Chabha
Iddir

Documentation technique

AP4 : GSB-médecins

Cette application mobile à était réaliser avec le Framework Spring, qui représente un ensemble d'outils conçus pour faciliter le processus de développement d'application web ici il est utilisé comme API qui est une interface logicielle qui permet de « connecter » un logiciel ou un service à un autre logiciel ou service afin d'échanger des données et des fonctionnalités. Il se connecte à l'application GSB-Médecins fait avec Flutter qui est un Framework de développement d'applications mobiles open source de Google.

On retrouve plus bas la structure des données ainsi que le diagramme représentant le cas général d'utilisation du site.

Modèle Logique de données :

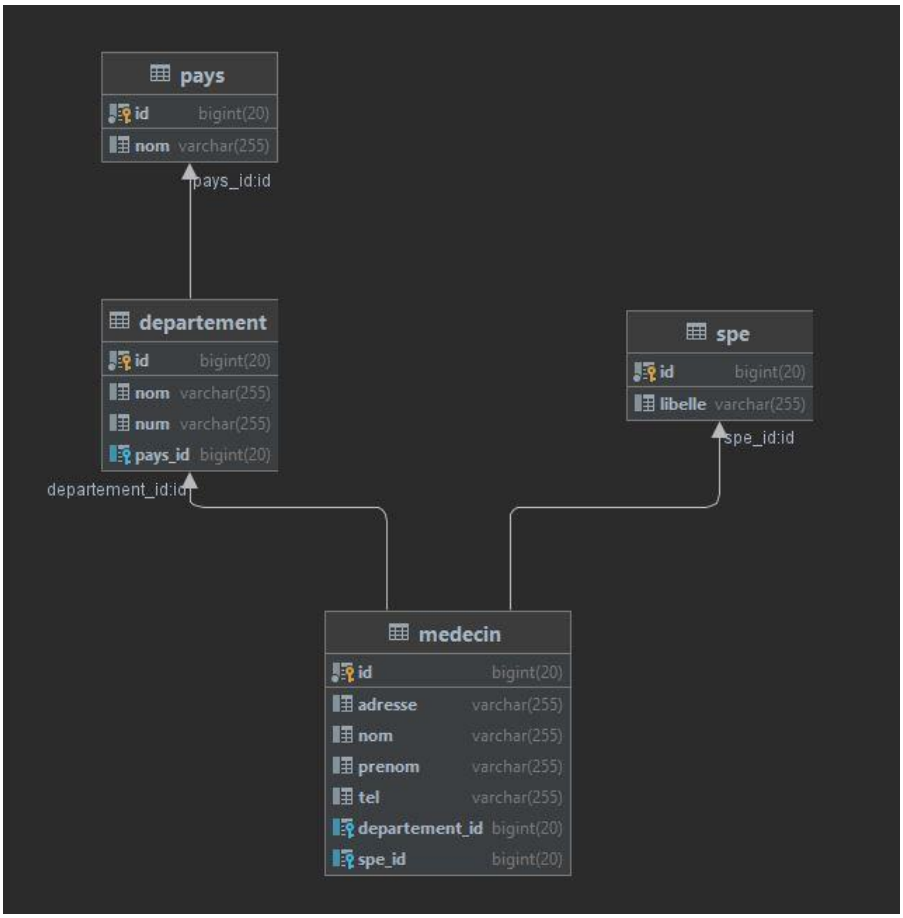
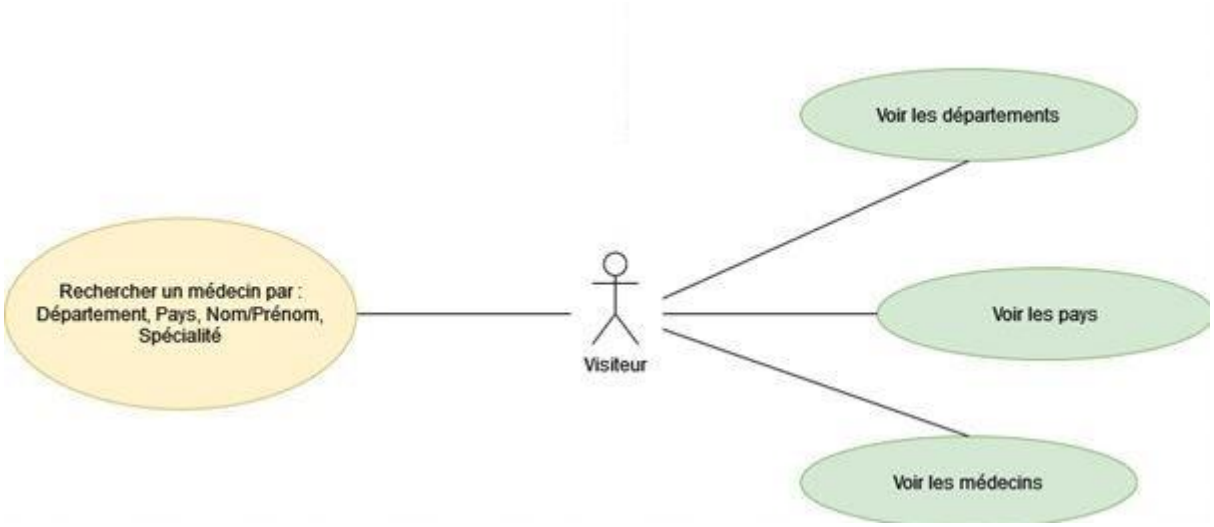


Diagramme des cas d'utilisations :



Spring en détails :

Le framework spring fait le lien entre BDD et Poo et transforme une table en un objet facilement manipulable via ses attributs. L'intérêt principal est d'éviter beaucoup de code très similaire. Pour ce projet Spring se charge de l'API REST (style architectural qui permet aux logiciels de communiquer entre eux sur un réseau ou sur un même appareil) pour l'application mobile.

Avantages

- Réduction du code à créer
- Homogénéité du code objet
- Accélération du temps de développement

Inconvénients

- Problèmes de mise en place lorsque la base de données n'est pas faite dans les règles de l'art
- Ne permet pas de gérer les requêtes un peu complexes (jointures, groupements.

Structure de l'application:

Les controllers :

Les controllers appelle les repository pour charger et enregistrer les données. Des annotations sont utilisées pour définir les routes. De plus, les méthodes définies acceptent en paramètre des objets qui seront injectés par le Framework. Ceux-ci permettent de définir les attributs et sont envoyés au client.

Les entités :

Représente les objets métier de l'application la plupart représentent une table dans la base de données. On utilise des annotations Java pour configurer les entités. L'ORM (Hibernate) va se charger de générer la structure de la base de données et gérer l'accès aux données (à l'aide des Repository).

Repository :

Les repositories permettent l'accès aux données, ça va centraliser tout ce qui touche à la récupération de nos entités. Ils contiennent la logique métier. Ils font la liaison avec les controllers.

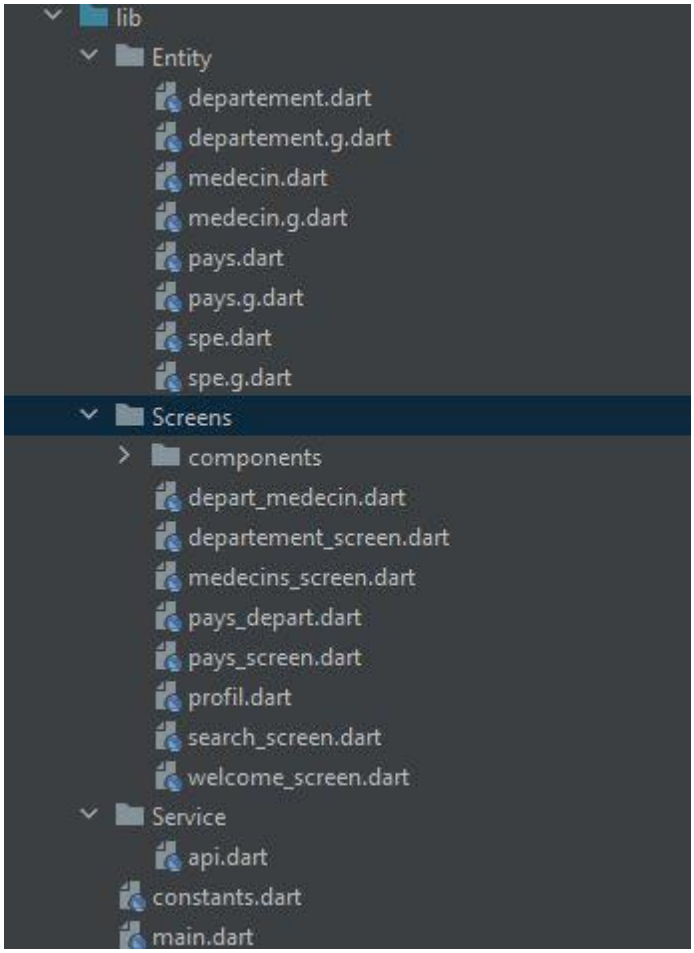
Projection:

Représente les interfaces de projection, plutôt que la classe d'entité, est utilisée comme type d'élément dans la collection renvoyée. Cela va nous permettre de récupérer une collection d'objet qui contient un autre objet sans tous ces attributs.

Application mobile :

L'application mobile est réalisée en Dart avec le framework Flutter.

Structure de l'application:



Description des classes:

Le dossiers Entity :

Regroupe les objets métiers, ils contiennent également les méthodes pour convertir depuis et vers le JSON. Ces méthodes sont générées automatiquement dans les fichiers .g.dart.

```dart
import 'package:gsb_medecins/Entity/medecin.dart';
import 'package:gsb_medecins/Entity/pays.dart';
import 'package:json_annotation/json_annotation.dart';

part 'departement.g.dart';

@JsonSerializable()
class Departement {
  int id;
  String nom;
  String num;
  List<Medecin>? medecins;
  Pays? pays;

  Departement({
    required this.id,
    required this.nom,
    required this.num,
    this.medecins,
    this.pays,
  });

  factory Departement.fromJson(Map<String, dynamic> json) =>
      _$DepartementFromJson(json);

  Map<String, dynamic> toJson() => _$DepartementToJson(this);
}
```

```dart
// GENERATED CODE - DO NOT MODIFY BY HAND

part of 'departement.dart';

// **************************************************************************
// JsonSerializableGenerator
// **************************************************************************

Departement _$DepartementFromJson(Map<String, dynamic> json) => Departement(
      id: json['id'] as int,
      nom: json['nom'] as String,
      num: json['num'] as String,
      medecins: (json['medecins'] as List<dynamic>?)
          ?.map((e) => Medecin.fromJson(e as Map<String, dynamic>))
          .toList(),
      pays: json['pays'] == null
          ? null
          : Pays.fromJson(json['pays'] as Map<String, dynamic>),
    );  // Departement

Map<String, dynamic> _$DepartementToJson(Departement instance) =>
    <String, dynamic>{
      'id': instance.id,
      'nom': instance.nom,
      'num': instance.num,
      'medecins': instance.medecins,
      'pays': instance.pays,
```

Service :

Il contient les classes qui permettent d'accéder aux données depuis l'API.

```dart
class Api {
  String baseUrl = "http://192.168.1.11:8080/api";

  Future<List<Medecin>> getMedecins() async {
    var response = await http.get(Uri.parse(baseUrl + "/medecins/"));
    var jsonData = jsonDecode(utf8.decode(response.bodyBytes)) as List;
    List<Medecin> medecins = [];

    for (var m in jsonData) {
      Medecin medecin = Medecin.fromJson(m);
      medecins.add(medecin);
    }
    print(medecins.length);
    return medecins;
  }

  Future<List<Departement>> getDepartements() async {
    var response = await http.get(Uri.parse(baseUrl + "/departements/"));
    var jsonData = jsonDecode(utf8.decode(response.bodyBytes)) as List;
    List<Departement> departements = [];

    for (var m in jsonData) {
      Departement departement = Departement.fromJson(m);
      departements.add(departement);
    }
    return departements;
  }
```

Screens :

Ces fichiers représentent les écrans de l'application. Chaque fichier est composé de deux classes, une s'occupant des états et l'autre qui contient le widget. Chaque classe widget doit être composée d'une méthode build() qui contient les différents widgets qui composent ce widget.

```dart
class PaysScreen extends StatefulWidget {
  const PaysScreen({Key? key}) : super(key: key);

  @override
  _PaysScreenState createState() => _PaysScreenState();
}

class _PaysScreenState extends State<PaysScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      // Add from here...
      appBar: AppBar(
          backgroundColor: kPrimaryColor,
          title: const Center(
            child: Text('GSB - Listes des médecins'),
          )),  // Center, AppBar

      body: Card(
        child: FutureBuilder<List<Pays>>(
          future: Api().getPays(),
          builder: (context, snapshot) {
            if (!snapshot.hasData) {
              return const Center(
                child: Text('Chargement...'),
              );  // Center
            } else {
              return ListView.builder(
                  itemCount: snapshot.data!.length,
                  itemBuilder: (context, i) {
                    return ListTile(
                        title: Text(snapshot.data![i].nom),
                        onTap: () {
                          Navigator.pushNamed(context, PaysDepart.routeName,
                              arguments: Api()
                                  .getDepartementsByPays(snapshot.data![i]));
                        });  // ListTile
                  });  // ListView.builder
            }
          },
```