# Table des matières



# I. Introduction

La ferme "Maya" est une exploitation agricole familiale située dans une région rurale. Forte de ses traditions ancestrales et de son engagement envers des pratiques agricoles durables, la ferme Maya produit une variété de produits biologiques de haute qualité, allant des fruits et légumes frais aux produits laitiers et à la viande.

Malgré son succès dans la production agricole, la ferme Maya reconnaît l'importance croissante de la technologie pour optimiser ses opérations et améliorer son service à la clientèle. Ainsi, consciente des avantages d'une présence numérique accrue, la ferme Maya a décidé de se tourner vers le développement d'une application mobile personnalisée pour mieux servir ses clients et dynamiser ses activités.

Cette application, baptisée "Emaya", vise à offrir une expérience améliorée à ses clients en leur permettant de découvrir et d'acheter facilement les produits de la ferme. De plus, elle vise à renforcer l'engagement de la communauté locale envers la ferme Maya en fournissant des informations sur les événements à venir, les promotions spéciales et les pratiques agricoles durables.

La ferme Maya souhaite que l'application Emaya soit conviviale, intuitive et reflète l'identité unique de leur marque. Ils cherchent à collaborer avec des développeurs expérimentés capables de comprendre leurs besoins spécifiques et de créer une application mobile qui répondra aux attentes élevées de leurs clients tout en renforçant leur présence sur le marché numérique.

## 1) Flutter c’est quoi

Flutter est un framework open-source développé par Google, utilisé pour créer des applications multiplateformes. Il permet aux développeurs de créer des applications pour Android, iOS, Windows, Mac, Linux et le web à partir d'une seule base de code.

Flutter utilise le langage de programmation Dart, également développé par Google, qui offre des performances élevées et une syntaxe moderne. L'un des avantages clés de Flutter est sa capacité à fournir une interface utilisateur réactive et fluide grâce à sa technologie de rendu personnalisée appelée "Skia".

Les principales caractéristiques de Flutter incluent son architecture basée sur des widgets, qui permet une personnalisation facile et une expérience utilisateur cohérente sur différentes plateformes. De plus, Flutter dispose d'un ensemble complet de widgets prêts à l'emploi pour la création d'interfaces utilisateur riches et attrayantes.

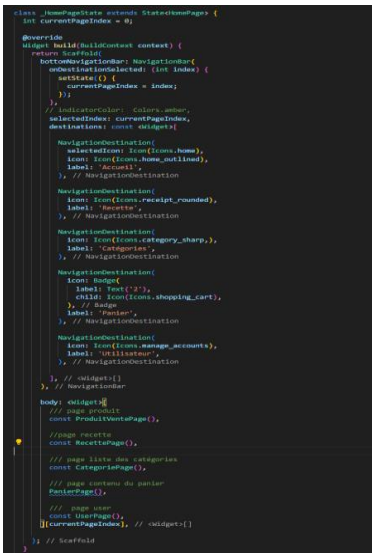
En résumé, Flutter est un outil puissant pour le développement d'applications multiplateformes, offrant une productivité élevée, des performances optimisées et une expérience utilisateur de qualité.

# II. Barre de Navigation


Le code de la  classe `HomePage` gère l'état de la page d'accueil, avec un index `currentPageIndex` pour suivre la destination sélectionnée.

Dans la méthode `build`, un Scaffold est utilisé pour la mise en page de l'application. La barre de navigation (`bottomNavigationBar`) contient des instances de `NavigationDestination`, représentant chaque destination. En fonction de la destination sélectionnée, le corps affiche la page correspondante, comme la page de produits, la page de recettes, la page de catégories, la page de panier ou la page utilisateur.

En résumé, ce code permet à l'utilisateur de naviguer entre différentes fonctionnalités de l'application à l'aide d'une barre de navigation intuitive, offrant une expérience utilisateur fluide et organisée.



# III. Accueil


## 1) ProduitPromotionWidget

- Ce widget représente un produit en promotion et est conçu pour être utilisé dans une liste de produits en promotion.

- Il est encapsulé dans un Material avec une couleur de carte légèrement transparente pour donner un effet visuel agréable.

- Lorsque l'utilisateur appuie sur cette carte, aucune action spécifique n'est déclenchée pour l'instant.

- La carte contient une image du produit et son nom. Le nom est suivi d'une indication "Le kilo", peut-être pour indiquer que le prix est par kilogramme.

## 2) ProduitVentePage

Cette classe représente une page dédiée à la vente de produits, y compris ceux qui sont en promotion.

La page commence par un Swiper, qui est un widget permettant de faire défiler des images. Il affiche des images associées à la marque "Maya".

Sous le Swiper, il y a un bouton "Voir tout" qui semble permettre à l'utilisateur de voir tous les produits en promotion. Cependant, l'action de ce bouton n'est pas encore implémentée.

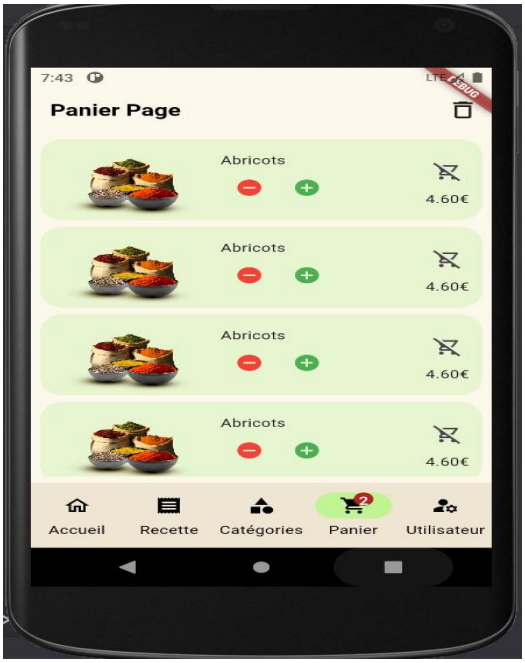
Ensuite, il y a une section "Promotions" qui affiche une liste horizontale de produits en promotion. Chaque produit est représenté par un widget ProduitVenteWidget.

Après la section des promotions, il y a une section "Nos produits" qui contient une liste de produits disponibles. Il y a un bouton "Afficher tout" à côté du titre, mais son action n'est pas encore définie.

Enfin, la page présente une grille de produits en vente. La grille est limitée à huit produits pour l'affichage, et chaque produit est représenté par un widget ProduitVenteWidget.

Pendant que les produits sont chargés, un indicateur de chargement est affiché au centre de la page pour informer l'utilisateur que les données sont en cours de chargement.

## IV. Panier




La page `PanierPage` affiche le contenu du panier de l'utilisateur. Voici un résumé de son fonctionnement :

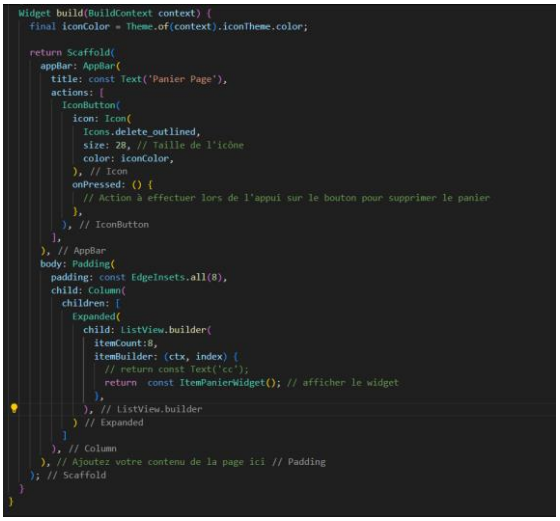
La barre d'applications affiche le titre "Panier Page" avec un bouton d'action en haut à droite. Ce bouton est représenté par une icône de corbeille qui permettra probablement à l'utilisateur de vider son panier.

Le corps de la page contient une liste de widgets `ItemPanierWidget`. Ces widgets représentent chaque élément contenu dans le panier de l'utilisateur. La liste est construite à l'aide d'un `ListView.builder`.

Chaque `ItemPanierWidget` affiche les détails d'un élément du panier, y compris une image, le nom du produit, la quantité sélectionnée avec des boutons "+" et "-", ainsi que le prix et une option pour supprimer l'élément du panier.

Dans chaque `ItemPanierWidget`, les boutons "+" et "-" permettent à l'utilisateur d'ajuster la quantité du produit dans son panier. Cela est géré par l'état local du widget.

Lorsque l'utilisateur appuie sur le bouton de suppression dans un `ItemPanierWidget`, une action peut être déclenchée. Pour l'instant, cette action est laissée vide dans le code.



```dart
Widget build(BuildContext context) {
  final iconColor = Theme.of(context).iconTheme.color;

  return Scaffold(
    appBar: AppBar(
      title: const Text('Panier Page'),
      actions: [
        IconButton(
          icon: Icon(
            Icons.delete_outlined,
            size: 28, // Taille de l'icône
            color: iconColor,
          ), // Icon
          onPressed: () {
            // Action à effectuer lors de l'appui sur le bouton pour supprimer le panier
          },
        ), // IconButton
      ],
    ), // AppBar
    body: Padding(
      padding: const EdgeInsets.all(8),
      child: Column(
        children: [
          Expanded(
            child: ListView.builder(
              itemCount:8,
              itemBuilder: (ctx, index) {
                // return const Text('cc');
                return  const ItemPanierWidget(); // afficher le widget
              },
            ), // ListView.builder
          ) // Expanded
        ]
      ), // Column
    ), // Ajoutez votre contenu de la page ici // Padding
  ); // Scaffold
 }
}
```

Le widget `ItemPanierWidget` représente un élément individuel du panier dans l'interface utilisateur. Voici un résumé de son fonctionnement :

Ce widget est un StatefulWidget, ce qui signifie qu'il peut maintenir un état interne qui peut être modifié au fil du temps.

L'état interne de ce widget comprend un compteur `_counter`, qui est utilisé pour suivre la quantité de l'élément dans le panier.

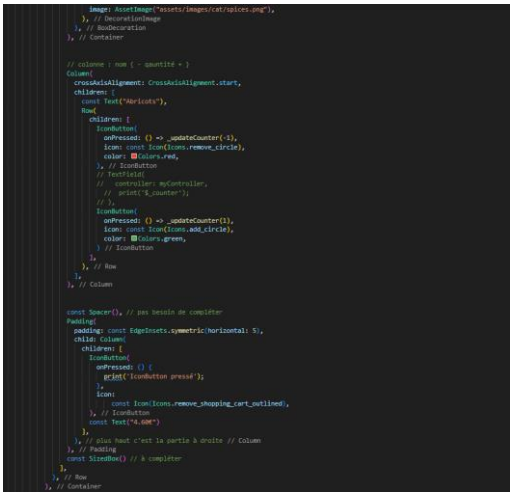
Il utilise également un `TextEditingController` appelé `myController`, bien que pour l'instant il ne soit pas utilisé dans le code.

Lorsque l'utilisateur appuie sur les boutons "+" ou "-", la méthode `_updateCounter` est appelée pour ajuster la valeur du compteur en fonction de la quantité souhaitée. Cette méthode met à jour l'état du widget en utilisant `setState` pour refléter les changements dans l'interface utilisateur.

Le widget est construit avec une rangée (Row) contenant plusieurs éléments :

- Une image représentant le produit dans le panier.
- Le nom du produit et des boutons "+" et "-" pour ajuster la quantité.
- Une zone pour supprimer l'élément du panier et afficher le prix.

La disposition des éléments dans le widget est gérée à l'aide de Row, Column et Spacer pour obtenir la mise en page souhaitée.



## V. Utilisateur


Dans ma conception d'application, j'ai pris soin de personnaliser la barre d'applications avec une classe appelée `CustomAppBar`. Cette barre d'applications affiche un titre dynamique et un icône d'utilisateur à côté. L'idée est d'offrir à l'utilisateur une expérience cohérente et intuitive dès qu'il entre dans la page utilisateur.

En entrant dans la page utilisateur, vous remarquerez une disposition organisée des différents éléments. J'ai utilisé une colonne pour aligner verticalement les options disponibles. Voici ce que vous pouvez trouver :

Gestion du téléphone : Une option qui permet à l'utilisateur de gérer son numéro de téléphone. En cliquant dessus, l'utilisateur peut accéder à une fonctionnalité pour modifier ou mettre à jour son numéro de téléphone.

Commandes : Cette option permet à l'utilisateur de consulter ses commandes passées. Lorsqu'il clique dessus, l'application le redirige vers une page dédiée aux commandes, où il peut voir les détails de chaque commande.

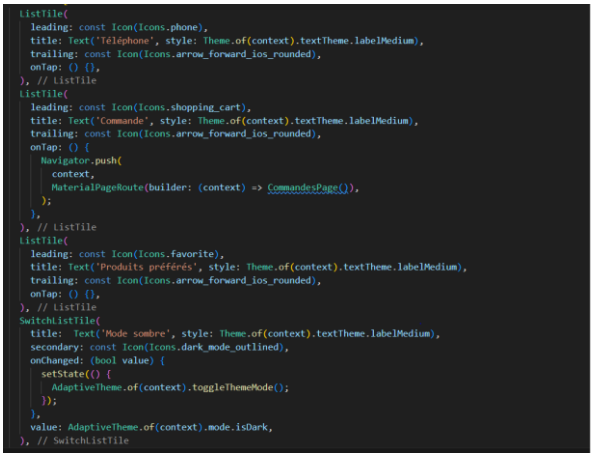
Produits préférés: Une section où l'utilisateur peut accéder à ses produits préférés. Cela peut être utile pour retrouver rapidement les articles qu'il aime acheter régulièrement.

Mode sombre: J'ai ajouté une fonctionnalité intéressante qui permet à l'utilisateur de basculer entre le mode sombre et le mode clair. En activant ou désactivant le curseur, l'interface utilisateur s'adapte instantanément à la préférence de l'utilisateur. J'ai utilisé le package `adaptive_theme` pour faciliter cette transition.

Changer de mot de passe Une option pour permettre à l'utilisateur de modifier son mot de passe. En cliquant dessus, l'application peut afficher un formulaire de changement de mot de passe pour l'utilisateur.

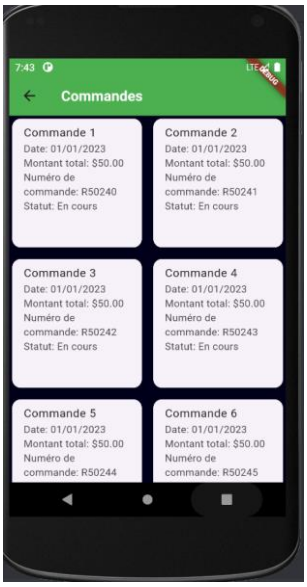
Se déconnecterEnfin, une option simple pour permettre à l'utilisateur de se déconnecter de son compte. Cela le redirigerait probablement vers l'écran de connexion ou afficherait une boîte de dialogue de confirmation.

En résumé, j'ai conçu la page utilisateur de mon application pour offrir une expérience fluide et intuitive à l'utilisateur. J'ai inclus des fonctionnalités essentielles telles que la gestion du profil, la consultation des commandes et la personnalisation du thème, tout en veillant à ce que l'interface reste propre et organisée pour une utilisation facile.



```
ListTile(
  leading: const Icon(Icons.phone),
  title: Text('Téléphone', style: Theme.of(context).textTheme.labelMedium),
  trailing: const Icon(Icons.arrow_forward_ios_rounded),
  onTap: () {},
), // ListTile
ListTile(
  leading: const Icon(Icons.shopping_cart),
  title: Text('Commande', style: Theme.of(context).textTheme.labelMedium),
  trailing: const Icon(Icons.arrow_forward_ios_rounded),
  onTap: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => CommandesPage()),
    );
  },
), // ListTile
ListTile(
  leading: const Icon(Icons.favorite),
  title: Text('Produits préférés', style: Theme.of(context).textTheme.labelMedium),
  trailing: const Icon(Icons.arrow_forward_ios_rounded),
  onTap: () {},
), // ListTile
SwitchListTile(
  title:  Text('Mode sombre', style: Theme.of(context).textTheme.labelMedium),
  secondary: const Icon(Icons.dark_mode_outlined),
  onChanged: (bool value) {
    setState(() {
      AdaptiveTheme.of(context).toggleThemeMode();
    });
  },
  value: AdaptiveTheme.of(context).mode.isDark,
), // SwitchListTile
```

## 1) Commande




Cette page `CommandesPage` affiche une liste de commandes dans une grille. Voici un résumé de son fonctionnement :

La barre d'applications affiche le titre "Commandes" avec un fond vert.

Le corps de la page contient une grille de commandes, chaque commande étant représentée par une carte dans la grille.

La grille est construite à l'aide de `GridView.builder`, ce qui permet une construction dynamique des éléments en fonction d'un nombre défini.

Chaque carte dans la grille représente une commande. Elle contient :

Un titre indiquant le numéro de commande.

Un sous-titre avec des détails de la commande, tels que la date, le montant total, le numéro de commande et le statut.

Lorsque l'utilisateur appuie sur une commande, rien ne se passe pour l'instant, car la fonction `onTap` est laissée vide.

La grille a deux colonnes grâce à `crossAxisCount: 2`, ce qui signifie que deux cartes sont affichées côte à côte dans chaque ligne.

Les espacements entre les colonnes et les lignes sont définis par `crossAxisSpacing` et `mainAxisSpacing`.

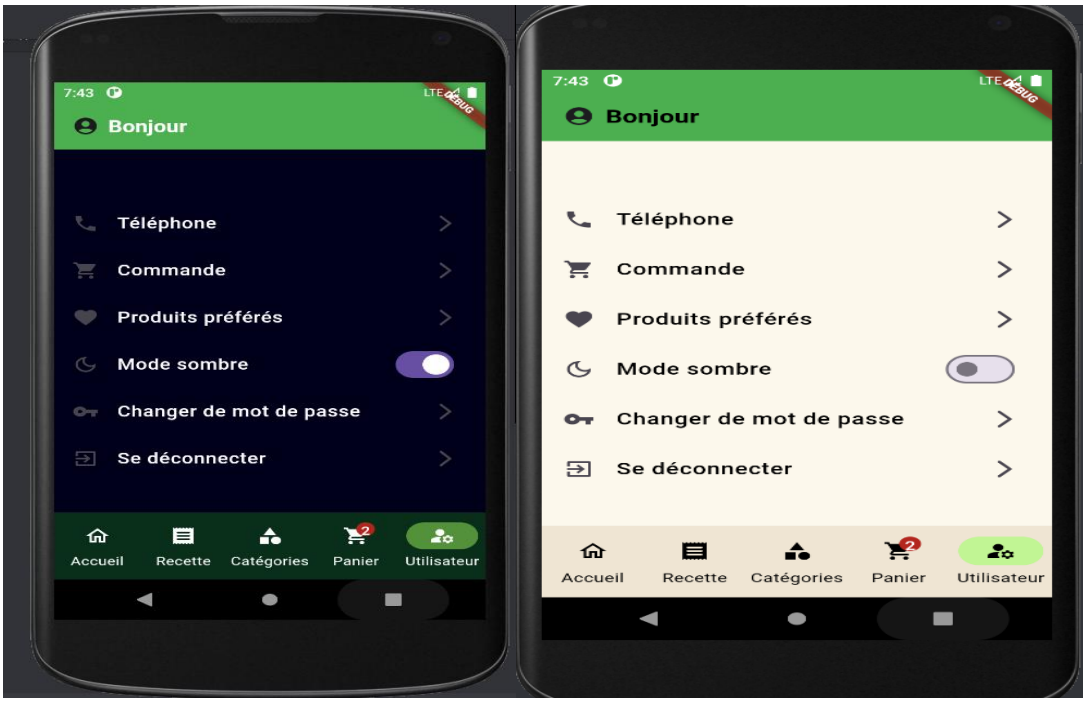
En résumé, cette page fournit à l'utilisateur un aperçu clair de ses commandes passées, en affichant les détails pertinents de chaque commande dans une disposition organisée et visuellement attrayante.

```dart
import 'package:flutter/material.dart';

class CommandesPage extends StatelessWidget {
  const CommandesPage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Commandes'),
        backgroundColor: Colors.green,
      ), // AppBar
      body: GridView.builder(
        gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: 2, // Nombre cartes
          crossAxisSpacing: 8.0,
          mainAxisSpacing: 8.0,
        ), // SliverGridDelegateWithFixedCrossAxisCount
        itemCount: 15, // Nombre de commandes
        itemBuilder: (context, index) {
          return Card(
            child: ListTile(
              title: Text('Commande ${index + 1}'), // Titre commande
              subtitle: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text('Date: 01/01/2023'), // Date
                  Text('Montant total: \$50.00'), // Montant total
                  Text("Numéro de commande: R5024${index}"),
                  Text('Statut: En cours'), // Statut
                ],
              ), // Column
              onTap: () {},
            ), // ListTile
          ); // Card
        },
      ), // GridView.builder
    ); // Scaffold
```

## VI. Mode Jour/Nuit




Le code définit deux thèmes pour l'application, un thème clair (`lightThemeData`) et un thème sombre (`darkThemeData`). Ces thèmes sont utilisés pour personnaliser l'apparence de l'application en fonction du mode de luminosité de l'appareil de l'utilisateur.

Voici un aperçu des caractéristiques principales de chaque thème :

### 1) Thème Clair (`lightThemeData`)

- La couleur d'arrière-plan de la barre d'applications est définie sur `AppColor.lightBackGroundColor`.

- La couleur de la police dans la barre d'applications est noire.

- La couleur d'arrière-plan de la page est également définie sur `AppColor.lightBackGroundColor`.

- Les couleurs principales de l'application sont définies sur `AppColor.lightPrimaryColor`.

- Les couleurs du texte sont définies en fonction de la taille du texte (`labelSmall`, `labelMedium`, `labelLarge`) avec une couleur de police noire.

- Les couleurs des cartes et des canevas sont définies sur `AppColor.lightCardColor` et `AppColor.lightCanvasColor` respectivement.

- Les couleurs de la barre de navigation sont définies sur `AppColor.lightNavBackGroundColor` pour l'arrière-plan et sur `AppColor.lightIndicatorColor` pour la couleur de l'indicateur. La police est noire.

### 2) Thème Sombre (`darkThemeData`)

- La couleur d'arrière-plan de la barre d'applications est définie sur `AppColor.darkBackGroundColor`.

- La couleur de la police dans la barre d'applications est blanche.

- La couleur d'arrière-plan de la page est également définie sur `AppColor.darkBackGroundColor`.

- Les couleurs principales de l'application sont définies sur `AppColor.darkPrimaryColor`.

- Les couleurs du texte sont définies en fonction de la taille du texte (`labelSmall`, `labelMedium`, `labelLarge`) avec une couleur de police blanche.

- Les couleurs des cartes et des canevas sont définies sur `AppColor.darkCardColor` et `AppColor.darkCanvasColor` respectivement.

- Les couleurs de la barre de navigation sont définies sur `AppColor.darkNavBackGroundColor` pour l'arrière-plan et sur `AppColor.darkIndicatorColor` pour la couleur de l'indicateur. La police est blanche.

En utilisant ces deux thèmes, l’application peut s'adapter automatiquement aux préférences de luminosité de l'utilisateur, offrant ainsi une expérience cohérente et agréable dans différents environnements d'utilisation.

```dart
static ThemeData darkThemeData = ThemeData(
  useMaterial3: true,
  appBarTheme: AppBarTheme(
    backgroundColor: AppColor.darkBackGroundColor,
    titleTextStyle: const TextStyle(
      color: Colors.white,
      fontWeight: FontWeight.bold, // Titre de l'AppBar en gras
      fontSize: 20
    ), // TextStyle
  ),      // AppBarTheme
  scaffoldBackgroundColor: AppColor.darkBackGroundColor,
  primaryColor: AppColor.darkPrimaryColor,
  textTheme: TextTheme(
    labelSmall: TextStyle(color: AppColor.white, fontSize: 14),
    labelMedium: TextStyle(color: AppColor.white, fontSize: 18),
    labelLarge: TextStyle(color: AppColor.white, fontSize: 20),
  ), // TextTheme
  colorScheme: ThemeData().colorScheme.copyWith(
    secondary: const Color.fromARGB(255, 136, 145, 193),
    brightness: Brightness.dark,
  ),
  cardColor: AppColor.darkCardColor,
  canvasColor: AppColor.darkCanvasColor,
  navigationBarTheme: NavigationBarThemeData(
    backgroundColor: AppColor.darkNavBackGroundColor,
    shadowColor: AppColor.lightTextColor,
    indicatorColor: AppColor.darkIndicatorColor,
    labelTextStyle: MaterialStateProperty.all(const TextStyle(color: Colors.white)),
    iconTheme: MaterialStateProperty.all(
      IconThemeData(color: AppColor.white),
    ),
  ), // NavigationBarThemeData
); // ThemeData
class AppTheme {
  static ThemeData lightThemeData = ThemeData(
    useMaterial3: true,
    appBarTheme: AppBarTheme(
      backgroundColor: AppColor.lightBackGroundColor,
      titleTextStyle: const TextStyle(
        color: Colors.black,
        fontWeight: FontWeight.bold, // Titre de l'AppBar en gras
        fontSize: 20
      ), // TextStyle
    ),      // AppBarTheme
    scaffoldBackgroundColor: AppColor.lightBackGroundColor,
    primaryColor: AppColor.lightPrimaryColor,
    textTheme: TextTheme(
      labelSmall: TextStyle(color: AppColor.black, fontSize: 14),
      labelMedium: TextStyle(color: AppColor.black, fontSize: 18),
      labelLarge: TextStyle(color: AppColor.black, fontSize: 20),
      //bodyLarge:  TextStyle(color: Colors.blue, fontSize: 30),
      //bodyMedium:  TextStyle(color: Colors.blue, fontSize: 25),
      //bodySmall:  TextStyle(color: Colors.blue, fontSize: 20),
    ), // TextTheme
    cardColor: AppColor.lightCardColor,
    canvasColor: AppColor.lightCanvasColor,
    navigationBarTheme: NavigationBarThemeData(
      backgroundColor: AppColor.lightNavBackGroundColor,
      shadowColor: AppColor.darkTextColor,
      indicatorColor: AppColor.lightIndicatorColor,
      labelTextStyle: MaterialStateProperty.all(const TextStyle(color: Colors.black)),
      iconTheme: MaterialStateProperty.all(
        IconThemeData(color: AppColor.black),
      ),
    ), // NavigationBarThemeData
  ); // ThemeData
```

## VII. Conclusion

Dans ce document, nous avons exploré différents aspects de l'application "Emaya" développée pour la ferme "Maya". Voici une conclusion générale basée sur les différentes fonctionnalités et composants discutés :

L'application "Emaya" offre une expérience complète aux utilisateurs, en mettant en avant la variété de produits biologiques de haute qualité de la ferme "Maya". Grâce à une interface utilisateur conviviale et intuitive, les clients peuvent facilement découvrir, explorer et acheter des produits, qu'ils soient en promotion ou non.

La barre de navigation bien conçue permet une navigation fluide entre les différentes fonctionnalités de l'application, notamment la page d'accueil, le panier, la page utilisateur et d'autres.

La page d'accueil propose une présentation attrayante des produits en promotion, des promotions spéciales, ainsi qu'une sélection de produits disponibles. Les widgets comme ProduitPromotionWidget offrent une représentation visuelle claire des produits, tandis que la page ProduitVentePage permet aux utilisateurs d'explorer une gamme plus large de produits en vente.

La fonctionnalité de panier offre aux utilisateurs la possibilité de gérer facilement leurs achats, avec la possibilité d'ajouter, de supprimer des articles et de visualiser le total de leurs achats.

La section utilisateur permet aux clients de gérer leur profil, de consulter leurs commandes passées et d'accéder à d'autres fonctionnalités importantes, le tout dans une interface bien organisée.

De plus, l'application prend en charge le mode jour/nuit, offrant une expérience visuelle adaptée aux préférences de luminosité de l'utilisateur, ce qui contribue à une utilisation confortable dans différentes conditions d'éclairage.

En conclusion, l'application "Emaya" offre une plateforme numérique complète pour la ferme "Maya", combinant une présentation attrayante des produits avec une navigation fluide et des fonctionnalités essentielles pour une expérience utilisateur optimale.