MédiLabo Solutions

We care for you

T2 Diabetes Detector

DOCUMENTATION

Fonctionnelle et technique

CAPPON SÉBASTIEN

Table des matières

1. Présentation du projet 3
1.1 Objectifs du projet 3
1.1.1 Micro-service : T2D2Authentication 3
1.1.2 Micro-service : T2D2Patient 4
1.1.3 Micro-service : T2D2Note 4
1.1.4 Micro-service : T2D2Diabetes 4
1.1.5 Edge micro-service : T2D2ServiceDiscovery 5
1.1.6 Edge micro-service : T2D2Gateway 5
1.1.7 Service Front 5
1.2 Éléments hors périmètre 6
2. Spécifications fonctionnelles 7
2.1 Micro-service Back : T2D2Authentication 7
2.2 Micro-service Back : T2D2Patient 8
2.3 Micro-service Back : T2D2Notes 10
2.4 Micro-service Back : T2D2Diabetes 11
2.5 Edges micro-service ServiceDiscovery et Gateway 11
3. Spécifications techniques 12
3.1 Diagramme de conception technique 12
3.2 Diagramme de conception technique : T2D2Authentication 13
3.3 Diagramme de conception technique : T2D2Patient 14
3.4 Diagramme de conception technique : T2D2Note 15
3.5 Diagramme de conception technique : T2D2Diabetes 16
3.6 Glossaire 17
3.7 Solutions techniques 18
3.7.1 Micro-service : T2D2Authentication 18
3.7.2 Micro-service : T2D2Patient 18
3.7.3 Micro-service : T2D2Note 18
3.7.4 Micro-service : T2D2Diabetes 19

3.7.5 Edge micro-service : T2D2ServiceDiscovery 19
3.7.6 Edge micro-service : T2D2Gateway 19
3.7.7 Service Front.....20

1. Présentation du projet

1.1 Objectifs du projet

T2 Diabetes Detector (T2D2) est une application facilitant la détection des risques de diabète de type 2 chez les patients afin de mettre en place des soins préventifs dans l'espoir de freiner l'apparition de cette maladie chronique.

Chaque **patient** y possède **un profil** que **son praticien alimente** avec les informations personnelles du patient. Chaque praticien peut **visualiser l'historique** du patient et **l'annoter**. Sur la base des informations personnelles du patient et de l'analyse lexicale des notes qui lui sont rattachées, le ***T2D2*** génère un **rapport de risque** allant de None *(aucun)* à EarlyOnset *(apparition précoce)*, en passant par Borderline *(risque limité)* et In Danger *(danger)*.

Au-delà du développement de l'application en micros-services, il a fallu mettre en place la dockerisation de celle-ci, grâce aux Dockerfiles présents dans chaque service, ainsi qu'au fichier docker-compose.yml situé à la racine du projet.

1.1.1 Micro-service : T2D2Authentication

T2D2Authentication est un micro-service né de la séparation du système d'authentification du micro-service ***T2D2Patient***. Dans un premier temps, le but était de déplacer l'authentification des utilisateurs dans l'edge micro-service ***T2D2Gateway***. Compte tenu de l'avancée du projet au moment où le gateway a été mis en place, ceci s'est avéré être une mauvaise idée. Le micro-service ***T2D2Authentication*** a toutefois été conservé par soucis de clarté. Chaque micro-service gère une fonctionnalité de l'application.

T2D2Authentication est rattaché à la base de données **mls_t2t2_practitioner** et se charge exclusivement de :

- Créer des comptes des Practitoners.
- Connecter le Practitioner au compte précédemment créé.
- Déconnecter le Practitioner de son compte.

Le routing de l'utilisateur au travers des différentes pages de l'application se fait dans le Front.

1.1.2 Micro-service : T2D2Patient

T2D2Patient est un micro-service qui octroie au Practitioner connecté la possibilité de gérer ses Patients. Il rattaché à la base de données **mls_t2d2_patient** et permet de :

- Créer des profils *(Person)* pour les nouveaux Patients.
- Lier des Persons déjà existantes dans la base de données au compte d'un Practitioner.
- Délier des Patients d'un Practitioner.
- Afficher la page de profil d'une Person
- Mettre à jour les informations d'une Person.

La relation entre les Practitioners et les Persons enregistrées dans la base de données sont représentées par la table de jointure **practitioner_person**. Plusieurs Practitioners peuvent avoir une même Person en tant que Patient.

La relation entre les Persons et leur(s) Addresses sont représentées par la table de jointure **person_address**. Plusieurs Persons peuvent habiter à la même Address et une Person peut avoir plusieurs Residences.

1.1.3 Micro-service : T2D2Note

T2D2Note est un micro-service octroyant au praticien la possibilité de gérer des Notes rédigées sur la fiche de chaque patient. Il est rattaché à la base de données non-relationnelle ***mls_t2d2_note*** et permet de :

- Créer des Notes.
- Modifier des Notes.
- Supprimer des Notes.

1.1.4 Micro-service : T2D2Diabetes

T2D2Diabetes est un micro-service octroyant au Practitioner la possibilité d'évaluer les Risks de diabète d'un Patient d'un simple clic. Le calcul des Risks se fait sur la base des informations du profil de la Person et de l'analyse des Notes qui lui sont rattachées. Afin d'affiner l'analyse, le Practitoner dispose d'un accès à un dictionnaire de TriggerTerms qu'il peut abonder.

T2D2Diabetes est rattaché à base de données **mls_t2d2_diabetes** et permet de :

- Rajouter des TriggerTerms au dictionnaire.
- Supprimer des TriggerTerms du dictionnaire.
- Calculer les Risks de diabètes chez une Person.

1.1.5 Edge micro-service : T2D2ServiceDiscovery

T2D2ServiceDiscovery est un edge micro-service assurant la détection et le ré-adressage des autres micro-services de l'application. Grâce au service discovery, les autres micro-services ne sont plus ciblés par leur adresse IP mais par leur nom d'application, définit dans le fichier de configuration *application.properties*.

Le fonctionnement de ***T2D2ServiceDiscovery*** repose essentiellement sur son propre *application.properties*.

1.1.6 Edge micro-service : T2D2Gateway

T2D2Gateway est un edge micro-service routant les requêtes de service front vers les micro-services back. Un gateway a deux principales fonctions : le routing et la sécurité. À l'heure où j'écris ces lignes, le gateway ne sert que de routeur. Il masque les adresses des micro-service aux yeux du service front qui ne communique donc qu'avec le gateway.

T2D2Gateway, couplé à ***T2D2ServiceDiscovery***, facilite grandement la dockerisation du projet.

1.1.7 Service Front

Le front de ***T2D2*** a été développé avec **Angular**. Nous ne reviendrons donc pas sur ce dernier dans le cadre de ce projet. Ce choix est délibéré. Il est essentiellement lié à la volonté d'évoluer vers la stack supérieure (fullstack) dans un avenir proche (3-5 ans).

Toutefois, il est bon de noter que l'emploi de cette technologie et d'une approche aussi réactive que possible, notamment par le biais des **Observables**, a imposé certaines contraintes lors de la création des micro-services Back.

1.2 Éléments hors périmètre

Le développement de l'application en micro-services présentée ci-avant a amené, à la réalisation d'objectifs complémentaires non requis :

- **Mise en place du Service Discovery**
  Clairement, le **Service Discovery** ne faisait pas partie des micro-service edge demandés. Toutefois, il est simple à mettre en place et facilite énormément la dockerisation de l'application. En effet, par défaut, le **gateway** cible les services grâce à leur URL. Le *localhost* du conteneur **Docker** diffère de celui de la machine de dev. De même, l'IP local du conteneur varie et n'est pas *127.0.0.1*. De ce fait, il est nécessaire de réécrire toutes les URL du **gateway** si l'on ne met pas en place de **Service Discovery**. Grâce au SD, les MS sont ciblés par leur **application name** et ne doivent pas être réécrit dans *docker-compose.yml*

- **Dockerisation des bases de données**
  C'était optionnel. Toutefois, la dockerisation des BDD facilite grandement celle de l'application globale. Ceci évite, par la même occasion, des utilisateurs de l'application dockerisée et, implicitement, toute manipulation de leur part en amont. Toutefois, les BDD sont considérées comme **stateful**. Les conteneur docker tirent leur puissance du fait qu'ils sont **stateless**. On peut s'interroger sur la viabilité de la mise en place d'un système **stateful** dans un système **stateless**.

- **Seeding partiel des bases de données**
  À l'origine, je comptais seeder la totalité des données de mon application locale. Toutefois, suite à des difficultés rencontrées pour seeder la BDD **MongoDB**, j'ai opté pour un seed partiel des BDD **MySQL** uniquement, en créant la structure des BDD et des tables depuis un script SQL, ainsi qu'en injectant l'utilisateur de la BDD utilisé par l'application.

- **Réalisation d'une documentation technique et fonctionnelle.**
  C'est le dernier projet, autant qu'il soit le plus beau possible ! Et si vous ouvrez cette doc, ça vous aidera à mieux *(et rapidement)* comprendre ce vaste projet. C'est à ça que sert une doc, non 😉.

2. Spécifications fonctionnelles

2.1 Micro-service Back : T2D2Authentication

- **Retourner la liste des praticiens :**
  GET http://localhost:8081/authentication-ms/practitioners
  Affiche la liste des objets Practitioners stockés dans la BDD avec leurs id, lastname, firstname, email et password hashé.

- **Retourner le profil d'un praticien :**
  GET http://localhost:8081/authentication-ms/practitioners/[practitionerId]
  Affiche les attributs d'un objet Practitioner, désigné par son id en PathVariable, à savoir : id, lastname, firstname, email et password hashé.

- **Connecter un praticien à l'application :**
  POST http://localhost:8081/authentication-ms/login
  Récupère les informations d'un objet Practitioner sur la base du DTO *(email et du password non hashé)* passé dans le RequestBody, et les renvoie vers le front pour la création d'un token d'authentification.

- **Créer un praticien :**
  POST http://localhost:8081/authentication-ms/practitioner
  Ajoute un nouvel objet Practitioner à la table **practitioner** de la BDD **mls_t2d2_practitioner**, à partir de l'objet Practitioner passées en RequestBody.

- **Modifier un praticien :**
  PUT http://localhost:8081/authentication-ms/practitioners/[practitionerId]
  Modifie un ou plusieurs attributs d'un objet Practitioner, désigné par son id en pathVariable, à partir de l'objet Practitioner passé en RequestBody.

- **Supprimer le profil d'un praticien :**
  DELETE http://localhost:8081/authentication-ms/practitioners/[practitionerId]
  Supprime de la BDD un objet Practitioner, désigné par son id en PathVariable.

2.2 Micro-service Back : T2D2Patient

- **Retourner la liste des adresses :**
  GET http://localhost:8082/patient-ms/addresses
  Affiche la liste des objets Address, stockés dans la BDD, avec leurs id, number, wayType, wayName, postcode, city, country.

- **Retourner les attributs d'une adresse :**
  GET http://localhost:8082/patient-ms/addresses/[addressId]
  Affiche les attributs d'un objet Address, désigné par son id en PathVariable, à savoir : id, number, wayType, wayName, postcode, city, country.

- **Créer une adresse :**
  POST http://localhost:8082/patient-ms/address
  Ajoute un nouvel objet Address à la table **address** de la BDD **mls_t2d2_patient**, à partir de l'objet Address passées en RequestBody.

- **Modifier une adresse :**
  PUT http://localhost:8082/patient-ms/addresses/[addressId]
  Modifie un ou plusieurs attributs d'un objet Address, désigné par son id en pathVariable, à partir de l'objet Address passé en RequestBody.

- **Supprimer une adresse :**
  DELETE http://localhost:8082/patient-ms/addresses/[addressId]
  Supprime de la BDD un objet Address, désigné par son id en PathVariable.

- **Retourner la liste des personnes :**
  GET http://localhost:8082/patient-ms/persons
  Affiche la liste des objets Person, stockés dans la BDD, avec leurs id, gender, lastname, firstname, birthdate, phone, email.

- **Retourner le profil d'une personne :**
  GET http://localhost:8082/patient-ms/persons/[personId]
  Affiche les attributs d'un objet Person, désigné par son id en PathVariable, à savoir : id, gender, lastname, firstname, birthdate, phone, email.

- **Créer une personne :**
  POST http://localhost:8082/patient-ms/person

Ajoute un nouvel objet Person à la table **person** de la BDD **mls_t2d2_patient**, à partir de l'objet Person passées en RequestBody.

- **Modifier une personne :**
  PUT http://localhost:8082/patient-ms/persons/[personId]
  Modifie un ou plusieurs attributs d'un objet Person, désigné par son id en pathVariable, à partir de l'objet Person passé en RequestBody.

- **Supprimer une personne :**
  DELETE http://localhost:8082/patient-ms/persons/[personId]
  Supprime de la BDD un objet Person, désigné par son id en PathVariable.

- **Retourner la liste des résidences :**
  GET http://localhost:8082/patient-ms/residences/persons/[personId]/addresses
  Affiche la liste des objets Address, stockés dans la BDD, liés à un objet Person, désigné par son id en PathVariable, avec leurs id, number, wayType, wayName, postcode, city, country.

- **Retourner la liste des résidences :**
  GET http://localhost:8082/patient-ms/residences/persons/email/[personEmail]/addresses
  Affiche la liste des objets Address, stockés dans la BDD, liés à un objet Person, désigné par son email en PathVariable, avec leurs id, number, wayType, wayName, postcode, city, country.

- **Créer une résidence :**
  POST http://localhost:8082/patient-ms/residence
  Lie un objet Address à un objet Person dans la table **person_address** de la BDD **mls_t2d2_patient**, à partir de l'objet PersonAddress passé en RequestBody.

- **Supprimer une résidence :**
  DELETE http://localhost:8082/patient-ms/residences/persons/[personId]/addresses/[addressId]
  Supprime de la BDD une jointure entre un objet Address et un objet Person, sur la base des id passés en PathVariable.

- **Retourner la liste des patients :**
  GET http://localhost:8082/patient-ms/residences/persons/[personId]/addresses
  Affiche la liste des objets Person, stockés dans la BDD, liés à un objet Practitoner, désigné par son id en PathVariable, avec leurs id, gender, lastname, firstname,

birthdate, phone, email.

- **Retourner la liste des personnes n'étant pas patient d'un praticien :**
  GET http://localhost:8082/patient-ms/patients/practitioners/[practitionerId]/persons/not-patients
  Affiche la liste des objets Person, stockés dans la BDD, qui NE sont PAS liés à un objet Practitioner, désigné par son id en PathVariable, avec leurs id, number, wayType, wayName, postcode, city, country.

- **Créer un patient :**
  POST http://localhost:8082/patient-ms/patient
  Lie un objet Person à un objet Practitioner dans la table **practitioner_person** de la BDD **mls_t2d2_patient**, à partir de l'objet PractitionerPerson passé en RequestBody.

- **Supprimer un patient :**
  DELETE http://localhost:8082/patient-ms/patients/practitioners/[practitionerId]/persons/[personId]
  Supprime de la BDD une jointure entre un objet Practitioner et un objet Person, sur la base des id passés en PathVariable.

2.3 Micro-service Back : T2D2Notes

- **Retourner la liste des notes :**
  GET http://localhost:8083/note-ms/notes
  Affiche la liste des objets Notes stockés dans la BDD avec leurs id, personId, date, content.

- **Retourner la liste des notes pour une personne donnée :**
  GET http://localhost:8083/note-ms/notes/persons/[personId]
  Affiche la liste des objets Notes stockés dans la BDD, liés à un objet Person désigné par son id en PathVariable, avec leurs id, personId, date, content.

- **Créer une note :**
  POST http://localhost:8083/note-ms/note
  Ajoute un nouvel objet Note à la collection **note** de la BDD **mls_t2d2_note**, à partir de l'objet Note passées en RequestBody.

- **Modifier une note :**
  PUT http://localhost:8083/note-ms/notes/[noteId]

Modifie l'attribut content d'un objet Note, désigné par son id en pathVariable, à partir de l'objet Note passé en RequestBody.

- **Supprimer une note :**
  DELETE http://localhost:8083/note-ms/notes/[noteId]
  Supprime de la BDD un objet Note, désigné par son id en PathVariable.

2.4 Micro-service Back : T2D2Diabetes

- **Retourner la liste des termes déclencheurs :**
  GET http://localhost:8084/diabetes-ms/triggers
  Affiche la liste des objets TriggetTerm stockés dans la BDD avec leurs id, et term.

- **Ajouter un terme déclencheur :**
  POST http://localhost:8084/diabetes-ms/trigger
  Ajoute un nouvel objet TriggerTerm à la table **triggerterm** de la BDD **mls_t2d2_diabetes**, à partir de l'objet TriggerTerm passées en RequestBody.

- **Supprimer un terme déclencheur :**
  DELETE http://localhost:8084/diabetes-ms/triggers/[triggerTermId]
  Supprime de la BDD un objet TriggerTerm, désigné par son id en PathVariable.

- **Calculer les risques pour un patient :**
  POST http://localhost:8084/diabetes-ms/risk
  Calcule les risques de diabète pour une personne sur la base du *DTO (gender, birthdate, notes)* passé en RequestBody. Retourne un objet Risk exploitable par le front pour afficher les risques.

2.5 Edges micro-service ServiceDiscovery et Gateway

Les edges micro-services ne disposent d'aucun endpoint, hormis les actuators, dans le cadre de ce projet. Plus généralement, ils ne se composent, respectivement, que d'un fichier application.properties.

3. Spécifications techniques

3.1 Diagramme de conception technique

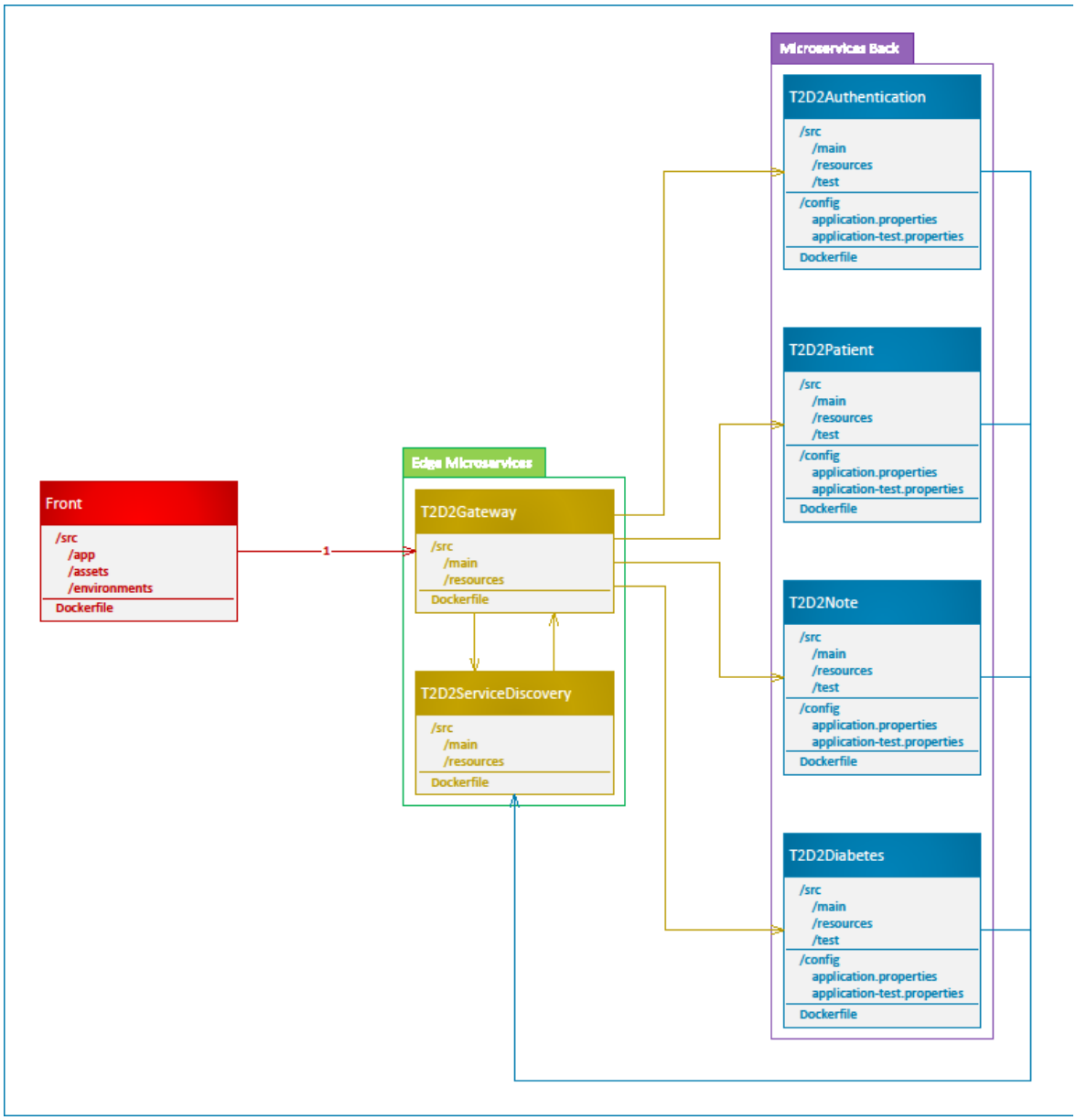

Figure 1 - Diagramme de fonctionnement global de l'application T2 Diabetes Detector

3.2 Diagramme de conception technique : T2D2Authentication

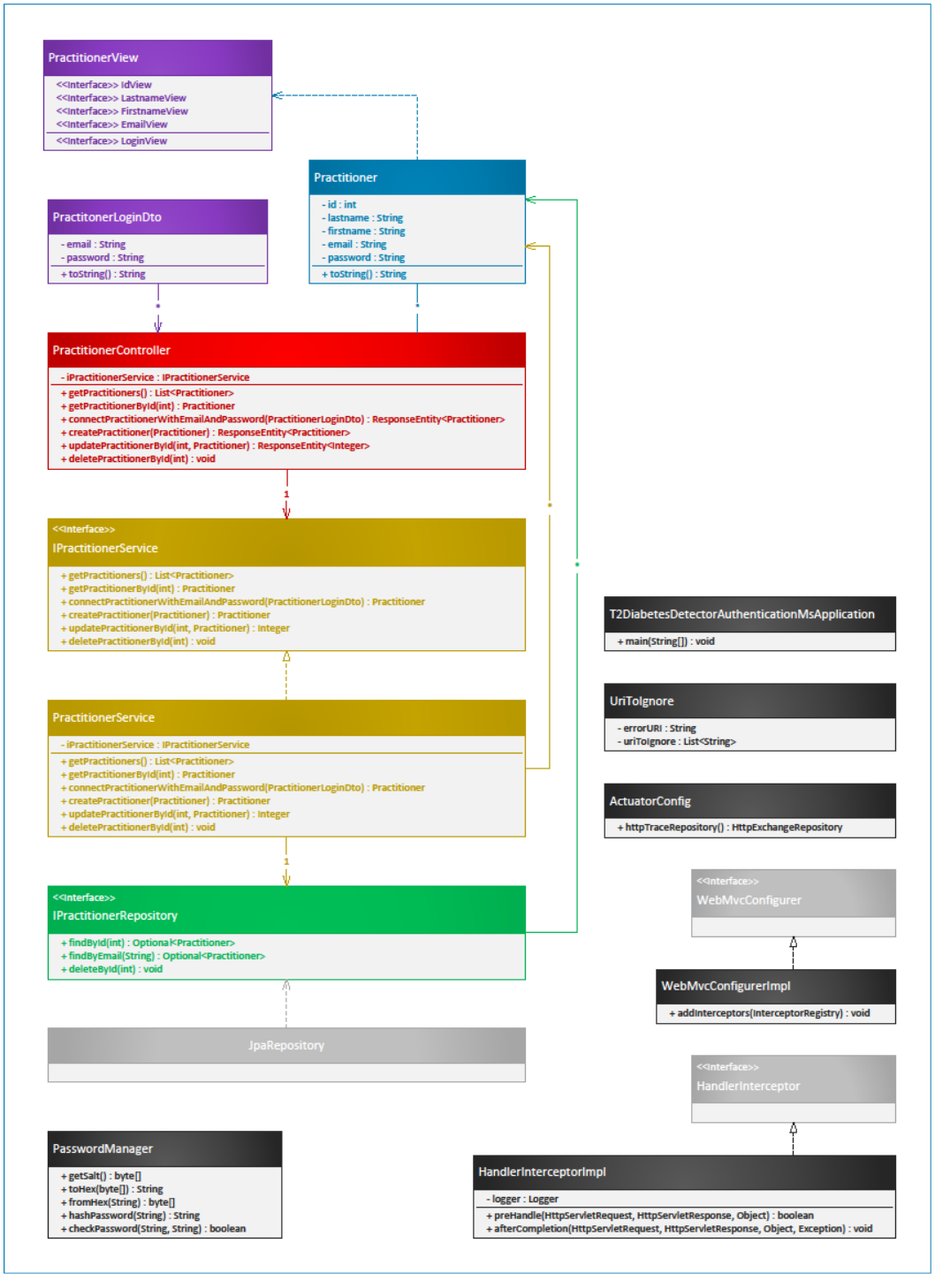

Figure 2 - Diagramme de classe du micro-service T2D2Authentication

3.3 Diagramme de conception technique : T2D2Patient

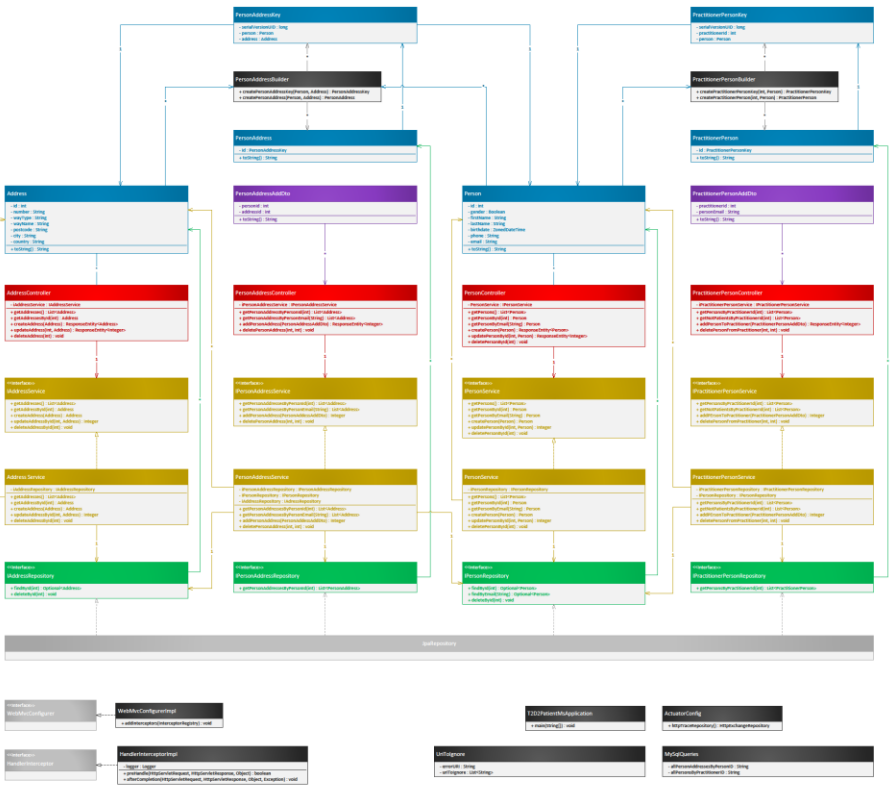

Figure 3 - Diagramme de classe du micro-service T2D2Patient

3.4 Diagramme de conception technique : T2D2Note

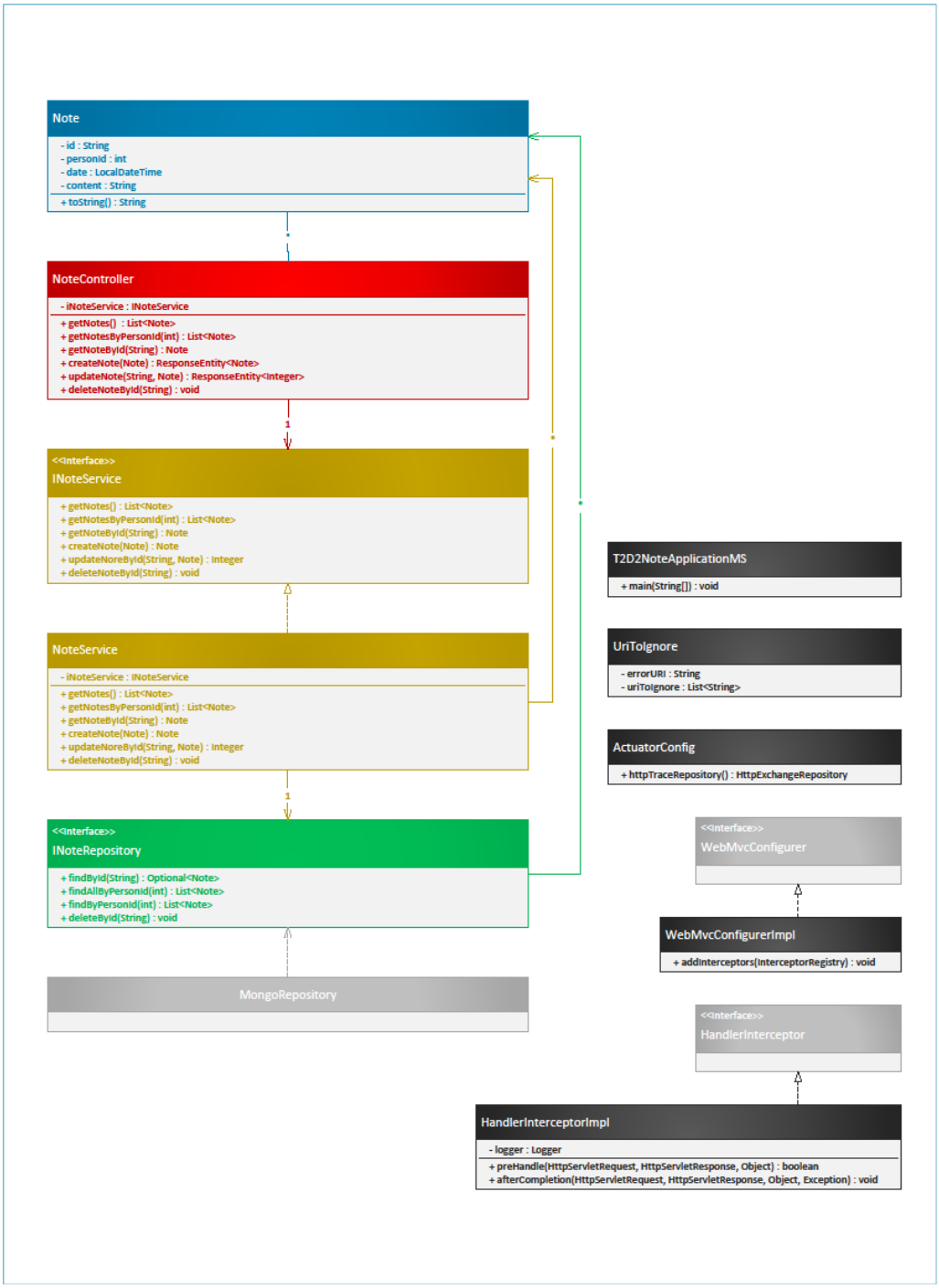

Figure 4 - Diagramme de classe du micro-service T2D2Note

3.5 Diagramme de conception technique : T2D2Diabetes

Figure 5 - Diagramme de classe du micro-service T2D2Diabetes

3.6 Glossaire

- **Practitioner :** Le praticien, premier utilisateur de l'application.
- **Person :** N'importe quel individu enregistré dans la base de données, qu'il soit ou non lié à un Practitioner ou plus.
- **Address :** N'importe quelle adresse enregistrée dans la base de données, à laquelle une Person ou plus peuvent résider.
- **Patient :** Une Person dès lors qu'elle est rattachée à un Practitioner. Patient désigne Person / Practitioner.
- **Residence :** Une Address dès lors qu'elle est rattachée à une Person. Residence désigne la relation Person / Address.
- **Note :** Court texte rattachée à une Person et rédigé par le Practitioner dont il est le Patient. Elle repose sur une terminologie précise, peut contenir un TriggerTerm ou plus et sert au calcul du Risk.
- **TriggerTerm :** Terme déclencheur pondéré dans le calcul du Risk. Une Note peut ne contenir aucun TriggerTerm, un, ou plus.
- **Risk :** Le niveau de risque d'un patient face au diabète. Se décline en 4 niveaux : None *(aucun)*, Borderline *(risque limité)*, Danger *(danger)*, EarlyOnset *(détection précoce)*.

3.7 Solutions techniques

3.7.1 Micro-service : T2D2Authentication

Le micro-service **T2D2Authentication** repose sur une architecture simple qui a fait ses preuves : une **API CRUD** suivant le pattern **MVC** et respectant autant que possible les principes **SOLID**.

Il fait appel à une BDD **MySQL** contenant une seule table, nécessaire au stockage des utilisateurs : les Practitioners.

La sécurité est gérée grâce à une classe **PasswordManager** permettant de **hasher** les mots de passe. Le routing de l'utilisateur à travers les pages l'applications est délégué au front.

À l'heure où cette documentation est rédigée, **Spring Security** n'est pas utilisé dans ce micro-service d'authentification.

3.7.2 Micro-service : T2D2Patient

Le micro-service **T2D2Patient** repose, lui aussi, sur cette même architecture simple qui a fait ses preuves : une **API CRUD** suivant le pattern **MVC** et respectant autant que possible les principes **SOLID**.

Il fait appel à une BDD **MySQL** contenant quatre tables, deux pour le stockage des Persons et des Addresses, et deux pour assurer les jointures entre Person et Address, ainsi que Practitioner et Person.

Bien qu'étant le plus grand des 4 micro-services back, **T2D2patient** est le plus sobre dans sa réalisation.

3.7.3 Micro-service : T2D2Note

Le micro-service **T2D2Note** repose, lui aussi, sur cette même architecture simple qui a fait ses preuves : une **API CRUD** suivant le pattern **MVC** et respectant autant que possible les principes **SOLID**.

Il fait appel à une BDD **MongoDB** contenant une unique collection regroupant les Notes des Practitioners pour chaque Person.

Le choix de **MongoDB** n'est motivé que par les prérequis de l'énoncé. Une BDD **MySQL** composée d'une table pour stocker les notes et une table de jointure pour les lier au Person aurait été un choix technique plus judicieux. Nous ne faisons ici aucune écriture de masse *(point fort de MongoDB)*. En revanche, nous effectuons de la sélection en quantité. De plus, les jointures auraient été les bienvenues dans le cas présent *(points forts de MySQL)*.

3.7.4 Micro-service : T2D2Diabetes

Le micro-service **T2D2Diabetes** repose, lui aussi, sur cette même architecture simple qui a fait ses preuves : une **API CRUD** suivant le pattern **MVC** et respectant autant que possible les principes **SOLID**.

Il fait appel à une BDD **MySQL** contenant une unique table pour le stockage des TriggerTerm.

Petite particularité, le micro-service dispose aussi d'un unique endpoint appelant l'algorithme de calcul du Risk.

3.7.5 Edge micro-service : T2D2ServiceDiscovery

Le micro-service **T2D2ServiceRecovery** se compose essentiellement d'un unique *application.properties*. Aucun code Java dans ce micro-service n'est rattaché à sa fonction première : renommer les addresses des services pour le **gateway**.

L'utilisation du seul *application.properties* est un choix délibéré. Les fichiers de configuration peuvent être externalisés et hébergés sur un repository ***GitHub*** en vue d'une mise à jour à la volée des configurations des micro-services formant l'application.

Malheureusement, cette solution, non requise, reposant **sur Spring Cloud Config**, n'a pas pu être implémentée, faute de temps.

3.7.6 Edge micro-service : T2D2Gateway

Le micro-service **T2D2Gateway** se compose essentiellement d'un unique *application.properties*. Aucun code Java dans ce micro-service n'est rattaché à sa

fonction première : router les requêtes du front vers les micro-services back afin d'en masquer les URLs.

L'utilisation du seul *application.properties* est un choix délibéré. Les fichiers de configuration peuvent être externalisés et hébergés sur un repository ***GitHub*** en vue d'une mise à jour à la volée des configurations des micro-services formant l'application.

Malheureusement, cette solution, non requise, reposant sur **Spring Cloud Config**, n'a pas pu être implémentée, faute de temps.

De même, l'autre point fort des **gateway** réside dans la sécurisation de l'accès aux micro-services. À l'heure où ces lignes sont écrites, cet aspect, non exigé dans l'énoncé, n'a pu être couvert, faute de temps. Il n'est pas exclu qu'il le soit d'ici la présentation du projet 😉

3.7.7 Service Front

Nous n'abordons le front que succinctement : Il repose sur **Angular**, un framework **TS**. Ce choix personnel a été effectué en accord avec une volonté d'évoluer vers le métier de développeur fullstack dans les années à venir.