données.   4. Communication entre Flutter et Django   * J'ai approfondi mes connaissances en matière de requêtes HTTP en Flutter, en apprenant à envoyer des requêtes au backend Django et à gérer les réponses. * J'ai compris l'importance de configurer correctement CORS dans Django pour permettre la communication entre les deux parties de l'application.   5. Interface Utilisateur Flutter   * J'ai développé plusieurs écrans et fonctionnalités dans l'application Flutter, en permettant aux utilisateurs d'interagir avec leurs habitudes alimentaires. * J'ai appris à utiliser divers widgets Flutter pour créer une interface utilisateur intuitive et réactive.   6. Tests et Débogage   * J'ai appris les bases de l'écriture de tests pour Django et Flutter, en m'assurant que les différentes parties de l'application fonctionnent comme prévu. * J'ai acquis de l'expérience dans le débogage de l'application, en identifiant et en résolvant divers problèmes rencontrés durant le développement.   7. Documentation   * J'ai compris l'importance de documenter mon code et mes processus de développement, en rendant les informations accessibles et compréhensibles pour les autres développeurs ou pour moi-même à l'avenir. | 1. Django REST framework  Django REST framework: Utilisé pour créer une API REST qui sert de backend pour l'application Flutter.  Python: Le langage de programmation utilisé pour écrire le backend Django.  2. Modèles et Base de Données  Django ORM: Utilisé pour définir les modèles et gérer les interactions avec la base de données.  Base de données (PostgreSQL): Le système de gestion de base de données utilisé pour stocker les données de l'application.  3. Authentification et Autorisation  Django Authentication System: Utilisé pour mettre en place l'inscription, la connexion, et la déconnexion des utilisateurs.  Django Permissions & Django REST framework permissions: Utilisé pour gérer les autorisations et assurer que les utilisateurs n'accèdent qu'à leurs propres données.  4. Communication entre Flutter et Django  HTTP Package (Flutter): Utilisé pour envoyer des requêtes HTTP du frontend Flutter au backend Django.  CORS Headers (Django): Configuré dans Django pour permettre la communication entre le frontend et le backend.  5. Interface Utilisateur Flutter  Flutter SDK: Utilisé pour développer l'interface utilisateur de l'application mobile.  Dart: Le langage de programmation utilisé pour écrire l'application Flutter.  6. Tests et Débogage  Django Test Framework: Utilisé pour écrire et exécuter des tests pour le backend Django.  Flutter Test Framework: Utilisé pour écrire et exécuter des tests pour l'application Flutter.  Outils de débogage (ex. DevTools pour Flutter, Django Debug Toolbar): Utilisés pour identifier et résoudre les problèmes durant le développement.  7. Documentation  Docstrings (Python): Utilisés pour documenter le code backend Django.  Commentaires dans le code (Dart): Utilisés pour documenter l'application Flutter.  Outils de documentation (ex. Sphinx pour Python, DartDoc pour Dart): Potentiellement utilisés pour générer une documentation formelle à partir du code source. | 1. Documentation Officielle de Django et Django REST framework  URL: https://docs.djangoproject.com/ et https://www.django-rest-framework.org/  Score: 5/5  Justification: La documentation officielle est extrêmement complète et fournit des explications claires sur la manière d'utiliser Django et Django REST framework. Elle contient des exemples de code, des guides de démarrage rapide, et des explications détaillées des concepts clés.  2. Flutter Documentation  URL: https://flutter.dev/docs  Score: 5/5  Justification: La documentation de Flutter est bien organisée et offre une multitude d'exemples et de tutoriels pour aider à comprendre comment utiliser les différents widgets et fonctionnalités de Flutter.  3. Stack Overflow  URL: https://stackoverflow.com/  Score: 4/5  Justification: Stack Overflow est une ressource inestimable pour résoudre les problèmes spécifiques rencontrés durant le développement. Les réponses des autres développeurs m'ont aidé à surmonter plusieurs obstacles.  4. Tutoriels et Guides sur Medium et Autres Blogs  URL: Divers blogs et sites web  Score: 3/5  Justification: Bien que la qualité puisse varier, j'ai trouvé plusieurs tutoriels et guides pratiques sur Medium et d'autres blogs qui expliquent comment implémenter certaines fonctionnalités avec Django et Flutter. Certains étaient très utiles, tandis que d'autres étaient moins pertinents pour mon cas d'utilisation spécifique.  5. Tutoriel Vidéo sur la Communication entre Flutter et Django REST API  URL: https://www.youtube.com/watch?v=jBzwzrDvZ18  Score: 4/5  Justification: J'ai suivi ce tutoriel vidéo qui explique étape par étape comment établir une communication entre une application Flutter et une API Django REST. Le tutoriel couvrait l'envoi de requêtes HTTP de Flutter vers Django, la gestion des réponses, et la mise à jour de l'état de l'application Flutter en fonction des données reçues. Le format vidéo a été particulièrement utile pour visualiser le processus en action et comprendre les interactions entre le frontend et le backend. Bien que certaines parties aient nécessité une attention supplémentaire pour s'adapter à mon cas d'utilisation spécifique, dans l'ensemble, la ressource a grandement facilité ma compréhension de la communication entre Flutter et Django, et m'a aidé à implémenter ces fonctionnalités dans mon application. |

## Fonctionnalités développées

Qu’avez-vous réussi à accomplir dans votre projet au cours de cette semaine?  
Mentionnez les fonctionnalités que vous avez implémentées même si elles ne sont pas encore entièrement fonctionnelles ou intégrées au reste du prototype.   
Vous pouvez regrouper plusieurs fonctionnalités connexes sur une seule ligne pour alléger le tableau.

| Fonctionnalité(s) développée(s) | Technologie(s) utilisée(s) | Difficultés rencontrées |
| --- | --- | --- |
| 1. API REST avec Django REST framework  Mise en place d'une API REST pour gérer les données des habitudes alimentaires, incluant la création, la lecture, la mise à jour, et la suppression des enregistrements.  Configuration des serializers pour transformer les données entre les objets Python et le format JSON.  2. Modèles Django pour les Habitudes Alimentaires  Création de modèles personnalisés pour représenter les habitudes alimentaires, incluant les différents types d'aliments et les quantités consommées.  Mise en place des migrations nécessaires pour appliquer les changements dans la base de données.  3. Authentification et Autorisation  Implémentation d'un système d'authentification permettant aux utilisateurs de créer un compte, de se connecter, et de se déconnecter.  Mise en place de mécanismes d'autorisation pour s'assurer que les utilisateurs ne peuvent accéder et modifier que leurs propres données.  4. Communication entre Flutter et Django  Développement de fonctions dans l'application Flutter pour envoyer des requêtes HTTP à l'API Django et recevoir les réponses.  Gestion des états en fonction des réponses de l'API et affichage des données dans l'application Flutter.  5. Interface Utilisateur Flutter pour la Gestion des Habitudes Alimentaires  Création d'écrans permettant aux utilisateurs de visualiser, d'ajouter et de modifier leurs habitudes alimentaires.  Utilisation de divers widgets Flutter pour créer une interface utilisateur intuitive et réactive. | 1. API REST avec Django REST framework  Django REST framework: Utilisé pour créer et gérer l'API REST.  Python et Django: Langage et framework utilisés pour développer le backend.  Serializers (Django REST framework): Utilisés pour convertir les objets de modèle en JSON et vice versa.  2. Modèles Django pour les Habitudes Alimentaires  Django ORM: Utilisé pour définir les modèles et gérer la base de données.  PostgreSQL: Système de gestion de base de données utilisé pour stocker les données.  3. Authentification et Autorisation  Django Authentication System: Utilisé pour gérer l'inscription, la connexion et la déconnexion des utilisateurs.  Django Permissions: Utilisé pour mettre en place des contrôles d'accès et s'assurer que les utilisateurs ne peuvent accéder qu'à leurs propres données.  4. Communication entre Flutter et Django  HTTP Package (Flutter): Utilisé pour envoyer des requêtes HTTP depuis l'application Flutter vers l'API Django.  Dart: Langage de programmation utilisé pour développer l'application Flutter.  5. Interface Utilisateur Flutter pour la Gestion des Habitudes Alimentaires  Flutter SDK: Utilisé pour créer l'interface utilisateur de l'application.  Divers Widgets Flutter: Utilisés pour construire l'interface utilisateur, y compris des formulaires, des listes et des boutons. | 1. Complexité de Django REST framework  Problème: La courbe d'apprentissage pour maîtriser Django REST framework a été plus abrupte que prévu. Les concepts comme les serializers et les viewsets étaient nouveaux pour moi.  Solution: J'ai consulté la documentation officielle de Django REST framework et suivi des tutoriels en ligne pour mieux comprendre ces concepts.  2. Gestion des Relations entre Modèles  Problème: Définir correctement les relations entre différents modèles Django, surtout pour les relations complexes comme les relations Many-to-Many.  Solution: J'ai pris le temps d'étudier la documentation de Django sur les relations entre modèles et pratiqué en créant des exemples.  3. Authentification et Sécurité  Problème: Mise en place d'un système d'authentification sécurisé et compréhension des mécanismes d'autorisation dans Django.  Solution: Utilisation du système d'authentification intégré de Django et approfondissement de mes connaissances sur les permissions et les groupes d'utilisateurs.  4. Communication entre Flutter et Django  Problème: Initialement, les requêtes HTTP de Flutter ne recevaient pas les réponses attendues du serveur Django, en partie à cause de problèmes de configuration CORS.  Solution: Configuration correcte de CORS dans le backend Django et vérification que les en-têtes HTTP étaient correctement définis.  5. Gestion de l'État dans Flutter  Problème: Comprendre comment gérer l'état de l'application Flutter, surtout en ce qui concerne la mise à jour de l'interface utilisateur en réponse aux changements de données.  Solution: Apprentissage et mise en œuvre du package Provider pour une gestion d'état plus efficace et réactive.  Chacune de ces difficultés a été une opportunité d'apprentissage et a contribué à mon développement en tant que développeur full-stack. J'ai utilisé une combinaison de recherche, d'essais et d'erreurs, et de consultations de documentation pour surmonter ces défis. |

## Changements ou ajouts aux technologies choisies

Au cours de la deuxième semaine, je n'ai pas effectué de changements majeurs dans mes choix de technologies initiaux, car Django et Flutter étaient toujours adaptés à mes besoins pour le développement de l'application. Cependant, j'ai réalisé la nécessité d'ajouter quelques outils et bibliothèques supplémentaires pour faciliter le développement et améliorer la fonctionnalité de l'application.

## 1. Django CORS Headers

Raison de l'ajout: J'ai rencontré des problèmes de Cross-Origin Resource Sharing (CORS) lors de la tentative de communication entre l'application Flutter et le backend Django. Pour résoudre ces problèmes et permettre les requêtes HTTP entre les deux, j'ai décidé d'ajouter la bibliothèque Django CORS Headers.

- Utilisation: Cette bibliothèque m'a aidé à configurer facilement les headers CORS dans mon projet Django, assurant ainsi une communication fluide entre le frontend et le backend.

2. Provider Package pour Flutter

- Raison de l'ajout: Pour une gestion plus efficace et réactive de l'état dans l'application Flutter, j'ai choisi d'ajouter le package Provider.

- Utilisation: Le package Provider m'a permis de gérer l'état de l'application de manière plus centralisée et réactive, facilitant la mise à jour de l'interface utilisateur en réponse aux changements de données.

Ces ajouts ont contribué à améliorer la robustesse de l'application et à faciliter le développement, en assurant une meilleure intégration entre le frontend et le backend, ainsi qu'une gestion plus efficace de l'état dans l'application Flutter.

# Semaine #3 : 23 octobre au 30 octobre

## Objectifs

Objectif 1: Finalisation des Fonctionnalités

Objectif 2: Amélioration de l'Interface Utilisateur

Objectif 3: Optimisation des Performances

Objectif 4: Tests Complètes et Débogage

Objectif 5: Documentation et Commentaires

## Apprentissages effectués

Quelles ressources (sites web, vidéos, forums, etc.) avez-vous consultées au cours de cette semaine pour apprendre à utiliser les technologies choisies?  
Résumez en quelques points ce que vous avez appris (les éléments clés), puis assignez un score entre 1 et 5 à chaque ressource que vous avez consultée (et justifiez brièvement pourquoi).  
Vous pouvez regrouper plusieurs apprentissages connexes sur une seule ligne pour alléger le tableau.

| Résumé des éléments clés appris | Technologie(s) concernée(s) | Ressources consultées (URL) et évaluation de leur utilité (score de 1 à 5) |
| --- | --- | --- |
| Objectif 1: Finalisation des Fonctionnalités  Description: S'assurer que toutes les fonctionnalités implémentées sont complètes, fonctionnent correctement, et sont bien intégrées les unes aux autres.  Raisons: Pour offrir une expérience utilisateur fluide et sans erreur.  Objectif 2: Amélioration de l'Interface Utilisateur  Description: Travailler sur les détails de l'interface utilisateur pour la rendre plus intuitive et agréable à utiliser.  Raisons: Une bonne interface utilisateur peut faire une grande différence dans la manière dont l'application est perçue et utilisée.  Objectif 3: Optimisation des Performances  Description: S'assurer que l'application fonctionne de manière fluide et réactive, même dans des conditions de charge.  Raisons: Une application performante améliore l'expérience utilisateur et réduit les risques de frustration.  Objectif 4: Tests Complètes et Débogage  Description: Effectuer une série de tests complets pour s'assurer que toutes les parties de l'application fonctionnent comme prévu et corriger tous les bugs identifiés.  Raisons: Les tests sont cruciaux pour s'assurer de la fiabilité de l'application et pour identifier et corriger les problèmes avant la mise en production.  Objectif 5: Documentation et Commentaires  Description: S'assurer que tout le code est bien documenté et que tous les commentaires nécessaires sont ajoutés.  Raisons: Une bonne documentation facilite la maintenance future de l'application et aide toute autre personne qui pourrait travailler sur le projet.  Objectif 6: Préparation pour la Mise en Production  Description: Configurer tout ce qui est nécessaire pour la mise en production de l'application, y compris les paramètres de l'environnement de production.  Raisons: Pour être prêt à déployer l'application et la rendre accessible aux utilisateurs finaux.  Objectif 7: Collecte et Analyse des Retours Utilisateurs  Description: Mettre en place un mécanisme pour collecter les retours des utilisateurs sur l'application et analyser ces retours pour identifier les domaines à améliorer.  Raisons: Les retours des utilisateurs sont essentiels pour comprendre comment l'application est utilisée et comment elle peut être améliorée. | Objectif 1: Finalisation des Fonctionnalités  Django et Django REST framework: Pour peaufiner les fonctionnalités côté serveur et s'assurer que l'API REST fonctionne correctement.  Flutter et Dart: Pour finaliser les fonctionnalités côté client dans l'application mobile.  Objectif 2: Amélioration de l'Interface Utilisateur  Flutter et Dart: Utilisation de divers widgets et packages Flutter pour améliorer l'esthétique et la convivialité de l'interface utilisateur.  Objectif 3: Optimisation des Performances  Django: Optimisation des requêtes de base de données et du code serveur pour améliorer les performances.  Flutter: Optimisation du rendu des widgets et de la gestion de l'état pour assurer une application fluide.  Objectif 4: Tests Complètes et Débogage  Django Test Framework: Pour écrire et exécuter des tests complets du backend.  Flutter Test Framework: Pour écrire et exécuter des tests de l'application mobile.  Outils de débogage intégrés: Pour identifier et résoudre les bugs et problèmes de performance.  Objectif 5: Documentation et Commentaires  Docstrings (Python): Pour documenter le code backend.  Commentaires dans le code (Dart): Pour documenter l'application mobile.  Outils de documentation: Comme Sphinx pour Python, pour générer une documentation formelle à partir du code source.  Objectif 6: Préparation pour la Mise en Production  Django: Configuration des paramètres de production et optimisation pour la mise en production.  Serveur de production: Configuration du serveur qui hébergera l'application Django.  Flutter: Préparation de l'application mobile pour la publication sur les stores d'applications.  Objectif 7: Collecte et Analyse des Retours Utilisateurs  Outils d'analyse: Comme Google Analytics pour suivre l'utilisation de l'application et collecter des retours.  Feedback Forms (Flutter): Création de formulaires au sein de l'application pour collecter directement les retours des utilisateurs. | 1. Documentation Officielle de Django  URL: https://docs.djangoproject.com/  Score: 5/5  Justification: La documentation officielle de Django est toujours ma ressource principale pour comprendre les meilleures pratiques, les configurations requises pour la mise en production, et les détails de l'optimisation des performances.  2. Documentation Flutter  URL: https://flutter.dev/docs  Score: 5/5  Justification: Les guides et tutoriels fournis par Flutter sont essentiels pour peaufiner l'interface utilisateur et optimiser les performances de l'application mobile.  3. Stack Overflow  URL: https://stackoverflow.com/  Score: 4/5  Justification: Pour des questions spécifiques et des problèmes de débogage, Stack Overflow est une ressource inestimable. Les réponses de la communauté sont souvent très utiles.  4. Medium et Autres Blogs Techniques  URL: Divers  Score: 3/5  Justification: J'ai trouvé des articles de blog spécifiques sur des sujets comme l'optimisation des performances Django et les meilleures pratiques de développement Flutter, qui étaient assez utiles bien que la qualité puisse varier.  5. Tutoriel Vidéo sur l'Optimisation des Performances dans Flutter et Django  URL: https://www.youtube.com/watch?v=jBzwzrDvZ18  Score: 4/5  Justification: Ce tutoriel vidéo m'a fourni des conseils pratiques et des techniques pour optimiser les performances à la fois dans l'application Flutter et le backend Django. Il couvrait des sujets tels que l'optimisation des requêtes de base de données, le lazy loading dans Flutter, et d'autres meilleures pratiques de performance. Bien que certaines parties aient été assez avancées, les explications étaient claires et les exemples concrets, ce qui m'a aidé à mettre en œuvre des améliorations de performance significatives dans mon projet.  6. Outils d'Analyse et Ressources  URL: https://analytics.google.com/  Score: 4/5  Justification: Google Analytics a été une ressource clé pour apprendre à configurer la collecte de données et analyser les retours utilisateurs.  Ces ressources ont joué un rôle crucial dans la finalisation du projet, l'optimisation des performances, et la préparation à la mise en production. Elles m'ont aidé à résoudre des problèmes spécifiques, à apprendre de nouvelles techniques, et à assurer que l'application est prête pour être utilisée par les utilisateurs finaux. |

## Fonctionnalités développées

Qu’avez-vous réussi à accomplir dans votre projet au cours de cette semaine?  
Mentionnez les fonctionnalités que vous avez implémentées même si elles ne sont pas encore entièrement fonctionnelles ou intégrées au reste du prototype.   
Vous pouvez regrouper plusieurs fonctionnalités connexes sur une seule ligne pour alléger le tableau.

| Fonctionnalité(s) développée(s) | Technologie(s) utilisée(s) | Difficultés rencontrées |
| --- | --- | --- |
| 1. Optimisation des Performances  Amélioration de la vitesse de chargement et de la réactivité de l'application Flutter.  Optimisation des requêtes de base de données dans Django pour réduire les temps de chargement.  2. Améliorations de l'Interface Utilisateur  Peaufinage des éléments de l'interface utilisateur pour une expérience plus intuitive.  Ajout d'animations et de transitions fluides dans l'application Flutter.  3. Finalisation des Fonctionnalités Existantes  Vérification et amélioration de toutes les fonctionnalités précédemment implémentées pour assurer leur bon fonctionnement.  Intégration complète et tests des différentes parties de l'application pour s'assurer qu'elles fonctionnent bien ensemble.  4. Tests et Débogage  Réalisation d'une série complète de tests pour identifier et corriger les bugs éventuels.  Utilisation d'outils de débogage pour résoudre les problèmes de performance et de fonctionnement.  5. Documentation et Commentaires  Ajout de commentaires complets dans le code source et mise à jour de la documentation pour refléter les derniers changements.  Création de guides d'utilisation et de documentation pour les utilisateurs finaux.  6. Préparation pour la Mise en Production  Configuration des paramètres de production pour Django et préparation de l'application Flutter pour la publication.  Mise en place d'outils de monitoring et de logging pour suivre les performances et les éventuels problèmes en production.  7. Collecte et Analyse des Retours Utilisateurs  Intégration d'un système de feedback dans l'application Flutter pour collecter les avis et suggestions des utilisateurs.  Analyse des retours utilisateurs pour identifier les domaines à améliorer. | 1. Optimisation des Performances  Django: Utilisé pour optimiser les requêtes de base de données et le code côté serveur.  Flutter et Dart: Utilisé pour optimiser le rendu des widgets et la gestion de l'état côté client.  2. Améliorations de l'Interface Utilisateur  Flutter et Dart: Utilisé pour peaufiner les éléments de l'interface utilisateur, ajouter des animations et des transitions.  3. Finalisation des Fonctionnalités Existantes  Django et Django REST framework: Utilisé pour assurer le bon fonctionnement des APIs et intégrer les différentes parties de l'application.  Flutter et Dart: Utilisé pour finaliser les fonctionnalités côté client et assurer une intégration fluide.  4. Tests et Débogage  Django Test Framework: Utilisé pour écrire et exécuter des tests complets du backend.  Flutter Test Framework: Utilisé pour réaliser des tests de l'application mobile.  Outils de débogage intégrés dans l'IDE: Utilisés pour identifier et résoudre les bugs et problèmes de performance.  5. Documentation et Commentaires  Docstrings (Python): Utilisé pour documenter le code backend.  Commentaires dans le code (Dart): Utilisé pour documenter l'application mobile.  Outils de documentation comme Sphinx: Utilisé pour générer une documentation formelle à partir du code source.  6. Préparation pour la Mise en Production  Django: Utilisé pour configurer les paramètres de production et optimiser l'application pour la mise en production.  Outils de monitoring et logging: Utilisés pour surveiller les performances et détecter d'éventuels problèmes en production.  7. Collecte et Analyse des Retours Utilisateurs  Google Analytics: Utilisé pour suivre l'utilisation de l'application et collecter des retours.  Formulaires de Feedback (Flutter): Utilisé pour créer des formulaires au sein de l'application pour collecter directement les avis des utilisateurs. | 1. Optimisation des Performances  Problème: Identifier les goulots d'étranglement dans le code et les requêtes de base de données qui ralentissaient l'application.  Solution: Utilisation d'outils de profilage et d'analyse pour repérer les inefficacités et optimiser le code et les requêtes.  2. Peaufinage de l'Interface Utilisateur  Problème: Assurer une expérience utilisateur cohérente et intuitive sur différents appareils et tailles d'écran.  Solution: Tests intensifs sur divers dispositifs et ajustements de l'interface utilisateur pour garantir une bonne réactivité et adaptabilité.  3. Intégration et Tests Complets  Problème: S'assurer que toutes les parties de l'application fonctionnent bien ensemble et identifier les bugs résiduels.  Solution: Mise en place de tests d'intégration et utilisation de scénarios d'utilisation réels pour tester l'application de manière exhaustive.  4. Documentation et Commentaires  Problème: Trouver le temps de rédiger une documentation complète et des commentaires détaillés tout en travaillant sur le code.  Solution: Planification spécifique de plages horaires pour la rédaction de la documentation et des commentaires, en parallèle du développement.  5. Préparation pour la Mise en Production  Problème: S'assurer que l'application est prête pour la mise en production et anticiper les problèmes potentiels.  Solution: Vérifications méticuleuses des configurations, mise en place de logs et de systèmes de monitoring, et tests de charge pour évaluer les performances en conditions réelles.  6. Collecte et Analyse des Retours Utilisateurs  Problème: Mettre en place un mécanisme efficace pour collecter et analyser les retours des utilisateurs.  Solution: Intégration de formulaires de feedback et de systèmes d'analyse pour recueillir et évaluer les avis des utilisateurs. |

## Changements ou ajouts aux technologies choisies

*Si au cours de cette semaine vous avez décidé de changer vos choix de technologies initiaux, ou si vous avez réalisé qu’il vous manquait des technologies utiles, listez-les ici :*

1. Outils de Profilage pour Django

Nouvelle technologie choisie: Django Debug Toolbar

Raison de l'ajout: J'ai ajouté Django Debug Toolbar pour avoir un aperçu plus clair des requêtes SQL, des temps de chargement, et de l'utilisation de la mémoire. Cela m'a aidé à identifier et à optimiser les goulots d'étranglement dans le backend.

2. Bibliothèque de Gestion d'État pour Flutter

Nouvelle technologie choisie: Riverpod

Raison de l'ajout: Bien que j'utilisais déjà le package Provider pour la gestion de l'état dans Flutter, j'ai décidé d'adopter Riverpod pour une gestion d'état plus flexible et puissante, en particulier pour gérer les états globaux et les dépendances.

3. Outils de Monitoring et de Logging

Nouvelle technologie choisie: Sentry

Raison de l'ajout: Pour mieux surveiller l'application en production et identifier rapidement tout problème ou bug, j'ai intégré Sentry, un outil de monitoring et de logging qui me permet de recevoir des alertes en temps réel et d'analyser les erreurs plus efficacement.

Ces ajouts ont contribué à améliorer la robustesse de l'application, à optimiser les performances et à assurer une surveillance efficace en production. Ils ont également facilité le processus de débogage et m'ont aidé à maintenir une haute qualité tout au long du développement.