*AP 4 – Projet « GSB Médecins »*

Documentation Technique

Cette documentation vise à présenter les axes majeurs de la méthode de conception du projet « GSB Médecins », application mobile permettant à des utilisateurs de gérer les médecins rattachés à la structure « Galaxy Swiss Bourdin »

**Sommaire:**

[Présentation du projet GSB Médecins 3](#_Toc98921228)

[Contexte 3](#_Toc98921229)

[Fonctionnalités détaillées 3](#_Toc98921230)

[Architecture et méthodes à mettre en œuvre 3](#_Toc98921231)

# Présentation du projet GSB Médecins

## Contexte

GSB Médecins est une application mobile qui cible le système d’exploitation « Android ».

L’application a pour but de permettre à ses utilisateurs d’effectuer des opérations de maintenance sur des données concernant les médecins rattachés à l’organisation « Galaxy Swiss Bourdin ».

Le visiteur doit, après une connexion via identifiant et mot de passe, avoir accès à de multiples fonctionnalités, à savoir des recherches avancées, la suppression, l’ajout et la modification des médecins, ainsi que du Pays et du département auquel ils appartiennent.

## Cahier des charges et attentes

L’application se décompose, à l’origine, en 3 entités (et tables) distinctes, afin de gérer le contexte :

Une image contenant texte, noir, métal, tableau de points

Description générée automatiquementUn médecin comporte un identifiant unique (clef primaire), une adresse complète, un nom, un prénom, une spécialité complémentaire optionnelle et un numéro de téléphone. Une clef étrangère lui est également passé, afin de le rattacher à un département précis.

Le département, tout comme le nom, ne contient qu’une clef primaire et un nom. Cependant, le département possède une clef étrangère qui le lie directement à un Pays.

Ainsi, Le médecin possède un département, qui est possédé par un Pays.

Les données devront être stockées sur un Système de Gestion de Bases de Données Relationnelles (SGBDR) PostgreSQL.

L’application devra lister les médecins, les pays ainsi que les départements.

L’utilisateur devra pouvoir effectuer des recherches de médecin par nom, par spécialité, ainsi que par département et pays d’appartenance.

La suppression, la modification et l’ajout d’une entité de chaque classe sera également à prévoir.

## Autres fonctionnalités

Afin de mener ces modifications à bien, des profils utilisateurs devront être mis en place.

Les utilisateurs se décomposent en deux enfants, les membres et les administrateurs.

Contrairement aux membres qui ne peuvent qu’effectuer des opérations dites « GET », à savoir de la collecte de données, les administrateurs auront la permission d’effectuer les requêtes de suppression, de modification et d’ajout, ainsi que d’inviter d’autres membres :

La création d’utilisateur est également prise en charge par l’application, mais nécessitera une « invitation ».

Un administrateur peut créer une invitation, à savoir un code d’une poignée de chiffres et lettres que devra utiliser un nouvel arrivant afin de pouvoir mettre en place son compte.

Un utilisateur ne pourra mettre en place qu’une seule invitation à la fois, et cette dernière comportera une limite d’utilisations. Une fois la limite atteinte, l’invitation deviendra inutilisable, et devra être supprimée ou renouvelée par l’utilisateur.

Les utilisateurs rejoignant l’application par ce procédé seront relégués au rang de membre. L’ajout de nouveaux administrateurs, étant une opération relativement délicate, devra être menée à bien par un administrateur de base de données, faisant l’opération manuellement au travers d’une fonction implémentée.

## Architecture et méthodes sélectionnées

Le projet sera décomposé en deux parties distinctes, à savoir :

* Une « API », permettant de fournir et traiter les données qu’elle collecte et fournit
* Un « Client lourd », permettant aux utilisateurs d’exploiter les données provenant de l’API, et de mener à bien leur mission de maintenance.

## Technologies utilisées

L’API sera conçue en Java, un langage orienté objet, et ce au travers du cadriciel « Spring Boot », simplifiant grandement la mise en place d’API par rapport au langage natif.

L’application mobile sera, quant à elle, conçue en « Flutter ». Flutter est un cadriciel basé sur le langage objet Dart, et va compiler ce code en Kotlin, qui est le langage utilisé pour les applications mobiles visant Android.

L’API suivra une architecture « Controller-Service-Repository », qui servira à découper l’application efficacement, et à l’organiser de manière convenable.

L’API sera ainsi principalement décomposée en 4 dossiers :

Une image contenant texte

Description générée automatiquement

* « Entities » : dossier regroupant les classes objet de l’API. Spring Boot se base sur ces entités pour construire de lui-même le squelette de la base de données (tables, contraintes et jointures).

« Repositories » : Ce dossier contient l’ensemble des requêtes SQL de sélection et de suppression qui seront effectuées. Elles sont définies par des méthodes spécifiques à Spring Boot, et seront donc des fonctions Java qui seront traduites en SQL par le cadriciel.

Exemple :



Ces fonctions retournent chacune une liste de Médecins, et prennent des paramètres divers. Spring s’occupera de traduire le nom de la fonction en requête SQL, et de placer les paramètres comme convenu.

* « Services » : dossier prenant en charge la mise en œuvre des opérations POST et PUT, et définissant certaines contraintes à ces requêtes.
* « Controllers » : dossier permettant de « mapper » les requêtes, à savoir rattacher une action (GET, POST…) à un URL précis.

Une image contenant texte

Description générée automatiquement

Cette requête permet de récupérer les médecins par leur nom en fonction du paramètre qui est passé. Cette requête possède donc un Mapping qui devra être utilisé afin d’appeler cette fonction et de récupérer les informations depuis l’API.

L’API passera ces données au format JSON, format facilitant le transport et la manipulation de données depuis une API ou une application vers une autre.

L’application Flutter devra donc récupérer ses informations JSON et les convertir en objet Dart.

Une image contenant texte

Description générée automatiquement

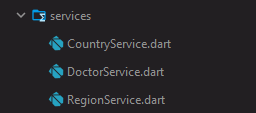
Un dossier d’entités, à la même manière que l’API, permet de définir l’ensemble des classes objets.

Les fichiers comprenant un « .g » sont dédiés à la transformation des données, que ça soit d’un fichier JSON à un objet, ou d’un objet à un JSON afin de la faire parvenir à l’API.

Une image contenant texte

Description générée automatiquement

L’application comportera un second dossier essentiel à la récupération des données : le dossier « service ».



Chaque fichier de service est rattaché à une classe, par souci de cohérence. Les fichiers contiennent des fonctions qui récupèrent les données depuis l’URL rattaché, et les convertissent en objets ou listes d’objets afin d’être exploitées au mieux par l’application.

Une image contenant texte

Description générée automatiquement