*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 **Erreur ! Signet non défini.**](#_Toc98921230)

[Architecture et méthodes à mettre en œuvre 6](#_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 une seule et unique catégorie, mais verront leur autorité définie par une liste de « rôles »

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.

Une image contenant texte

Description générée automatiquementUn utilisateur se verra affecté une liste de rôles. Un rôle est défini par un identifiant unique et un nom unique également.

Une image contenant texte

Description générée automatiquement

L’avantage principal justifiant l’utilisation de cette méthode est la capacité à cumuler les rôles. On peut ainsi attribuer à chaque requête un ou plusieurs rôles, donnant à l’utilisateur l’autorisation d’effectuer ces requêtes.

Ainsi, un administrateur cumulera à la fois le rôle de membre (permettant de récolter les données), mais aussi le rôle d’administrateur (permettant de modifier les données). Cette solution permet également l’ajout de rôles et de fonctionnalités rapidement, ce qui représente un fort avantage en cas d’évolution future de l’application.

Exemple d’instanciation d’utilisateurs, et des rôles qui leur seront attribués (ces derniers sont créés par défaut à chaque lancement d’application).

Une image contenant texte

Description générée automatiquement

On attribue, comme mentionné précédemment, le rôle d’administrateur ET de membre au second utilisateur. Ce dernier pourra effectuer l’ensemble des opérations qui sont définies dans le fichier de configuration de l’authentification Spring :

Une image contenant texte

Description générée automatiquement

L’autorisation se base sur un critère d’autorité. Ainsi, chaque requête, exceptés celles liées à la connexion (première ligne de code) nécessitera un rôle précis afin d’être appelée.

## 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.

## Architecture

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 6 dossiers :

Une image contenant texte, équipement électronique, capture d’écran

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.

* « Security » : Ce document contient le fichier de configuration dédié à la dépendance « Spring boot security », qui permet la mise en place d’authentification et d’autorisation lors de l’accès aux requêtes. Ce fichier contiendra l’ensemble des modalités permettant de définir ces critères et contraintes d’accès aux requêtes.

Une image contenant texte

Description générée automatiquement

On constate, dans cet exemple, qu’on a pu modifier l’URL rattaché aux requêtes de connexion. On a pu également gérer la méthode de gestion de session, et appliquer une fonction limitant les risques d’attaques cross-website (CSRF).

De plus amples paramètres peuvent être passés.

* « Filters » : Les fichiers de filtres prennent à leur charge les opérations de connexion. On définit au sein de fonctions sans corps (que l’on va compléter) la manière dont l’API doit gérer les tentatives de connexion, ainsi que les situations d’échecs et réussites qui suivront.

Une image contenant texte

Description générée automatiquementLes deux fonctions principales seront donc la fonction de tentative de connexion, et celle de succès.

Une image contenant texte

Description générée automatiquementLa fonction « attempt » va récupérer le contenu des champs « nom d’utilisateur » et « mot de passe ». En effet, l’authentification spring prend la forme d’un formulaire web au départ. Cependant, nous lui passeront, pour finir, un fichier JSON contenant ces deux paramètres dans des champs au nom identique :

Cette fonction permet ainsi d’illustrer la sécurité dans l’authentification et l’autorisation de notre API.

## Sécurité

L’autorisation fonctionne sur principe de « Tokens ». Un token contiendra l’ensemble des informations concernant un utilisateur, mais sous une forme illisible par l’homme, afin que ces informations n’apparaissent pas en clair.

Exemple avec un token attribué à l’utilisateur « bfinnegan » :

Une image contenant texte, capture d’écran, moniteur, écran

Description générée automatiquement

Le statut de la requête par nom d’utilisateur et mot de passe sera « 200 », ce qui signifie que la requête est validée. Un « access\_token » sera alors renvoyé, il contiendra l’ensemble des informations liées à l’utilisateur.

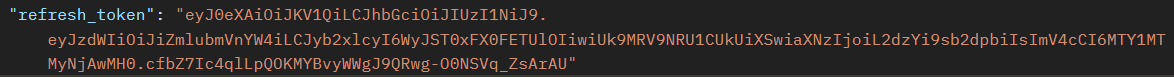
Une image contenant texte

Description générée automatiquement

On constate que le token renvoyé contient effectivement les informations nécessaires, à savoir le nom de l’utilisateur ciblé, l’ensemble de ses rôles, ainsi que d’autres informations complémentaires, comme le chemin duquel provient ce token.

Ce token d’accès possède une durée de vie limitée (environ 1h), il sera donc accompagné d’un token de « refresh », permettant de garder une trace de l’utilisateur et de renouveler sont token d’accès, qui sera nécessaire afin de procéder à une opération.

Si il est détecté que le token de l’utilisateur est expiré, alors une requête permettant de mettre à jour son token d’accès à partir du token de rafraîchissement sera alors effectuée, avant de repasser à la requête initialement prévue.



Ce token prend une forme similaire à l’autre, cependant, il possède une durée de vie bien supérieure à celle du token d’accès (environ 30 jours dans notre cas).

Une image contenant texte, capture d’écran, noir, écran

Description générée automatiquement

La requête « /token/refresh » sera appelée afin de remettre à jour le token d’accès. Cette requête prendra en paramètre une Autorisation, qui contiendra le token.

Le token devra être précédé de la mention « GSB\_WT  » avant le token afin que ce dernier soit considéré comme valable (Cette mention peut être modifié dans le code).

Tokens originels :

Une image contenant texte, intérieur, capture d’écran, fermer

Description générée automatiquement

Nouveaux tokens :

Une image contenant texte

Description générée automatiquement

Les tokens seront tous deux actualisés, prenant en contre de nouveaux paramètres gérés par Spring afin de conserver un certain anonymat des données.

Si le token de rafraîchissement est expiré, alors un message d’erreur ainsi qu’un status 403 (interdit) sera renvoyé :

Une image contenant texte

Description générée automatiquement

Ce même message sera renvoyé si le token d’accès est expiré, nous appellerons alors à ce moment la méthode de rafraîchissement afin de mettre à jour ce token et de réessayer la requête.

Extrait de la fonction mettant à jour le token :

Une image contenant texte

Description générée automatiquement

A condition que le token soit présent, non nul et qu’il commence par la chaine de caractère mentionnée ultérieurement, un algorithme de chiffrement va rafraichir le token à partir des informations de l’utilisateur.

Ce token sera, finalement, précisé dans chaque requête afin que l’utilisateur et ses rôles soient identifiés, et que l’accès à la requête lui soit accordé.

La librairie « flutter\_session » permettra de stocker les tokens et autres informations de l’utilisateur lors de sa connexion, afin que ces informations soient fournies automatiquement lorsqu’il appelle une requête.

Exemples d’utilisation :

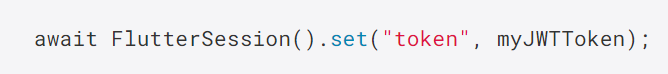
(Mise en place d’une session)

Solution 1 :

Une image contenant texte

Description générée automatiquement

Solution 2 :



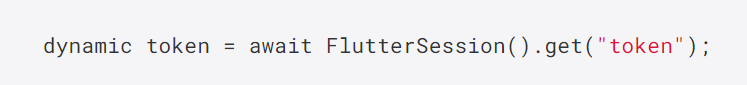
Les données seront stockées dans une classe dédiée à la session une fois récoltées.

Exemple de classe contenant les données :

Une image contenant texte

Description générée automatiquement

On pourra, pour finir, récupérer le token de manière simple :



## Récupération et mise en forme des données

L’ensemble des informations seront donc passées sous format JSON, permettant le transport d’objets/de données d’une application à une autre au travers d’un format populaire, simple d’utilisation et très standardisé.

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

Une image contenant texte

Description générée automatiquement

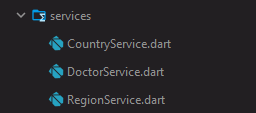
Un dossier d’entités, de la même manière que l’API, permet de définir l’ensemble des classes objets utilisées.

Les fichiers comprenant un « .g » sont dédiés à la conversion 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

D’autres vont effectuer des opérations (POST, PUT…)

Une image contenant texte

Description générée automatiquement

On utilise la fonction « ToJson » qui va convertir l’objet Dart en JSON. On envoi ensuite ce JSON dans le corps de la requête (body).