Modélisation des données :

1. **Diagramme de classes simplifié :**
```plaintext
Utilisateur
- id (PK)
- email
- mot_de_passe
- role (commercial/admin)
- created_at

Médicament
- id (PK)
- nom_scientifique
- nom_commercial
- description
- dosage
- prix
- stock

Commande
- id (PK)
- utilisateur_id (FK)
- statut (en attente/validée/etc.)
- date_creation
- date_modification

DetailCommande
- id (PK)
- commande_id (FK)
- medicament_id (FK)
- quantite

Notification
- id (PK)
- utilisateur_id (FK)
- message
- type (nouveau_produit/statut_commande)
- date
```

2. **Relations :**
- Un Utilisateur peut avoir plusieurs Commandes
- Une Commande contient plusieurs DetailCommande
- Un DetailCommande référence un Médicament

Code de démarrage :

1. **Backend (Django) :**
```python
# models.py
from django.db import models

class Utilisateur(models.Model):
    email = models.EmailField(unique=True)
    password = models.CharField(max_length=128)
    role = models.CharField(max_length=20)

class Medicament(models.Model):
    nom_scientifique = models.CharField(max_length=100)
    nom_commercial = models.CharField(max_length=100)
    prix = models.DecimalField(max_digits=10, decimal_places=2)
    stock = models.IntegerField()

class Commande(models.Model):
    STATUT_CHOICES = [
        ('attente', 'En attente'),
        ('validee', 'Validée'),
        ('livree', 'Livrée')
    ]
    utilisateur = models.ForeignKey(Utilisateur, on_delete=models.CASCADE)
    statut = models.CharField(max_length=20, choices=STATUT_CHOICES)
    date_creation = models.DateTimeField(auto_now_add=True)

class DetailCommande(models.Model):
    commande = models.ForeignKey(Commande, on_delete=models.CASCADE)
    medicament = models.ForeignKey(Medicament, on_delete=models.CASCADE)
    quantite = models.IntegerField()
```

2. **Frontend Mobile (Flutter) :**
```dart
// Modèle Medicament
class Medicament {
  final String id;
  final String nomScientifique;
  final String nomCommercial;
  final double prix;

  Medicament({
    required this.id,
    required this.nomScientifique,
    required this.nomCommercial,
    required this.prix,
  });
}
```

```
// Service d'authentification
class AuthService {
  Future<void> login(String email, String password) async {
    // Implémentation avec Dio
    final response = await Dio().post(
      'https://api.example.com/login',
      data: {'email': email, 'password': password},
    );
    // Gestion du token JWT
  }
}

// Exemple de widget de recherche
class SearchScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: TextField(
          decoration: InputDecoration(hintText: 'Rechercher médicament...'),
          onChanged: (value) => _searchMedicaments(value),
        ),
      ),
      body: FutureBuilder<List<Medicament>>(
        future: _loadMedicaments(),
        builder: (context, snapshot) {
          // Afficher la liste des résultats
        },
      ),
    );
  }
}
```

3. **API Endpoints (Django REST Framework) :**
```python
# views.py
from rest_framework import viewsets
from .models import Medicament
from .serializers import MedicamentSerializer

class MedicamentViewSet(viewsets.ModelViewSet):
    queryset = Medicament.objects.all()
    serializer_class = MedicamentSerializer

# urls.py
from django.urls import include, path
```

```
from rest_framework import routers

router = routers.DefaultRouter()
router.register(r'medicaments', MedicamentViewSet)

urlpatterns = [
    path('api/', include(router.urls)),
]
```

Recommandations pour commencer :
1. Initialiser le projet Django avec `django-admin startproject backend`
2. Créer une app Flask pour l'API
3. Configurer la base de données PostgreSQL dans settings.py
4. Initialiser le projet Flutter avec `flutter create mobile_app`
5. Ajouter les packages nécessaires dans pubspec.yaml :
```yaml
dependencies:
  flutter_bloc: ^8.1.3
  dio: ^5.3.2
  shared_preferences: ^2.2.2
```

Cette base couvre l'architecture essentielle pour démarrer le développement des fonctionnalités principales. Les prochaines étapes consisteraient à implémenter les systèmes d'authentification JWT, le panier de commandes, et les notifications push.