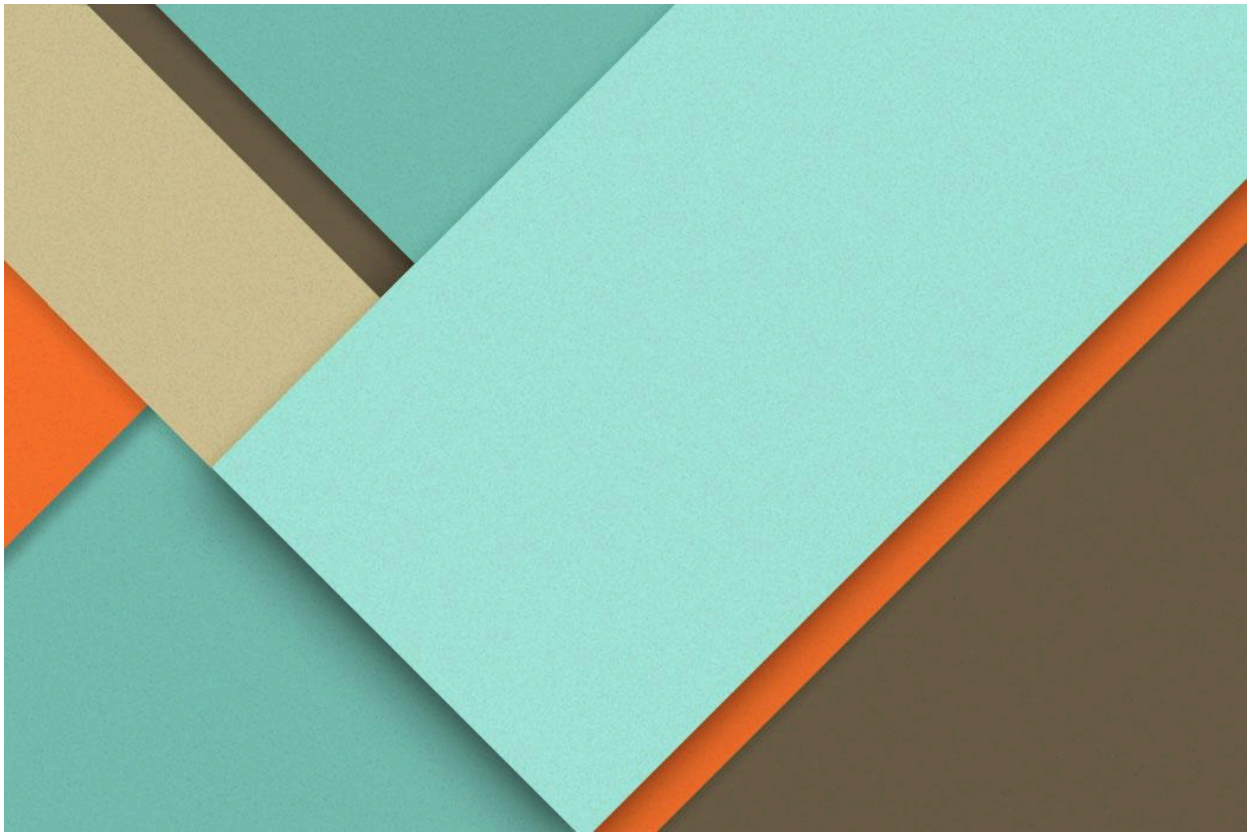


Documentation Fonctionnelle et Technique

22.08.2024

EDDARI
thery
thery.eddari@hotmail.fr

1. Présentation du projet

1.1 Objectifs du projet

Le projet vise à développer une application de gestion des patients pour le suivi du diabète, utilisant une architecture microservices. L'application permettra aux praticiens de visualiser, gérer et évaluer les informations relatives aux patients.

1.2 Hors du champ d'application

Ce projet n'inclut pas :

- La gestion de l'inscription ou des droits d'accès utilisateurs autres que l'authentification basique.
- L'intégration avec des systèmes externes non spécifiés.

1.3 Mesures du projet

Le succès sera mesuré par :

- La conformité des fonctionnalités aux besoins spécifiés.
- La fonctionnalité des microservices et leur intégration.
- La précision de l'évaluation du risque de diabète.
- La satisfaction du client concernant les livrables.

2. Spécifications fonctionnelles

2.1 Fonctionnalités

Gestion des Comptes utilisateurs

- Page de connexion : Permet aux utilisateurs de se connecter à l'application.
- Page de bord : Permet aux utilisateurs d'avoir un message de bienvenu (pour l'instant)
- Page de connexion : Permet aux utilisateurs de se déconnecter de l'application.

Gestion des comptes Clients

- Visualisation des Informations Personnelles : Permet aux utilisateurs de voir les informations des patients.
    - permettre le retour à la liste manuel
- Mise à jour des Informations Personnelles : Permet de mettre à jour les informations des patients.
    - permettre le retour automatique à la liste après mise à jour
    - permettre le retour manuel à la liste
- Ajout d'Informations Personnelles : Permet d'ajouter des informations aux dossiers des patients.
    - permettre le retour automatique à la liste après ajout d'information
    - permettre le retour manuel à la liste

Gestion des Notes Médicales

- Vue historique du Patient : Permet aux praticiens de voir l'historique des informations d'un patient.
    - permettre uniquement un retour manuel à la liste
- Ajout de Notes : Permet aux praticiens d'ajouter des notes au dossier du patient.
    - permettre uniquement un retour manuel à la liste

Évaluation du Risque de Diabète

- Génération de Rapport de Risque : Évalue et affiche le niveau de risque de diabète d'un patient en fonction des informations et notes médicales.

2.2 User Stories et Critères d'acceptation

Gestion des Comptes Utilisateur

Page de Connexion			
User Story 1	En tant qu'utilisateur, je veux me connecter pour accéder à l'application.		
Critère d'acceptance	Donnée Initial	Action	Conséquence
Positif	l'utilisateur est sur la page de login	l'utilisateur rentre le bon mot de passe et nom d'utilisateur et je clique sur "se connecter"	L'utilisateur est authentifié et redirigé vers la page d'accueil.
Négatif		l'utilisateur rentre le mauvais mot de passe	Message d'erreur "identifiant incorrect" au dessus du formulaire

Page de Bord			
User Story 2	En tant qu'utilisateur, je veux pouvoir être sur ma page d'accueille		
Critère d'acceptance	Donnée Initial	Action	Conséquence
Positif	l'utilisateur est sur sa page d'accueille	ne fais rien	un message de bienvenue reste sur la page d'accueille "Bienvenue sur votre espace"
Négatif		ne fais rien	un message de bienvenue reste sur la page d'accueille "Bienvenue sur votre espace"

Page de Déconnexion			
User Story 3	En tant qu'utilisateur, je veux pouvoir me déconnecter de l'application.		
Critère d'acceptance	Donnée Initial	Action	Conséquence
Positif	je suis connecter a mon compte sur n'importe quelle page et je veux me déconnecter	je clique sur le bouton "me deconnecter"	Je suis redirigé vers la page de connexion
Négatif		//	un message d'erreur "déconnexion impossible, réessayer" apparaît

Gestion des comptes Clients

Visualisation des Informations client			
User Story 4	En tant qu'utilisateur, j'aimerais voir les informations personnelles de mes patients.		
Critère d'acceptance	Donnée Initial	Action	Conséquence
Positif	Je suis sur ma page "Liste Client" ou et je souhaite pouvoir voir les informations suivantes : prénom, nom, date de naissance, genre, adresse postale, numéro de téléphone de mon client	je clique sur la ligne qui correspond à mon client	je suis sur la page "Client Perso"Je peux voir les informations voulu
Négatif		je clique sur le mauvais client	je vois un autre client

Visualisation des Informations client			
User Story 5.a	En tant qu'utilisateur, j'aimerais retourner sur ma liste de patient après avoir consulter la fiche d'un patient		
Critère d'acceptance	Donnée Initial	Action	Conséquence
Positif	je suis sur la page "Client Perso" qui me permet de voir les informations personnelle du client	je clique sur le bouton "retour"	Je suis sur ma page "Liste Client" et rien n'a été enregistré, je suis redirigé sur ma page "Liste Client"
Négatif		je clique sur enregistrer	les informations sont enregistré même si je ne le voulais pas et je suis redirigé sur ma page "Liste Client"

Ajout d'Infos Personnelles			
User Story 5.b	En tant qu'utilisateur, j'aimerais ajouter des informations personnelles aux patients.		
Critère d'acceptance	Donnée Initial	Action	Conséquence
Positif	Je suis sur ma page"Client Perso" qui me permet de voir les informations personnel du client	j'ajoute une donnée sur un des champs non encore fournis et je clique sur "Enregistrer"	Les informations sont ajouté au au client, un message "information mise à jour" apparaît je reste sur la page "Client Perso"
Négatif		je ne remplis pas un champs obligatoire	un message "veuillez renseigner ce champ" apparaît sous le champ m'indiquant l'erreur et je reste sur la page "Client Perso"

Modification d'Infos Personnelles			
User Story 5.c	En tant qu'utilisateur, j'aimerais modifier des informations personnelles.		
Critère d'acceptance	Donnée Initial	Action	Conséquence
Positif	Je suis sur ma page"Client Perso" qui me permet de voir les informations personnel du client	je modifie une donnée sur un champs déjà fournis et je clique sur "Enregistrer"	un message "information mise à jour" apparaît je reste sur la page "Client Perso"
Négatif		je ne remplis pas un champs obligatoire	un message "veuillez renseigner ce champ" apparaît sous le champ m'indiquant l'erreur et je reste sur la page "Client Perso"

Gestion des Notes Médicales

Vue Historique du Patient			
User Story 6	En tant qu' utilisateur, je veux voir l'historique des informations de mon patient.		
Critère d'acceptance	Donnée Initial	Action	Conséquence
Positif	Je suis sur ma page "Client Perso et je souhaite voir l'historique des informations de mon client que j'ai rentré	je clique sur le bouton "Suivi Client"	je suis sur la page "suivi client"Je peux voir les informations voulu sous forme d'un tableau avec la date d'ajout et la note dont le formatage n' a pas été modifié
Négatif		je clique sur un client qui n'a pas de note	le tableau est vide

Vue Historique du Patient			
User Story 7.a	En tant qu'utilisateur, j'aimerais retourner sur ma liste de patient après avoir consulter l'historique		
Critère d'acceptance	Donnée Initial	Action	Conséquence
Positif	je suis sur ma page "suivi client" et je souhaite retourner sur ma liste de client	je clique sur le bouton "retour"	Je suis sur ma page "Client Perso" et rien n'a été enregistré
Négatif		je clique sur enregistrer	les informations sont enregistré même si je ne le voulais pas et je ne suis pas redirigé

Vue Historique du Patient			
User Story 7.a.bis	En tant qu'utilisateur, j'aimerais retourner sur ma liste de patient après avoir consulter l'historique		
Critère d'acceptance	Donnée Initial	Action	Conséquence
Positif	je suis sur ma page "suivi client" et je souhaite retourner sur ma liste de client	je clique sur le bouton "Liste clients"	Je suis sur ma page "Liste Clients" et rien n'a été enregistré

Ajout de Notes			
User Story 7.b	En tant qu'utilisateur, je veux ajouter une note à l'historique du patient.		
Critère d'acceptance	Donnée Initial	Action	Conséquence
Positif	je suis sur ma page "suivi client" et je souhaite ajouter une note	je rentre mes notes dans la case de mon tableau qui est libre et je clique sur enregistrer	un message "suivis mis à jour" apparaît au-dessus du tableau et je constate que la ligne s'est ajoutée. je ne suis pas redirigé
Négatif		je rentre une note vide	un message apparaît "veuillez entrer une note". Je ne suis pas redirigé

Évaluation du Risque de Diabète

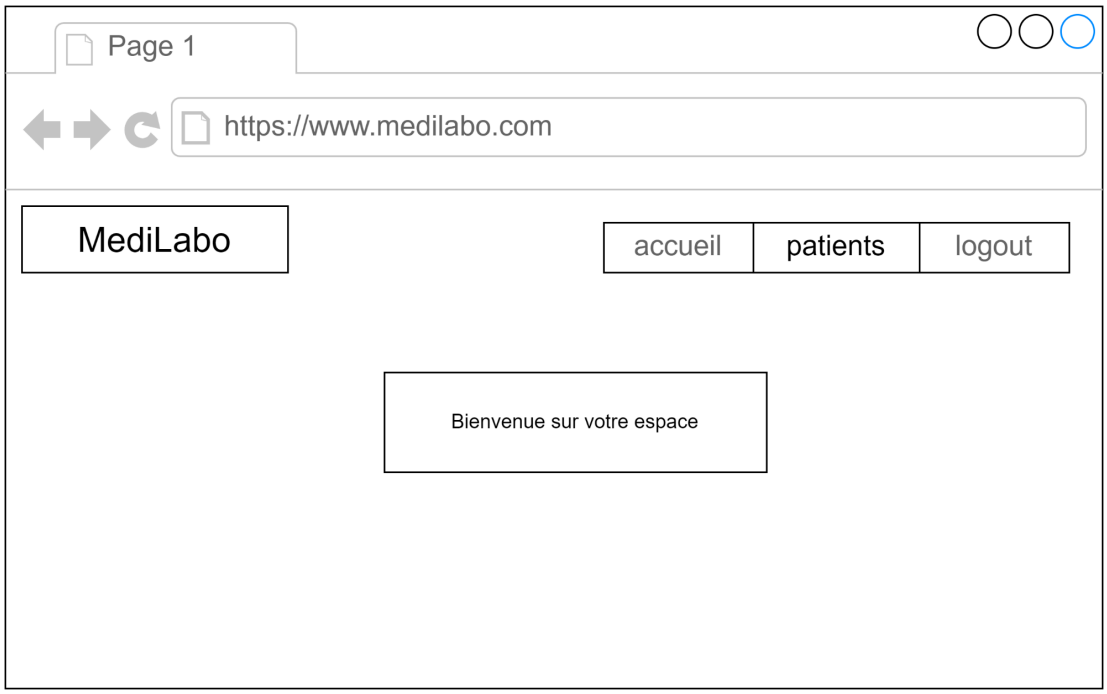
Risque Diabète			
User Story 7.c	En tant qu'utilisateur, je veux pouvoir consulter le risque de diabète pour un patient.		
Critère d'acceptance	Donnée Initial	Action	Conséquence
Positif	je suis sur ma page "suivi client" je souhaite connaître le risque de diabète pour un client	je regarde en a droite du tableau	je peux voir une case de couleur avec le risque présent pour le client
Négatif		un problème de connexion	un message apparaît "problème de connexion".

2.3 Wireframes

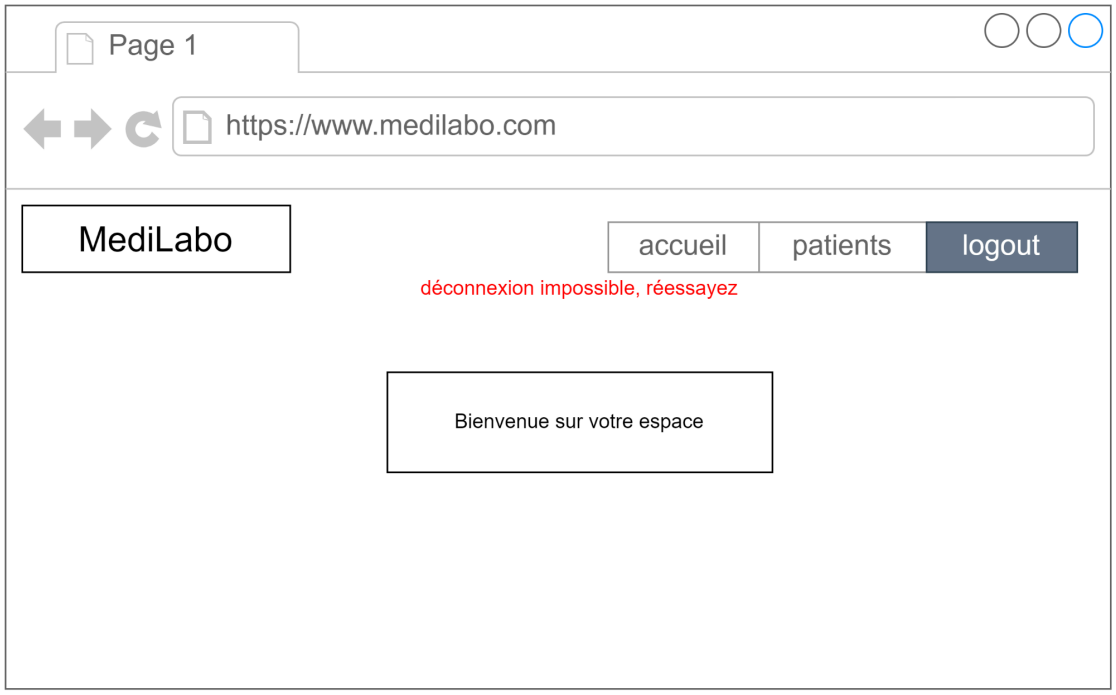
Gestion des Comptes Utilisateur

User stories 1

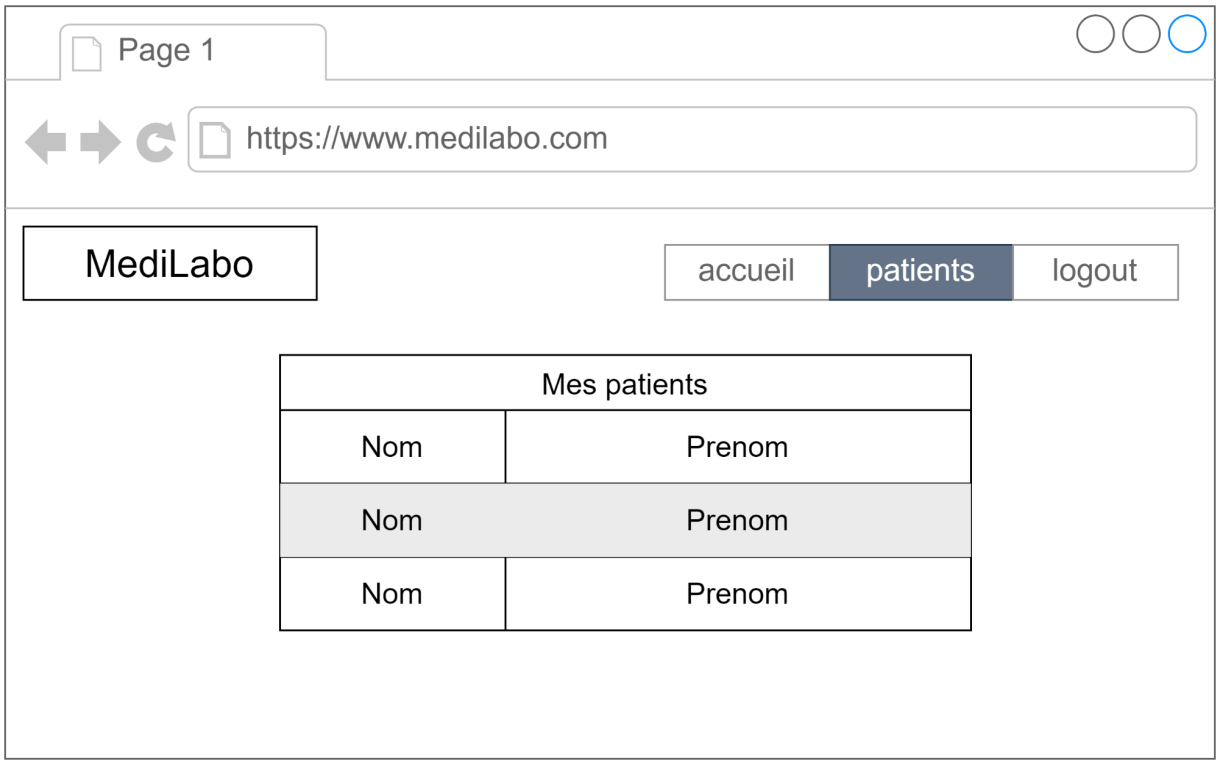
User stories 2



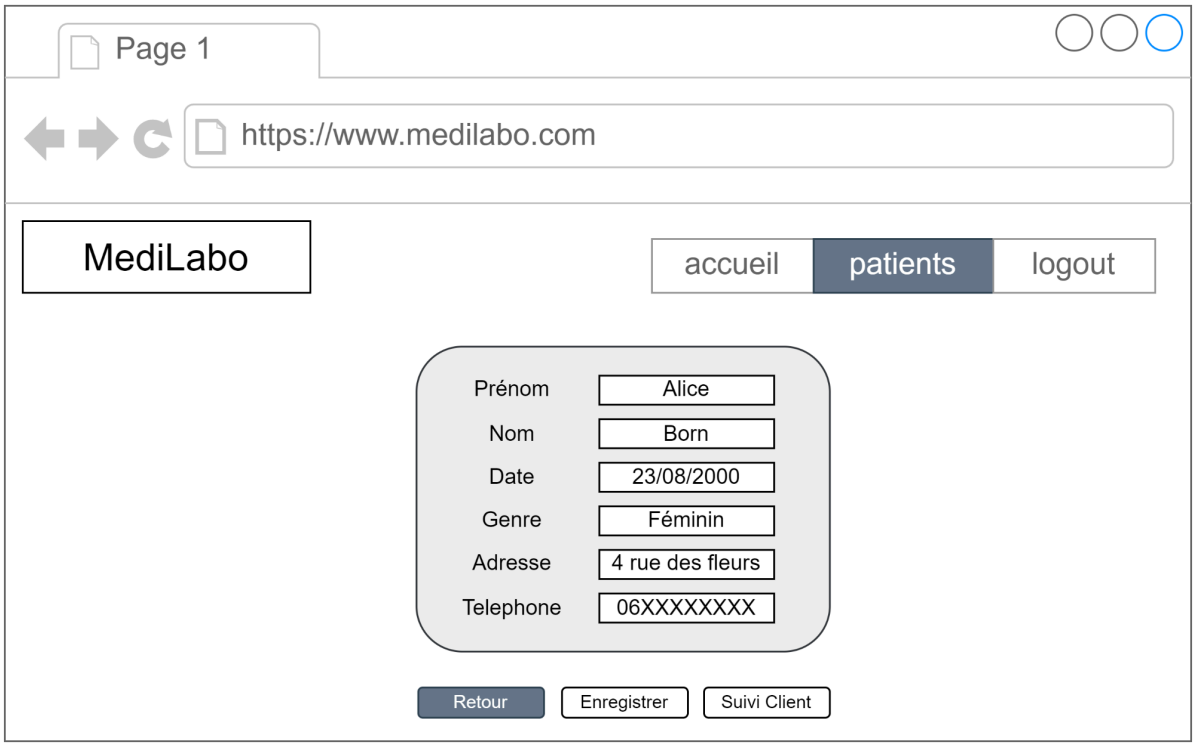
User stories 3

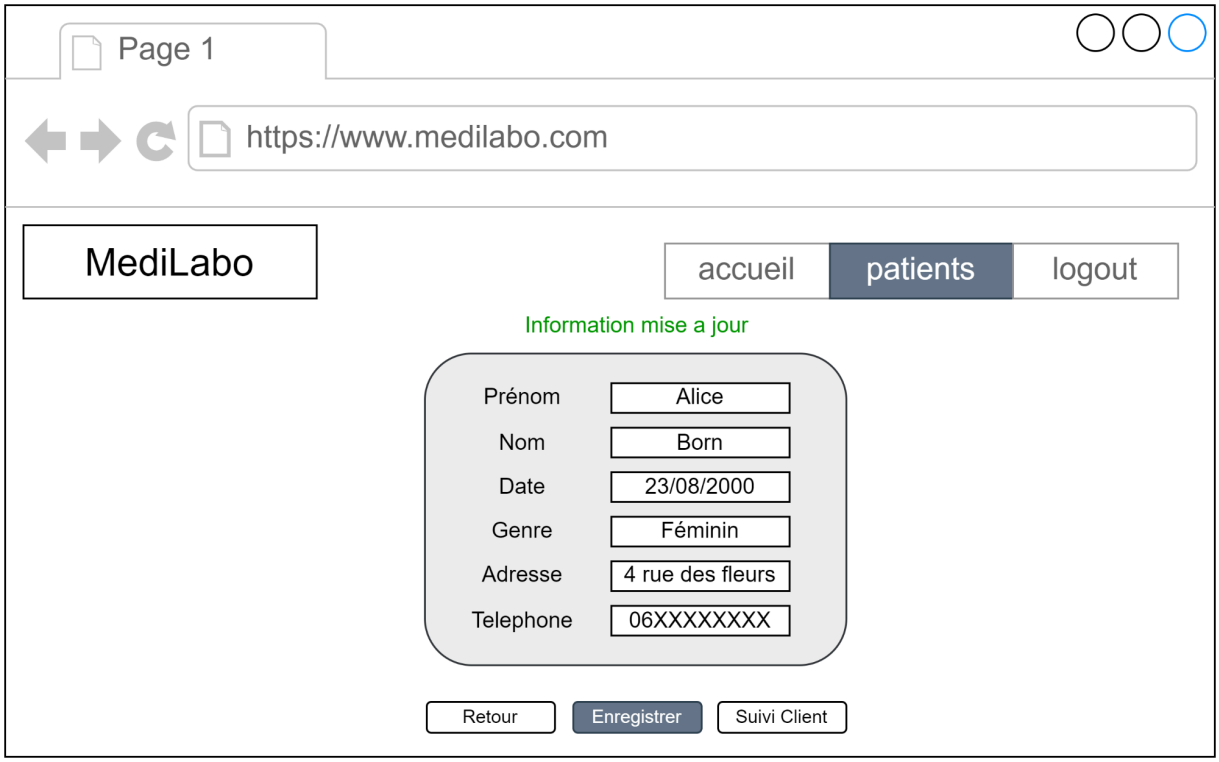


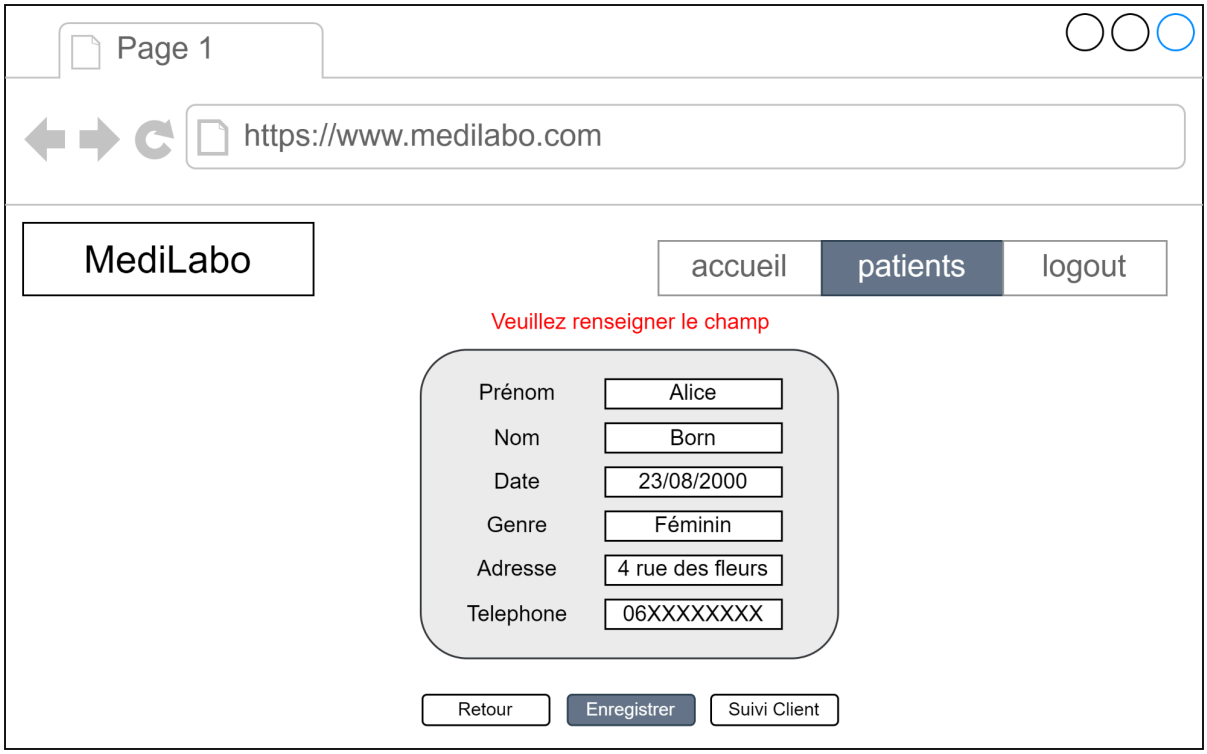
User stories 4



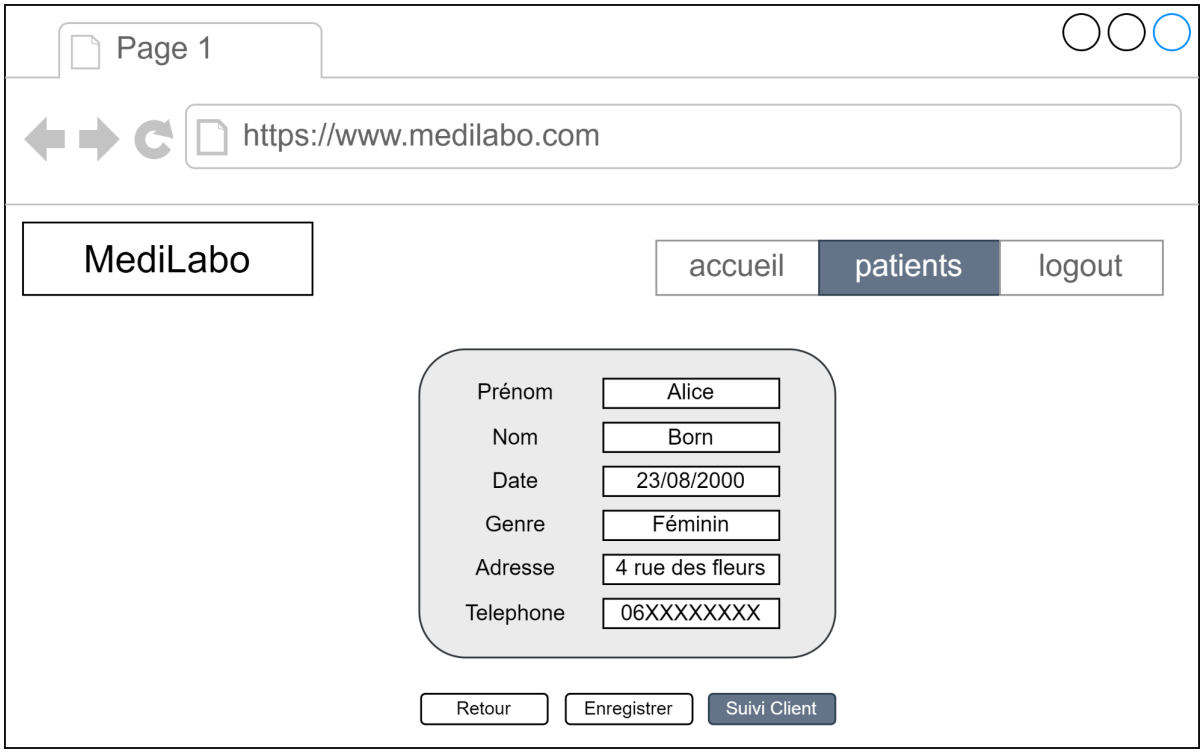
User stories 5



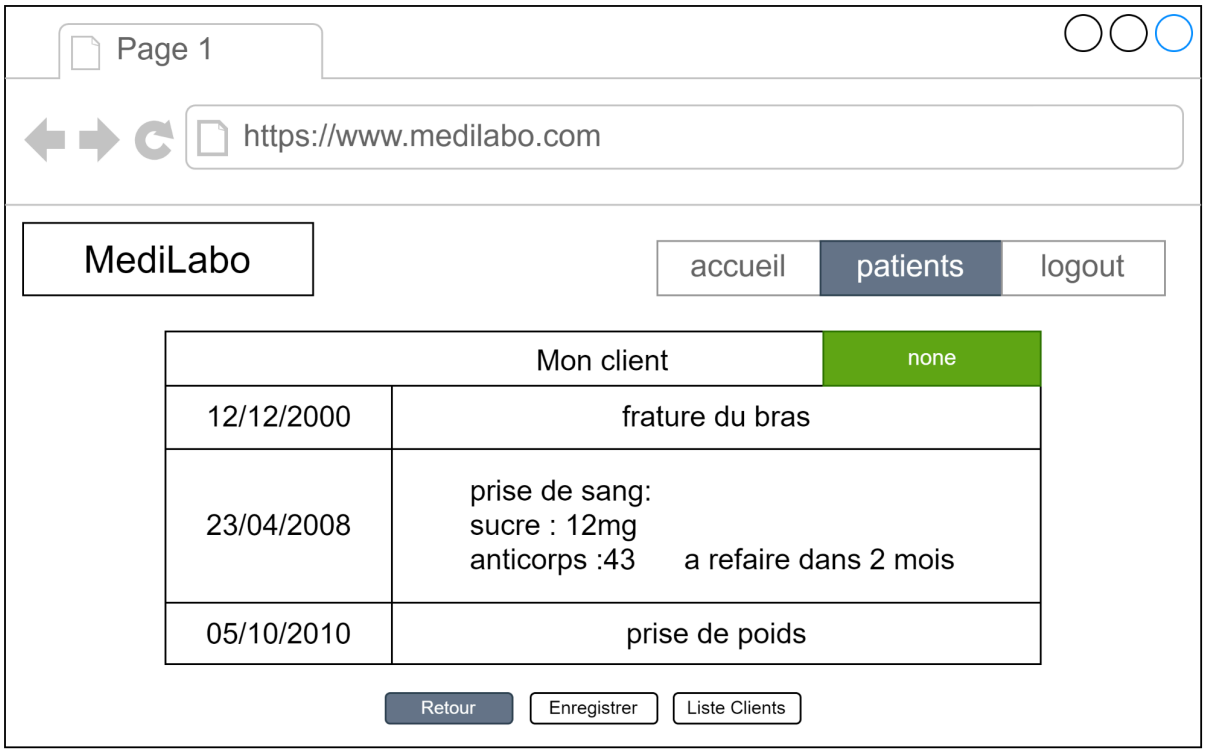


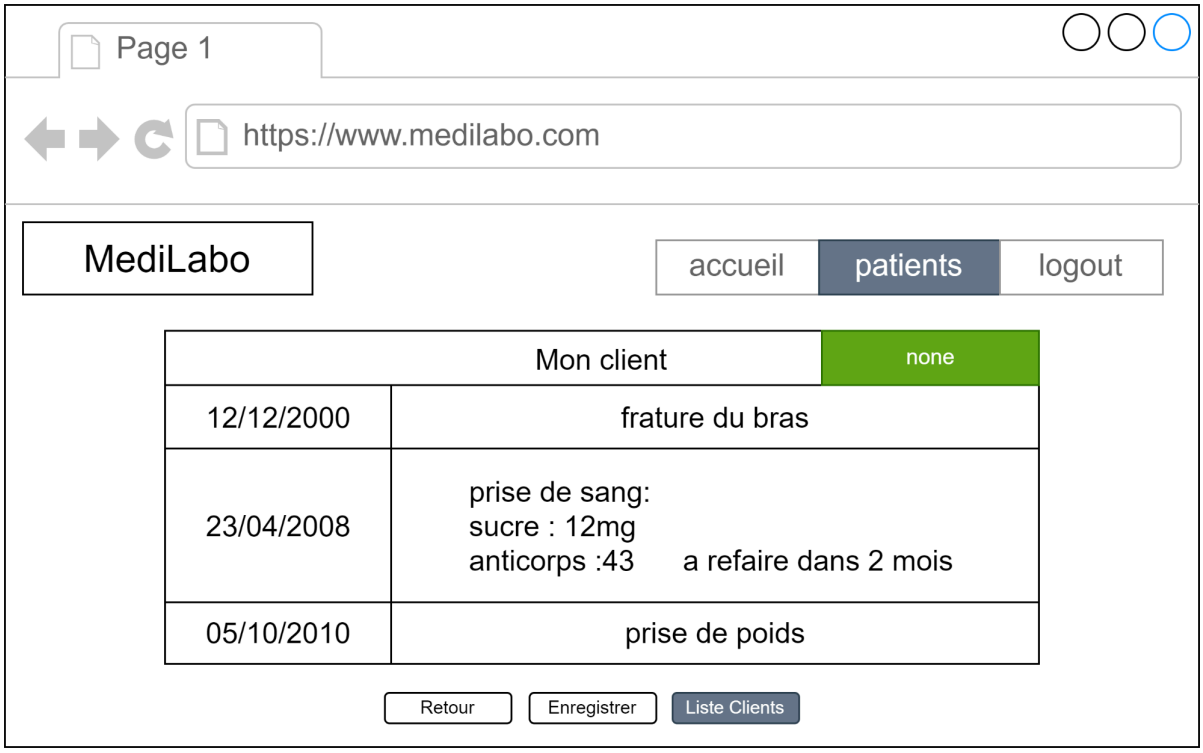


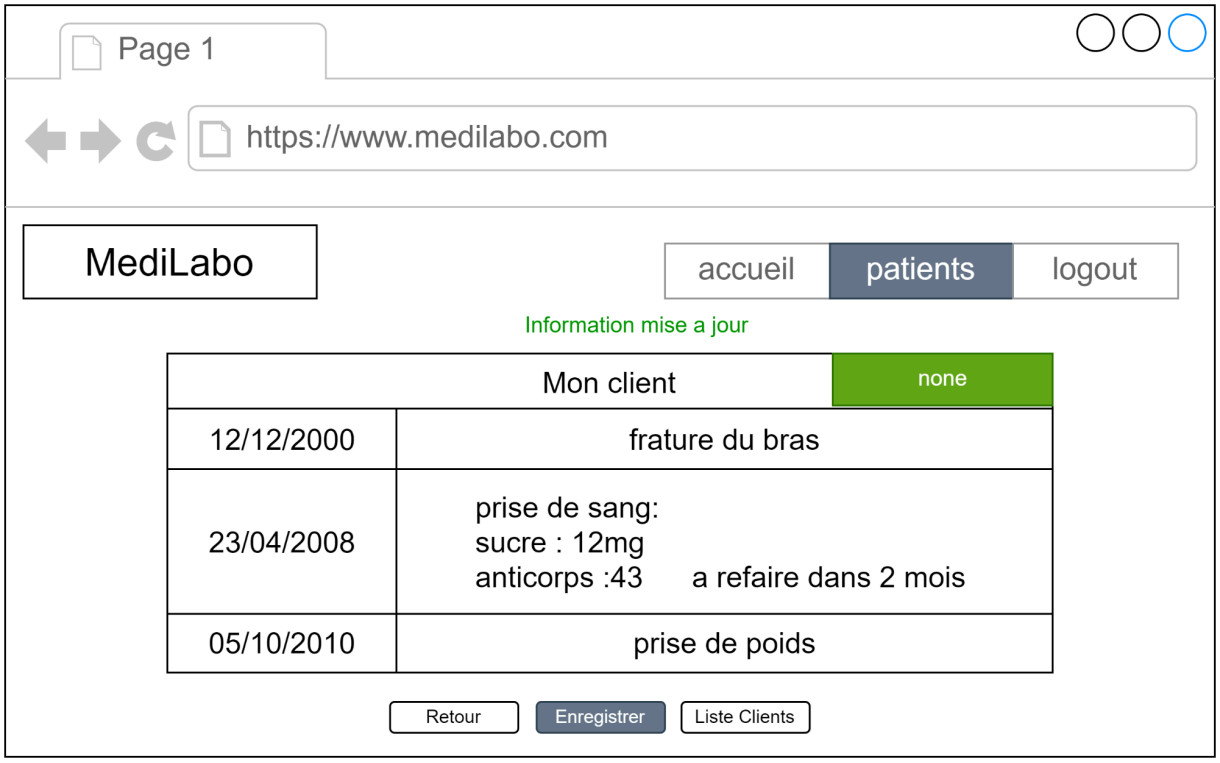
User stories 6

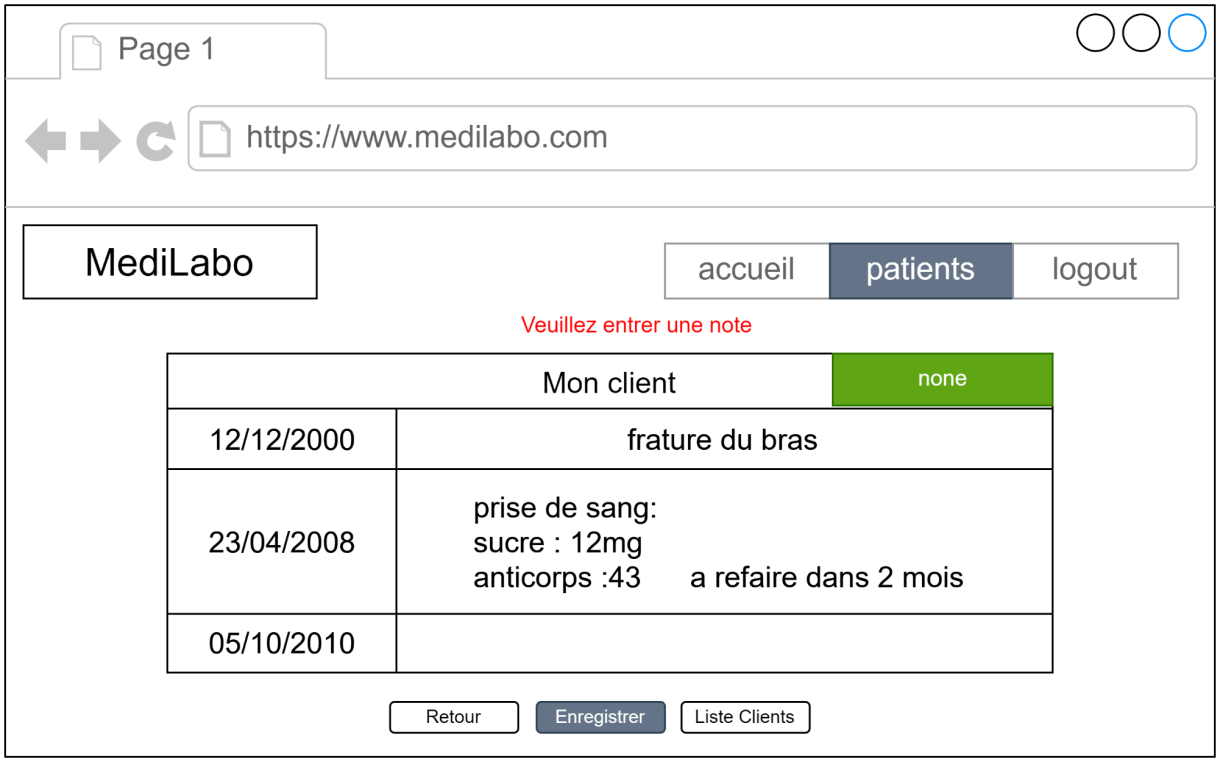


User stories 7






Page 1
https://www.medilabo.com
MediLabo
accueil
patients
logout
Information mise a jour
Mon client
none
12/12/2000
frature du bras
23/04/2008
prise de sang:
sucre : 12mg
anticorps :43 a refaire dans 2 mois
05/10/2010
prise de poids
Retour
Enregistrer
Liste Clients


Page 1
https://www.medilabo.com
MediLabo
accueil
patients
logout
Veuillez entrer une note
Mon client
none
12/12/2000
frature du bras
23/04/2008
prise de sang:
sucre : 12mg
anticorps :43 a refaire dans 2 mois
05/10/2010
Retour
Enregistrer
Liste Clients

3. Spécifications techniques

3.1 Besoin API (Endpoint API)

Connexion Authentification et sécurité

Chaque service backend (gateway, patient, note et report) on une connexion sécurisé :

- Authentification : L'API utilise HTTP Basic Authentication. Les identifiants doivent être fournis dans l'en-tête Authorization de chaque requête :
    - Paramètres : username et password.
- Endpoints : Il n'y a pas d'endpoint spécifique pour la connexion. L'authentification est incluse directement dans les requêtes API.

    Réponses possibles :

    - Succès : HTTP 200 OK.
    - Échec : HTTP 401 Unauthorized.
- CSRF : Désactivé, car nous utilisons HTTP Basic Authentication.
- Déconnexion :
    - Procédure : Pour se déconnecter, supprime les informations d'authentification stockées côté client
    - Pas de Endpoint de Déconnexion : Il n'y a pas de requête spécifique pour la déconnexion avec HTTP Basic Authentication.

3.1.1 Microservice Front-End Angular

Le service front-end Angular gère l'affichage et l'interaction utilisateur, et communique avec le micro service Gateway pour accéder aux données et fonctionnalités.

Pages et Vues

1. **Page de Connexion**
    - **URL** : `/login`
    - **Description** : Permet aux utilisateurs de se connecter en utilisant HTTP Basic Authentication. Les identifiants doivent être fournis dans l'en-tête Authorization de chaque requête API.
    - **Réponse Frontend** :
        - **Succès** : Redirection vers la page d'accueil (`/home`).
        - **Échec** : Affichage d'un message d'erreur indiquant que les identifiants sont incorrects.
2. **Page d'Accueil**
    - **URL** : `/home`
    - **Description** : Affiche la page d'accueil de l'application après authentification réussie.
    - **Réponse Frontend** :
        - **Succès** : Affichage du contenu de la page d'accueil.
        - **Échec** : Redirection vers la page de connexion (`/login`) avec un message d'erreur.
3. **Page de Liste des Patients**
    - **URL** : `/clientele`
    - **Description** : Affiche la liste des patients. Les données sont récupérées via le microservice Gateway.
    - **Réponse Frontend** :
        - **Succès** : Affichage de la liste des patients.
        - **Échec** : Affichage d'un message d'erreur indiquant que la récupération des données a échoué.
4. **Page de Détails d'un Patient**
    - **URL** : `/patient/{patientId}`
    - **Description** : Affiche les détails d'un patient spécifique. Permet également d'accéder aux options pour modifier ou ajouter des informations.
    - **Réponse Frontend** :

        - **Succès** : Affichage des détails du patient avec les options pour modifier ou ajouter des informations.
        - **Échec** : Affichage d'un message d'erreur indiquant que les détails du patient n'ont pas pu être récupérés.
    5. **Page des Notes Médicales d'un Patient**
        - **URL** : `/patient/notes/{patientId}`
        - **Description** : Affiche les notes médicales d'un patient spécifique. Permet également d'ajouter ou de modifier une note médicale. Le rapport de risque est intégré comme un voyant sur la page.
        - **Réponse Frontend** :
            - **Succès** : Affichage des notes médicales et du rapport de risque. Les options pour ajouter ou modifier des notes sont également disponibles.
            - **Échec** : Affichage d'un message d'erreur indiquant que les notes médicales n'ont pas pu être récupérées ou que l'ajout/modification a échoué.

3.1.2 Microservice Gateway

Le microservice Gateway reçoit les requêtes du front-end, les dirige vers les microservices appropriés, et renvoie les réponses au front-end en utilisant les mêmes objets.

1. Patient
    - ALL METHOD for enpoint non spécifié
        - Description : pour toutes les routes non définies dans la configuration de transformation des routes.
        - Call : GET /fallback
2. Patient
    - GET /api/clientele
        - Description : Demande la liste des patients au microservice backend patient.
        - Call : GET /clientele
    - GET /api/clientele/patient/{patientId}
        - Description : Demande les informations personnelles d'un patient au microservice backend patient.
        - Call : GET /patient/{patientId}

- POST /api/clientele/patient
  - Description : Met à jour ou ajoute les informations personnelles d'un patient via le microservice backend patient.
  - Call : POST /patient

3. FollowNote
   - GET /api/clientele/patient/notes/{patientId}
     - Description : Demande les notes concernant un patient au microservice backend follow-note.
     - Call : GET notes/{patientId}
   - POST /api/clientele/patient/notes/note
     - Description : Ajoute une note concernant un patient via le microservice backend follow-note.
     - Call : POST /note
4. Report
   - GET /api/clientele/patient/notes/report/{patientId}
     - Description : Demande le rapport de risque diabète concernant un patient au microservice backend report.
     - Call : GET /report/{patientId}

3.1.3 Microservice Backend Patient

Le microservice Backend Patient gère les informations personnelles des patients.

- GET /clientele
    - Description : Renvoie la liste des patients.
- GET /patient/{patientId}
    - Description : Renvoie les informations personnelles d'un patient.
- POST /patient
    - Description : Mise à jour ou ajouter des informations personnelles d'un patient.

3.1.4 Microservice Follow-Note Backend

Le microservice Follow-Note Backend gère les notes et suivis des patients.

- GET /follow-note/{patientId}
    - Description : Renvoie les notes concernant un patient.
- POST /note
    - Description : Ajoute une note concernant un patient.

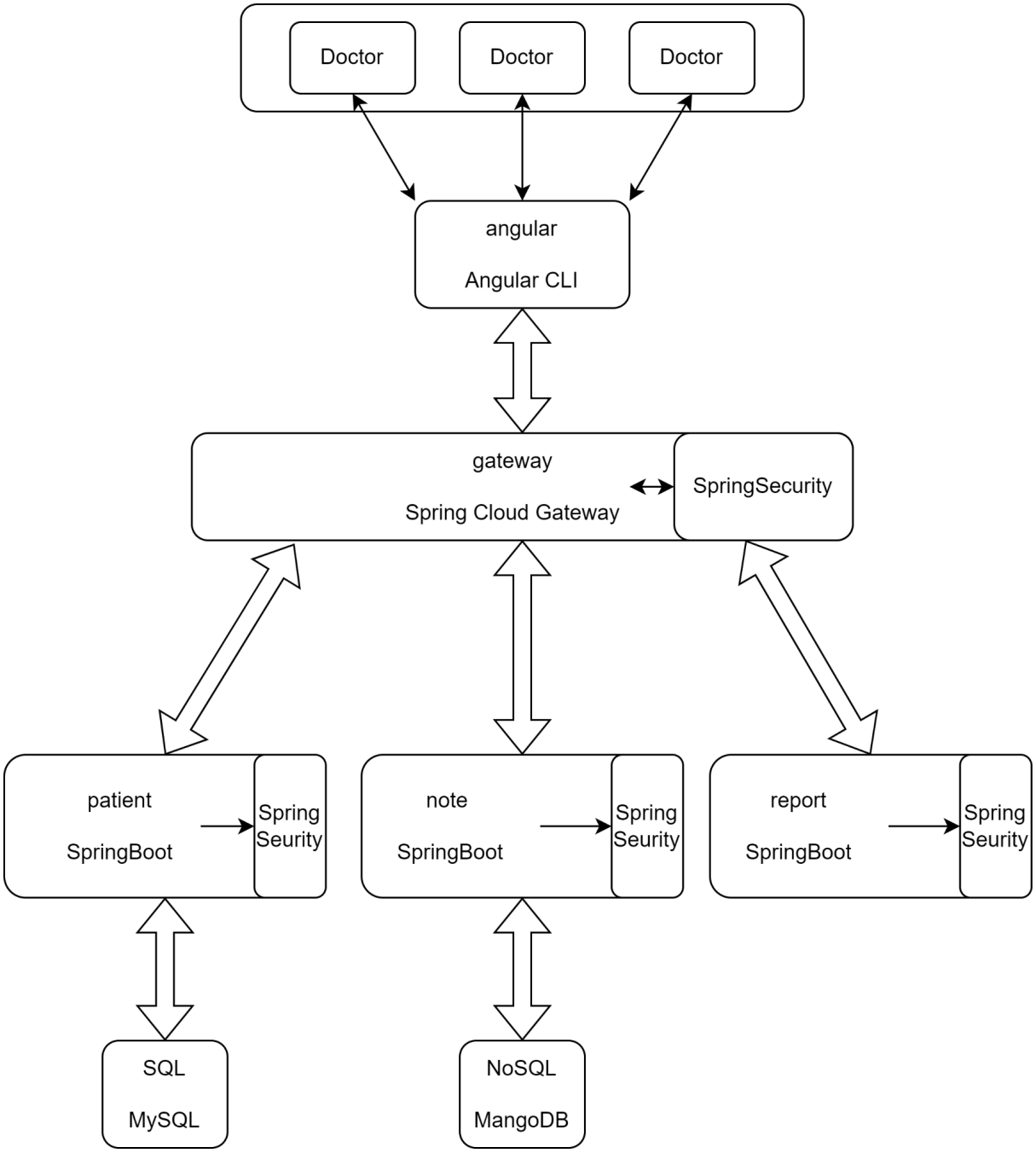
3.1.5 Microservice Report Backend

Le microservice Report Backend génère les rapports de risques pour les patients.

- GET /report/{patientId}
    - Description : Renvoie le rapport de risque diabète concernant un patient.

3.2 Schémas de conception technique

1. Schéma d'Architecture Globale



Explications :

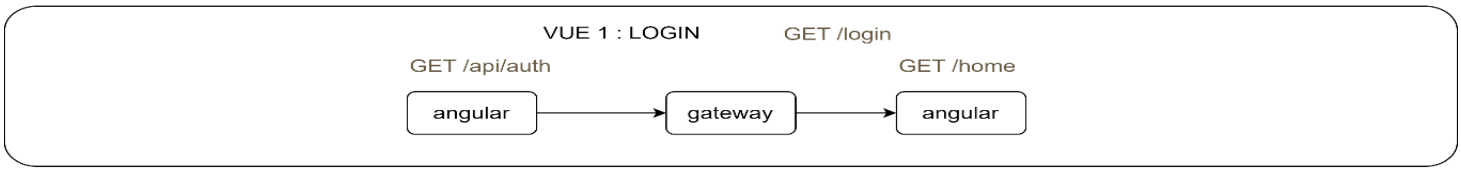
1. angular(Frontend Angular) :
    - Représente le côté client de l'application, développé avec Angular. Il envoie des requêtes HTTP au Backend.
2. gateway (Backend Spring Cloud Gateway) :
    - Composant de passerelle API pour la gestion des requêtes HTTP entrant dans le système.
3. SpringSecurity :
    - Composant de sécurité de Spring, utilisé pour sécuriser les applications Backend.
4. patient(Backend SpringBoot) :
    - Composant Backend utilisant Spring Boot, gérant les opérations liées aux entités Patient.

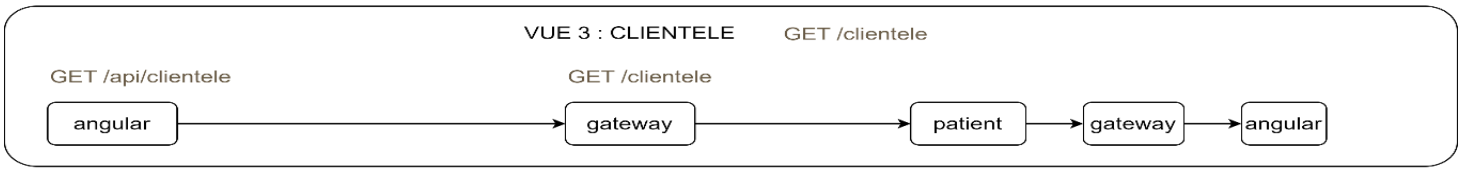
5.note (Backend, SpringBoot) :

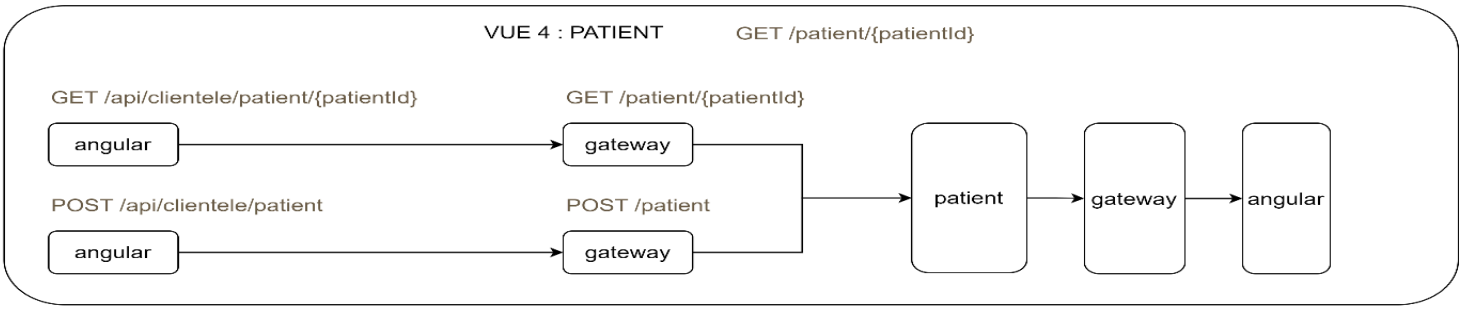
- Composant Backend utilisant Spring Boot, gérant les opérations liées aux entités Note.

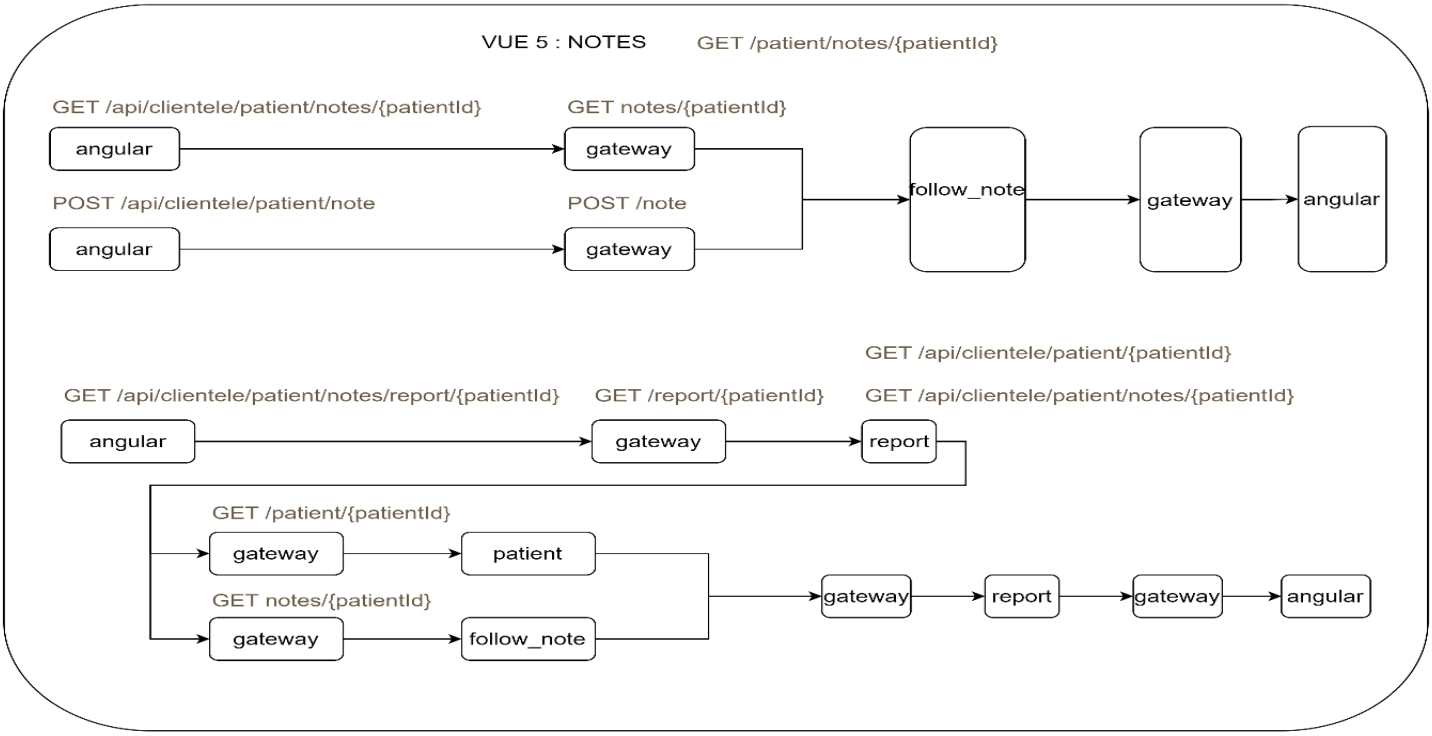
6. report (Backend, SpringBoot) :
    - Composant Backend dédié à la gestion des rapports, également basé sur Spring Boot.
7. SQL (MySQL) :
    - Base de données SQL utilisant MySQL pour le stockage des données structurées.
8. NoSQL (MongoDB) :
    - Base de données NoSQL utilisant MongoDB pour le stockage des données non structurées.
9. Doctor :
    - Utilisateur ou acteur dans le système qui interagit avec les entités et les composants via l'interface.

2. Schéma flux Api





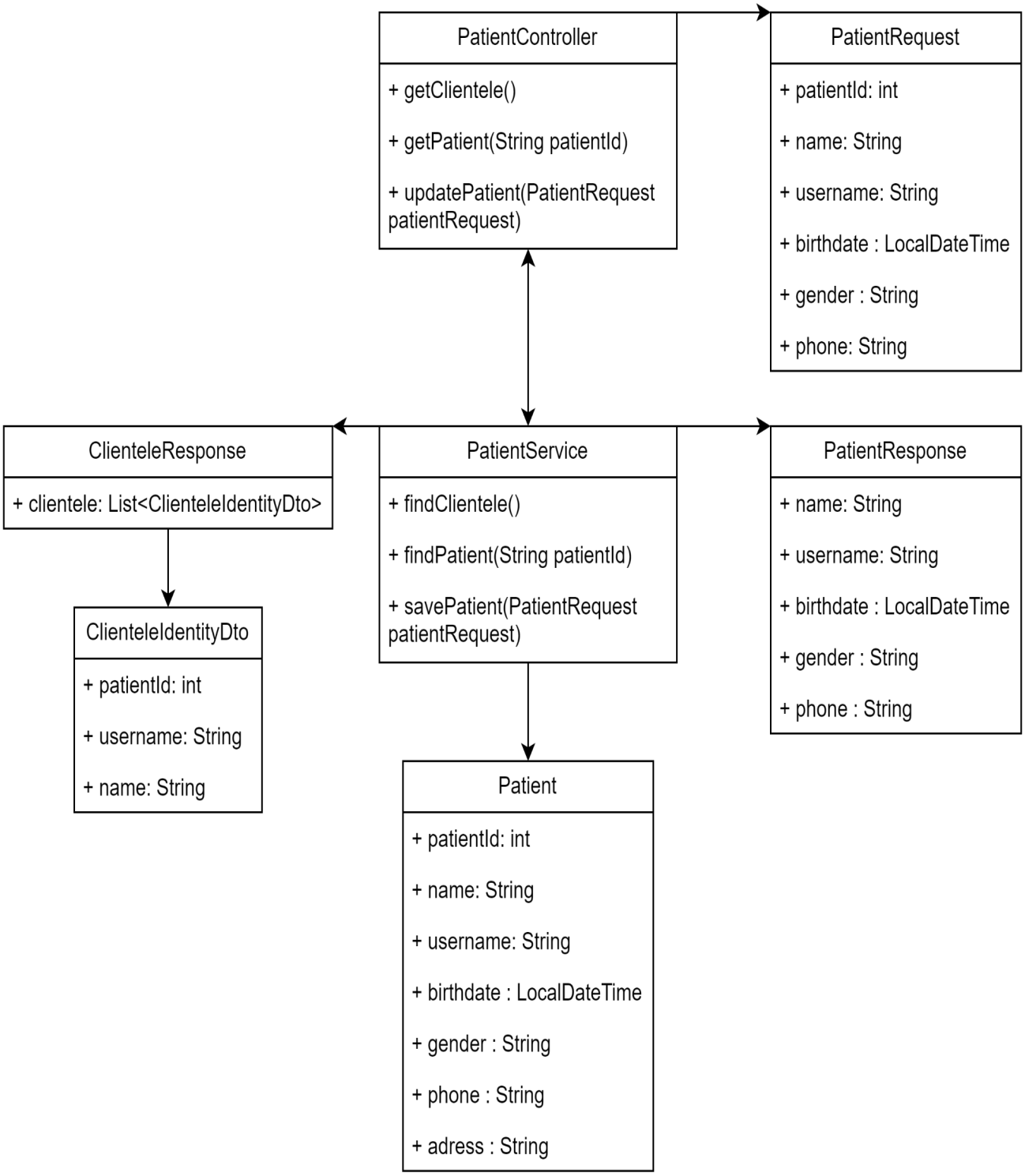




Connexions :

- Les flèches dans le diagramme représentent les interactions et les flux de données entre les différents composants.
    - angular communique avec gateway pour envoyer des requêtes.
    - gateway dirige les requêtes vers les composants Backend appropriés (comme patient, note, report).
    - Les composants Backend (note et patient) interagissent avec les bases de données SQL (MySQL) et NoSQL (MongoDB) pour lire et écrire des données.
    - le module report communique avec le module patient et note pour récupérer les informations nécessaires à sa tâche en passant par le gateway.

3. Schéma UML simplifié patient



Explications :

- PatientController :
  - Reçoit des requêtes HTTP, les convertit en objets PatientRequest, et les passe au PatientService.
  - Utilise ClienteleResponse et/ou PatientResponse pour renvoyer les données au client.
- PatientService :
  - Traite la logique métier, interagit avec Patient pour accéder aux données persistées, et crée des objets ClienteleResponse et PatientResponse pour les envoyer au contrôleur.
- Patient :
  - Représente l'entité de base de données avec les informations persistées.
- PatientRequest, PatientResponse et ClienteleResponse, Idendity:
  - Sont des DTOs utilisés pour transférer des données entre le client et le serveur.
  - Variables
    - PatientController

      patientService : Service utilisé pour gérer les informations des patients.

      clienteleService : Service utilisé pour gérer les informations des clientèles.
    - PatientService

      FindClientele : Méthode pour trouver les informations de la clientèle.

      FindPatient : Méthode pour trouver les informations d'un patient spécifique.

      SavePatient : Méthode pour enregistrer les informations d'un patient.
    - PatientRequest

      patientId : Identifiant du patient.

      name : Nom du patient.

username : Nom d'utilisateur du patient.

birthdate : Date de naissance du patient.

genre : Genre du patient.

telephone : Numéro de téléphone du patient.

- PatientResponse

  name : Nom du patient dans la réponse.

  username : Nom d'utilisateur du patient dans la réponse.

  birthdate : Date de naissance du patient dans la réponse.

  genre : Genre du patient dans la réponse.

  telephone : Numéro de téléphone du patient dans la réponse.

- Patient

  patientId : Identifiant du patient.

  name : Nom du patient.

  username : Nom d'utilisateur du patient.

  birthdate : Date de naissance du patient.

  genre : Genre du patient.

  telephone : Numéro de téléphone du patient.

  adresse : Adresse du patient.

- ClienteleResponse

  clientele : Liste des réponses des patients.

- Méthodes
  - PatientController

    + getClientele() : Méthode pour obtenir les informations de la clientèle.

    + getPatient(String patientId) : Méthode pour obtenir les informations d'un patient spécifié par son identifiant (patientId).

+ updatePatient(PatientRequest patientRequest) : Méthode pour mettre à jour les informations d'un patient en utilisant une requête de type PatientRequest.

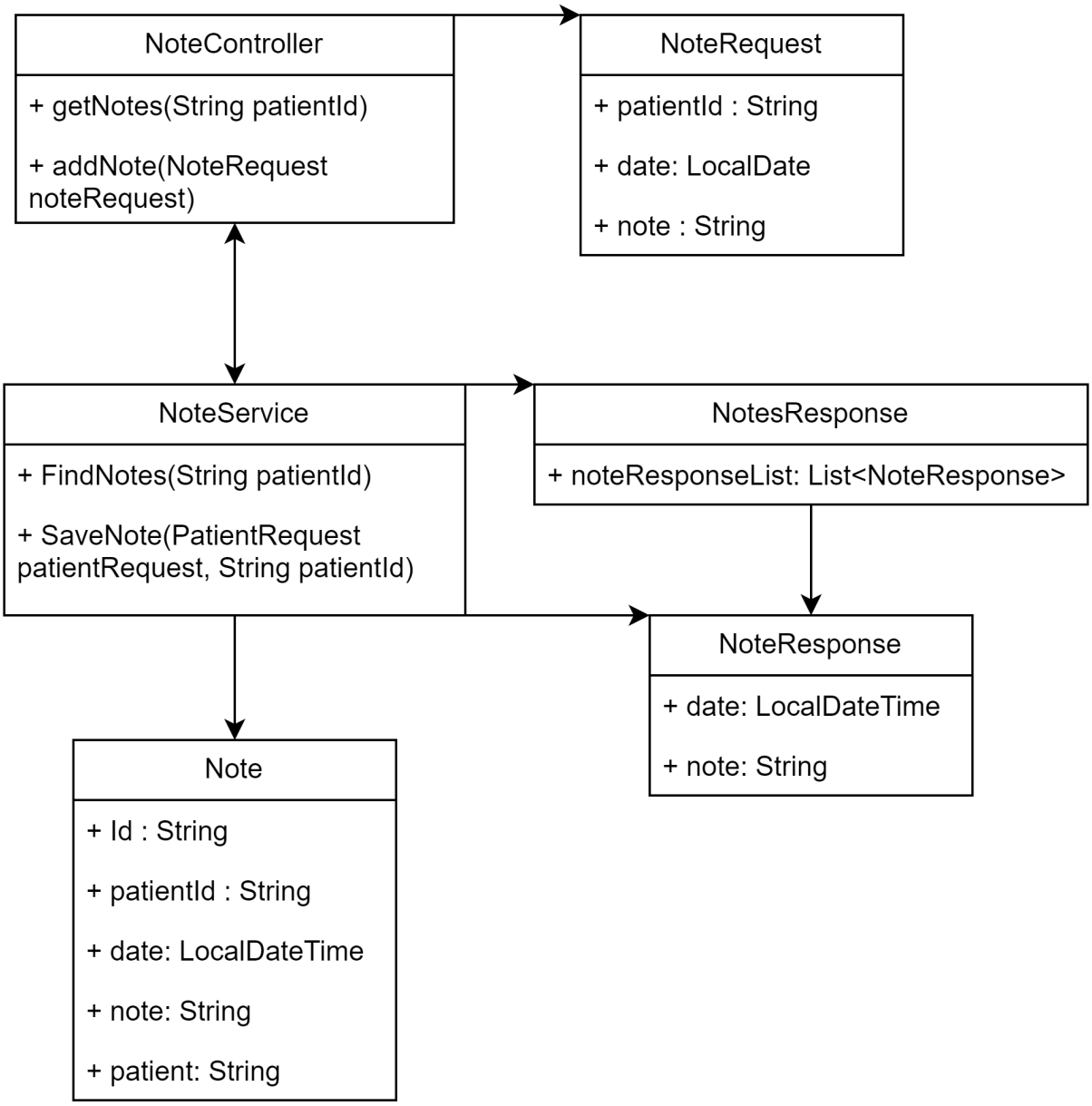
- PatientService

  + FindClientele() : Méthode pour trouver les informations de la clientèle.

  + FindPatient(String patientId) : Méthode pour trouver les informations d'un patient spécifié par son identifiant (patientId).

  + SavePatient(PatientRequest patientRequest) : Méthode pour enregistrer les informations d'un patient en utilisant une requête de type PatientRequest.

4. Schéma UML simplifié Note



Explications :

- NoteController:

  Reçoit des requêtes HTTP, les convertit en objets NoteRequest, et les passe au NoteService.

  Utilise NotesResponse et/ou NoteResponse pour renvoyer les données au client.

- NoteService: :

  Traite la logique métier, interagit avec Note pour accéder aux données persistées, et crée des objets NotesResponse et NoteResponse pour les envoyer au contrôleur.

- Note :

  Représente l'entité de base de données avec les informations persistées.

- NotesResponse , NoteResponse et NoteRequest:

  Sont des DTOs utilisés pour transférer des données entre le client et le serveur.

- Variables
  - NoteRequest

    patientId : Identifiant du patient.

    date : Date de la note.

    note : Contenu de la note.

  - NotesResponse

    noteResponseList : Liste des réponses de note

  - Note

    Id : Identifiant de la note.

    patientId : Identifiant du patient associé à la note.

    date : Date de la note.

    note : Contenu de la note.

    patient : le nom du patient

- NoteResponse

    date : Date de la note.

    note : Contenu de la note.

- Méthodes
    - NoteController

        + getNotes(String patientId) : Méthode pour obtenir les notes de suivi d'un patient spécifié par son identifiant (patientId).

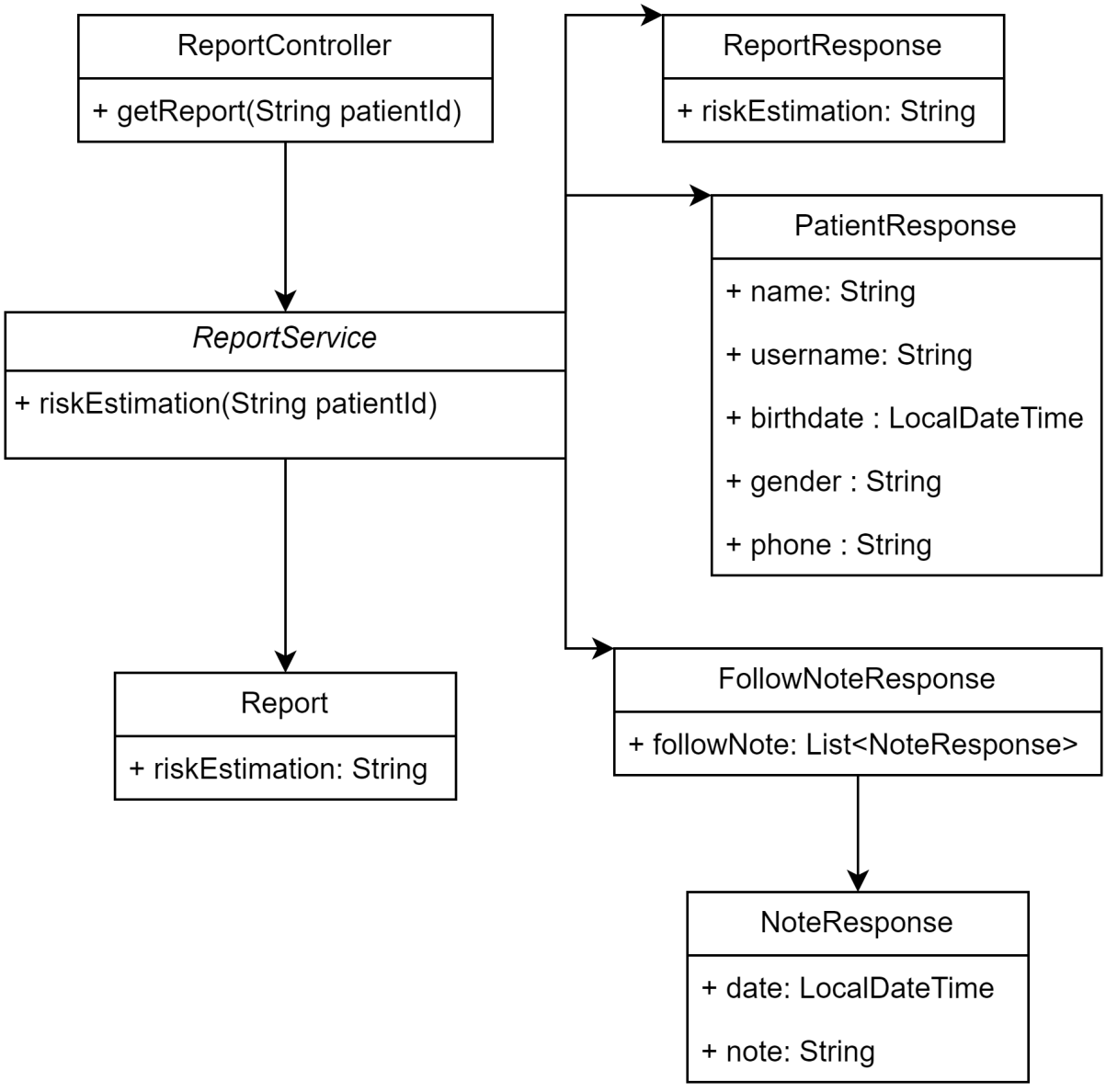
        + addNote(NoteRequest noteRequest) : Méthode pour ajouter une nouvelle note de suivi.

    - NoteService

        + FindNotes(String patientId) : Méthode pour trouver les notes de suivi d'un patient spécifié par son identifiant (patientId).

        + SaveNote(PatientRequest patientRequest, String patientId) : Méthode pour enregistrer une note pour un patient spécifié par son identifiant (patientId).

4. Schéma UML simplifié Report



Explications :

- ReportController:

  Utilise ReportResponse pour renvoyer les données au client.
- ReportController :

Traite la logique métier, interagit avec Note pour accéder aux données persistées, et crée des objets FollowNoteResponse et NoteResponse pour les envoyer au contrôleur.

- Note :

  Représente l'entité de base de données avec les informations persistées.

- ReportResponse:

  Est le DTO utilisé pour transférer des données entre le client et le serveur.

- Variables
  - ReportController

    patientService : Service pour gérer les informations des patients.

    followNoteService : Service pour gérer les notes de suivi des patients.

  - Report

    riskEstimation : Estimation du risque de diabète

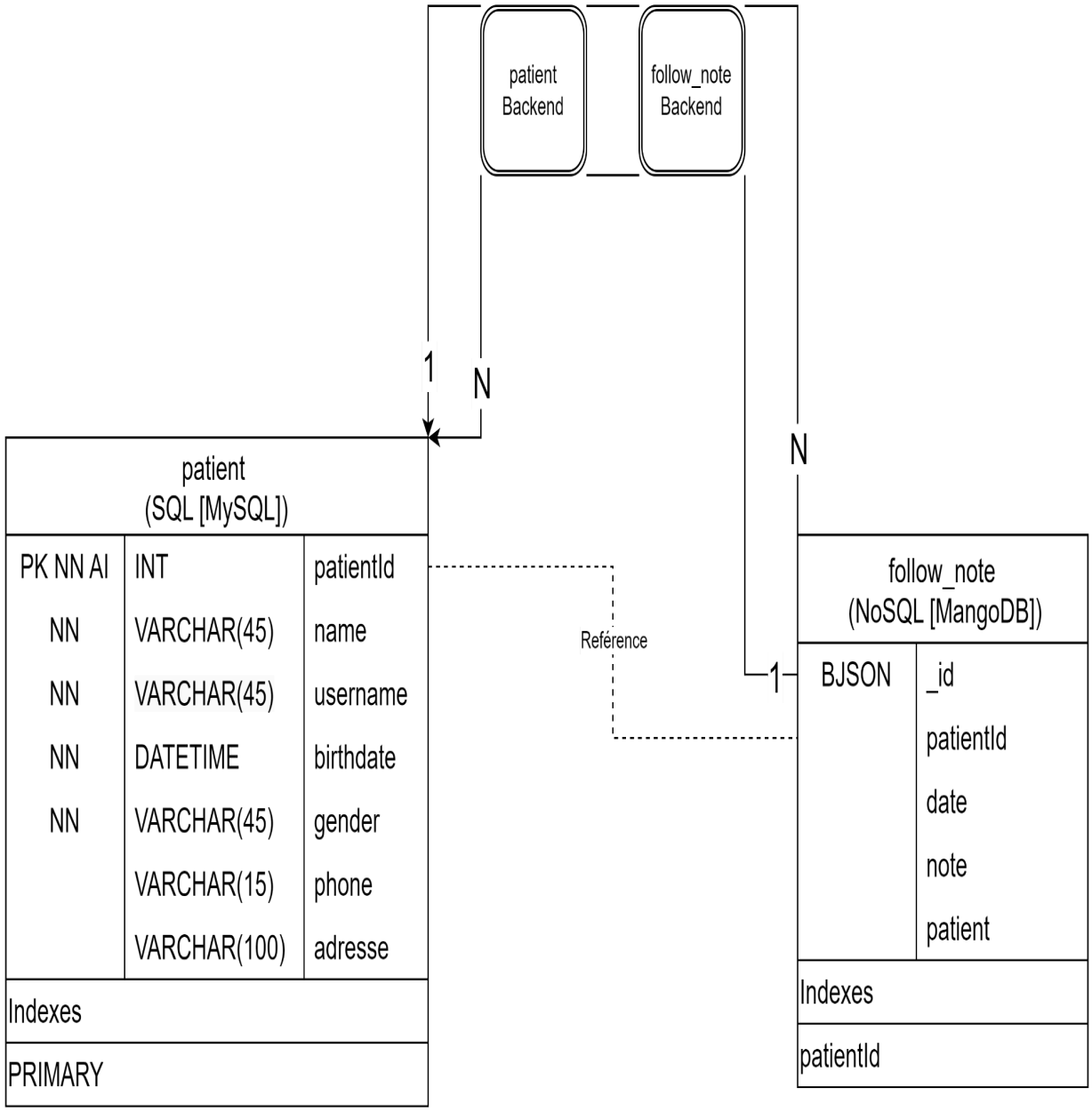
  - ReportResponse

    riskEstimation : Estimation du risque de diabète retournée dans la réponse.
- Méthodes
  - ReportController

    + getReport(String patientId) : Méthode pour obtenir le rapport de risque pour un patient spécifié par son identifiant (patientId).

Diagramme de Références Croisées entre Bases de Données

Cette section présente un diagramme des relations logiques entre les entités des différentes bases de données utilisées par nos microservices. Ce diagramme met en évidence la manière dont les données, bien que séparées par microservice, interagissent entre elles au travers des clés étrangères et des relations de dépendance.



Explication

- Table patient (SQL) : Représente une table SQL contenant des informations sur les patients, structurée avec des colonnes telles que patientId, nom, date de naissance, etc.
- Collection note(NoSQL) : Représente une collection NoSQL contenant des objets BSON (Binary JSON) qui représentent des notes chacune associé à un patient.Les objets BSON incluent des informations nécessaire pour la note
- Microservice patient : Un microservice qui gère les opérations sur les données de la table SQL Patient, telles que la création, la lecture, la mise à jour et la suppression des informations sur les patients.
- Microservice note: Un microservice qui gère les opérations sur les notes dans la collection NoSQL note, telles que la création, la lecture, la mise à jour et la suppression des informations sur les notes.
- Flèche noire (relation identifiée) : Indique une relation explicite entre les entités. Dans ce cas, une référence indirecte entre la table et la collection gérée par les micro services existe .
- Trait en pointillé annoté "référence" : Montre la référence entre les deux bases de données via un point commun (patientId). Cette relation est gérée par les microservices, qui permettent la synchronisation ou l'interaction entre la table SQL et la collection NoSQL.
- patientId : Identifiant utilisé dans les deux systèmes (SQL et NoSQL) pour associer les notes de la collection note aux enregistrements des patients dans la table SQL Patient.

- Variables
  - patientId: Identifiant du patient (clé primaire).
  - name: Nom du patient.
  - username : prénom du patient
  - birthdate: Date de naissance du patient.
  - gender: Genre du patient.
  - phone: Numéro de téléphone du patient.
  - adresse: Adresse du patient.
  - _id: Identifiant unique (le nom ne peut être changé (mangodb)
  - date : date de la prise de note
  - note : le compte rendu de l’utilisateur
  - patient: nom du patient

- Types de Données
    - INT: Type de données pour l'identifiant du patient, de type entier.
    - VARCHAR(45): Type de données pour le nom du patient et le genre du patient, chaîne de caractères, maximum 45 caractères.
    - DATETIME: Type de données pour la date de naissance, de type date et heure.
    - VARCHAR(15): Type de données pour le numéro de téléphone du patient, chaîne de caractères, maximum 15 caractères.
    - VARCHAR(100): Type de données pour l'adresse du patient, chaîne de caractères, maximum 100 caractères.
    - BJSON: Type de données BSON (Binary JSON) pour le stockage des notes dans MongoDB.
    - PK

3.3 Solutions techniques

Spring Boot 3.3.2

- **Justification** : Utilisation de Spring Boot pour simplifier le développement d'applications Java en fournissant des configurations prêtes à l'emploi et un support intégré pour de nombreuses fonctionnalités, telles que les contrôleurs REST, la gestion des dépendances, et l'intégration avec diverses bases de données et services.

Java 21

- **Justification** : Choix de la version Java la plus récente pour bénéficier des dernières fonctionnalités du langage, des améliorations de performance et des corrections de sécurité. Java 21 assure également la compatibilité avec Spring Boot 3.3.2.

Docker pour la Conteneurisation

- **Justification** : Docker est utilisé pour conteneuriser l'application, facilitant le déploiement, l'exécution, et la gestion des environnements de développement et de production. Les images Docker garantissent que l'application fonctionne de manière cohérente quel que soit l'environnement.

MangoDB

- **Justification** :Les bases de données orientées document stockent et retrouvent elles aussi les données sous forme de paire clé-valeur. Toutefois, la valeur est stockée sous forme de document au format BJSON ou XML. La valeur est ainsi comprise par la base de données et peut être trouvée à l'aide d'une requête.

Angular :

- **Justification** : Framework utilisé pour construire l'interface utilisateur côté client. Il permet de créer des applications web dynamiques en utilisant TypeScript et fournit des outils puissants pour le développement de SPA (Single Page Applications).

Spring Cloud Gateway :

- **Justification** : Utilisé comme passerelle API pour gérer les requêtes des clients et les acheminer vers les microservices appropriés. Il offre des fonctionnalités telles que la gestion du routage, des filtres et de la résilience.

Gestion de Version et CI/CD

- **GitHub pour le Versionnage du Code Source**
  - **Justification** : L'utilisation de GitHub pour le versionnage du code source permet de suivre l'historique des modifications, de gérer les branches et les merges, et de collaborer efficacement avec l'équipe de développement. GitHub offre une interface conviviale, des outils de collaboration puissants, et une intégration native avec GitHub Actions pour CI/CD.
- **GitHub Actions pour CI/CD**
  - **Justification** : GitHub Actions est choisi pour son intégration native avec GitHub, ce qui simplifie le flux de travail de développement et de déploiement. La gestion des conteneurs Docker via GitHub Container Registry (GHCR) permet un déploiement fluide et directement lié au dépôt source. Les images Docker construites peuvent être utilisées pour les déploiements orchestrés grâce à docker compose.

Structure du Pipeline CI/CD

- **Étape 1 : Build**
  - **Description** : Compilation du code source et construction des artefacts, notamment les images Docker. Cette étape utilise Maven pour la compilation Java et Docker Compose pour la création des images Docker, tout en incluant un label qui lie les images directement au dépôt GitHub.

- **Étape 2 : Tests**
  - **Description** : Exécution des tests unitaires et d'intégration pour valider le code. Les résultats des tests, ainsi que le succès de la construction des artefacts, sont notifiés par email, assurant une transparence totale pour l'équipe de développement.
- **Étape 4 : Déploiement**
  - Description : Déploiement automatique des images Docker sur GitHub Container Registry (GHCR). Les images sont étroitement liées au dépôt source et aux releases correspondantes. Un fichier compose.yaml est fourni pour simplifier l'installation et l'orchestration des services via Docker Compose.

Conclusion

- Le pipeline CI/CD décrit assure une intégration continue et un déploiement efficace, grâce à GitHub Actions et GHCR. Les images Docker sont automatiquement liées au dépôt de code source, ce qui permet un déploiement facile avec Docker Compose. Des notifications par email sont envoyées pour informer l'équipe du succès des tests et du déploiement.

Qualité du Code et Sécurité

- **JaCoCo**
  - **Justification** : JaCoCo mesure le pourcentage de code testé par les tests automatisés, garantissant une couverture adéquate.
- **Surefire**
  - **Justification** : Surefire exécute les tests unitaires et d'intégration, fournissant des rapports détaillés sur les résultats des tests.

3.4 Dockerisation

Pour garantir une cohérence et une portabilité optimales à travers les différents environnements de développement, de test, et de production, tous les composants de l'application seront dockerisés dès le début. Voici une description détaillée des composants qui seront conteneurisés et de leur configuration.

Configuration des Volumes et Réseaux

1. **Volumes :**
   - `mysql_data` : Persistance des données SQL.
   - `mongodb_data` : Persistance des données MongoDB.

Description : Ces volumes assurent que les données critiques ne sont pas perdues lorsque les conteneurs sont redémarrés ou recréés.

2. **Réseaux :**

   - ```
     my_medilab:

         driver: bridge
     ```

Description : Création d'un réseau Docker personnalisé pour permettre une communication entre les micro service à l'aide du driver bridge et la création du réseau
     ```

Composants Dockerisés

1. Base de Données SQL

- **Description :** Conteneurisation de la base de données SQL utilisée pour stocker les données structurées.
- **Image Docker :** Utilisation de l'image officielle MySQL (`mysql:8.0`).
- **Configuration :**
  - **Volume :**
    - **Nom du volume :** `mysql_data`
    - **Montage :**
      - `/var/lib/mysql`
      - `./init_scripts/mysql:/docker-entrypoint-initdb.d (emplacement script initial)`
    - **Description :** Ce volume garantit la persistance des données SQL, même en cas de redémarrage ou de suppression du conteneur.
  - **Port :**
    - **Port exposé :** `3306:3306`
    - **Description :** Le port `3306` est mappé pour permettre l'accès à la base de données MySQL depuis l'hôte ou d'autres services.
  - **Variables d'environnement :**
    - `MYSQL_CONTAINER_NAME=mysql_db`
    - `MYSQL_DATABASE=medilabo_patient`
    - `MYSQL_USER=user`
    - `MYSQL_PASSWORD=user`
    - `MYSQL_ROOT_PASSWORD=root`
    - `MYSQL_PORT=3306:3306`

2. Base de Données MongoDB

- **Description :** Conteneurisation de la base de données MongoDB pour le stockage de données non structurées au format JSON.
- **Image Docker :** Utilisation de l'image officielle MongoDB (`mongo:6.0`).
- **Configuration :**
  - **Volume :**
    - **Nom du volume :** `mongodb_data`
    - **Montage :**
      - `/data/db`
      - `./init_scripts/mangodb:/docker-entrypoint-initdb.d (script intial)`
    - **Description :** Le volume `mongodb-data` est utilisé pour persister les données MongoDB.
  - **Port :**
    - **Port exposé :** `27017:27017`
    - **Description :** Le port `27017` est mappé pour permettre l'accès à la base MongoDB.
  - **Variables d'environnement :**
    - `MONGODB_CONTAINER_NAME=mongodb`
    - `MONGO_INITDB_DATABASE=medilabo_note`
    - `MONGO_INITDB_ROOT_USERNAME=user`
    - `MONGO_INITDB_ROOT_PASSWORD=user`
    - `MONGODB_PORT_EXT=27017`
    - `MONGODB_PORT_INT=27017`
    - `MONGODB_AUTH_DATABASE=admin`
    - `MONGODB_HOST=mongodb`
    - `MONGODB_USERNAME=user`
    - `MONGODB_PASSWORD=user`
    - `MONGODB_URL: mongodb://${MONGODB_HOST}/${MONGO_INITDB_DATABASE}?authSource=${MONGODB_AUTH_DATABASE}`

3. Microservice Front-End Angular

- **Description :** Conteneurisation de l'application front-end développée avec Angular.
- **Image Docker :** Création d'une image Docker personnalisée pour Angular basée sur l'image `node:18.2.1` et le build Angular.
- **Configuration :**
  - **Port :**
    - **Port exposé :** `4200:80`
    - **Description :** Le port `4200` est utilisé pour le développement Angular, mappé sur le port `80` du conteneur.
  - **Variables d'environnement :**
    - **`ANGULAR_CONTAINER_NAME=angular`**
    - **`BUILD_PATH_ANGULAR=./frontend/angular`**
    - **`ANGULAR_PORT=4200:4200`**

Microservice Follow-Note Backend

- **Description :** Conteneurisation du microservice Follow-Note pour la gestion des notes et suivis des patients.
- **Image Docker :** Création d'une image Docker à partir d'une configuration Spring Boot (`openjdk:21-jdk-slim`).
- **Configuration :**
  - **Port :**
    - **Port exposé :** `8081:8080`
    - **Description :** Le port `8081` est mappé pour servir le microservice Follow-Note.
  - **Variables d'environnement :**
    - `NOTE_CONTAINER_NAME=note`
    - `BUILD_PATH_FOLLOWNOTE=./backend/note`
    - `NOTE_PORT=8081:8081`
    - `MONGODB_URL=mongodb://mongodb:27017/medilabo_note`
    - `MONGODB_USERNAME=user`
    - `MONGODB_PASSWORD=user`

5. Microservice Patient Backend

- **Description :** Conteneurisation du microservice Patient pour la gestion des informations personnelles des patients.
- **Image Docker :** Création d'une image Docker à partir d'une configuration Spring Boot (`openjdk:21-jdk-slim`).
- **Configuration :**
  - **Port :**
    - **Port exposé :** `8082:8080`
    - **Description :** Le port `8082` est mappé pour servir le microservice Patient.
  - **Variables d'environnement :**
    - `PATIENT_CONTAINER_NAME=patient`
    - `BUILD_PATH_PATIENT=./backend/patient`
    - `PATIENT_PORT=8082:8082`
    - `DB_URL=jdbc:mysql://mysql_db:3306/medilabo_patient`
    - `DB_USERNAME=user`
    - `DB_PASSWORD=user`

6. Microservice Report Backend

- **Description :** Conteneurisation du microservice Report pour la génération des rapports de risques.
- **Image Docker :** Création d'une image Docker à partir d'une configuration Spring Boot (`openjdk:21-jdk-slim`).
- **Configuration :**
  - **Port :**
    - **Port exposé :** `8083:8080`
    - **Description :** Le port `8083` est mappé pour servir le microservice Report.
  - **Variables d'environnement :**
    - `REPORT_CONTAINER_NAME=report`
    - `BUILD_PATH_REPORT=./backend/report`
    - `REPORT_PORT=8083:8083`

7. Microservice Gateway Cloud (Spring Boot)

- **Description :** Conteneurisation du service de passerelle (Gateway) qui gère les requêtes entre le front-end et les microservices backend.
- **Image Docker :** Création d'une image Docker à partir d'une configuration Spring Boot (`openjdk:21-jdk-slim`).
- **Configuration :**
  - **Port :**
    - **Port exposé :** `8080:8080`
    - **Description :** Le port `8080` est utilisé pour centraliser les requêtes entrantes.
  - **Variables d'environnement :**
    - `GATEWAY_CONTAINER_NAME=gateway_spring_cloud`
    - `BUILD_PATH_GATEWAY=./backend/gateway`
    - `GATEWAY_PORT=8080:8080`

Avantages de la Dockerisation

Cohérence de l'Environnement : Assure que tous les environnements de développement, de test, et de production sont identiques, minimisant les problèmes liés aux différences d'environnement.

Portabilité : Les conteneurs Docker permettent de déployer l'application facilement sur n'importe quelle infrastructure compatible Docker.

Scalabilité : Facilite la montée en charge des services en permettant de déployer facilement plusieurs instances de chaque service.

5 Glossaire

Vocabulaire du Domaine Métiers

1. Présentation du projet

- **Application de gestion des patients** : Un logiciel conçu pour collecter, stocker et gérer les informations relatives aux patients, en particulier ceux atteints de diabète.
- **Architecture microservices** : Une approche de conception de logiciels où l'application est décomposée en petits services indépendants qui interagissent entre eux via des interfaces bien définies.
- **Visualisation** : Processus de consultation et d'affichage des informations ou données, généralement via une interface utilisateur.
- **Gestion** : L'action de manipuler, organiser et administrer les informations ou les ressources.
- **Évaluation** : Analyse et estimation des données pour déterminer leur importance ou leur état.

2. Spécifications fonctionnelles

- **Gestion des Comptes Utilisateurs** : Fonctionnalités liées à l'authentification et à la gestion des profils des utilisateurs de l'application.
- **Page de connexion** : Interface utilisateur permettant aux utilisateurs d'entrer leurs identifiants pour accéder à l'application.
- **Page de bord** : Page d'accueil de l'application où les utilisateurs sont accueillis après la connexion.
- **Page de déconnexion** : Interface permettant aux utilisateurs de se déconnecter de l'application.
- **Gestion des Comptes Clients** : Fonctionnalités permettant de visualiser, ajouter, modifier et gérer les informations des patients.
- **Visualisation des Informations Personnelles** : Affichage des données personnelles d'un patient telles que le nom, l'adresse, et les coordonnées.
- **Mise à jour des Informations Personnelles** : Modification des informations existantes dans le dossier d'un patient.

- **Ajout d'Informations Personnelles** : Intégration de nouvelles informations dans le dossier d'un patient.
- **Gestion des Notes Médicales** : Fonctions relatives à l'ajout, la modification, et la consultation des notes médicales des patients.
- **Vue historique du Patient** : Affichage de l'historique des informations et des notes médicales d'un patient.
- **Ajout de Notes** : Processus d'enregistrement de nouvelles observations médicales dans le dossier d'un patient.
- **Évaluation du Risque de Diabète** : Calcul et affichage du niveau de risque de diabète d'un patient basé sur ses données médicales et historiques.

3. User Stories et Critères d'acceptation

- **User Story** : Description simple et concise d'une fonctionnalité du point de vue de l'utilisateur, expliquant ce que l'utilisateur veut accomplir et pourquoi.
- **Critère d'acceptation** : Ensemble de conditions à remplir pour qu'une fonctionnalité soit considérée comme terminée et acceptée.
- **Donnée Initiale** : Les informations ou l'état de départ avant qu'une action soit effectuée.
- **Action** : L'opération que l'utilisateur effectue ou la réponse du système à une action.
- **Conséquence** : Le résultat ou l'effet de l'action sur l'application ou l'utilisateur.
- **Positif/Négatif** : Résultat attendu lorsque l'action est réalisée avec succès (positif) ou échoue (négatif).

Vocabulaire Technique

1. Spécifications techniques

1.1 Besoin API (Endpoint API)

- **Microservice Front-End Angular** : Composant de l'architecture du projet qui gère l'interface utilisateur et les interactions côté client, utilisant le framework Angular.
- **Microservice Gateway** : Service intermédiaire qui reçoit les requêtes du front-end, les transmet aux microservices appropriés, et renvoie les réponses au front-end.
- **Microservice Backend Patient** : Le microservice Backend Patient gère les informations personnelles des patients.
- **Microservice Follow-Note Backend** : Le microservice Follow-Note Backend gère les notes et suivis des patients.
- **Microservice Report Backend** : Le microservice Report Backend génère les rapports de risques pour les patients.

1.2 Solutions techniques

- **Spring Boot** : Utilisé pour simplifier le développement d'applications Java en fournissant des configurations prêtes à l'emploi et un support intégré pour de nombreuses fonctionnalités, telles que les contrôleurs REST, la gestion des dépendances, et l'intégration avec diverses bases de données et services.
- **Java** : Choix de la version Java la plus récente pour bénéficier des dernières fonctionnalités du langage, des améliorations de performance et des corrections de sécurité. Assure également la compatibilité avec Spring Boot.
- **Asynchrone et Parallélisme** : L'utilisation de traitements asynchrones et parallèles pour gérer les tâches gourmandes en ressources de manière plus efficace, en améliorant les temps de réponse et en optimisant les performances de l'application.
- **Docker pour la Conteneurisation** : Utilisé pour conteneuriser l'application, facilitant le déploiement, l'exécution, et la gestion des environnements de développement et de production. Les images Docker garantissent que l'application fonctionne de manière cohérente quel que soit l'environnement.

- **MongoDB** : Base de données orientée document qui stocke et retrouve les données sous forme de documents au format JSON ou XML, facilitant la gestion des données complexes.
- **GitHub pour le Versionnage du Code Source** : Utilisé pour le versionnage du code source, permettant de suivre l'historique des modifications, de gérer les branches et les merges, et de collaborer efficacement avec l'équipe de développement.
- **GitHub Actions pour CI/CD** : Choisi pour son intégration native avec GitHub, simplifiant le flux de travail de développement et de déploiement avec une interface conviviale, des workflows définis en YAML, et une gestion des conteneurs Docker via GitHub Container Registry (GHCR).
- **JaCoCo** : Mesure le pourcentage de code testé par les tests automatisés, garantissant une couverture adéquate.
- **Surefire** : Exécute les tests unitaires et d'intégration, fournissant des rapports détaillés sur les résultats des tests.

Green Code

Qu'est-ce que le Green Code ?

Le Green Code (approche de la programmation qui vise à réduire l'empreinte écologique des applications informatiques) désigne une approche de la programmation et du développement logiciel qui vise à réduire l'empreinte écologique des applications informatiques. Cela inclut l'optimisation des ressources utilisées (comme l'énergie, le stockage, la bande passante) et la minimisation des impacts environnementaux tout au long du cycle de vie d'un logiciel (ensemble des étapes de développement, déploiement, maintenance, et fin de vie d'un logiciel), de la conception à l'utilisation finale.

Le Green Code s'inscrit dans une démarche plus large de responsabilité environnementale (approche où les entreprises et développeurs prennent en compte l'impact de leurs actions sur l'environnement), où les développeurs et les équipes techniques cherchent à concevoir des systèmes non seulement performants, mais également respectueux de l'environnement.

Actions pour un Code Écoresponsable dans ce Projet

Pour rendre ce projet conforme aux principes du Green Code, voici plusieurs actions et bonnes pratiques (méthodes recommandées pour atteindre des objectifs spécifiques, souvent basées sur l'expérience et les standards de l'industrie) qui peuvent être mises en place :

1. **Optimisation des Requêtes et des Données**

    - **Réduction des Appels Réseaux :** Minimisez le nombre de requêtes réseau (demandes envoyées à un serveur pour récupérer des données) en utilisant des techniques comme le cache local (mécanisme pour stocker temporairement les données fréquemment demandées localement) ou la pagination (technique pour diviser de grandes quantités de données en pages plus petites) pour réduire la charge sur le réseau et les serveurs

(ordinateurs ou systèmes qui fournissent des données ou des services à d'autres ordinateurs).

- **Compression des Données :** Utilisez la compression des données (réduction de la taille des fichiers ou des données transmises pour économiser de la bande passante et du stockage) envoyées et reçues pour réduire la quantité de données transmises, diminuant ainsi l'énergie consommée par le réseau.
- **Optimisation des Requêtes SQL :** Évitez les requêtes SQL inefficaces (commandes mal structurées ou complexes envoyées à une base de données pour récupérer ou manipuler des données) pour limiter la consommation des ressources serveur.

2. **Efficacité du Code**

- **Code Efficace et Léger :** Évitez le code inutile ou redondant (parties de code qui répètent la même fonction ou effectuent des tâches inutiles) pour diminuer la charge CPU (unité centrale de traitement, responsable de l'exécution des instructions du programme) et la mémoire (ressource informatique utilisée pour stocker temporairement les données lors de l'exécution d'un programme) nécessaire à l'exécution du programme.
- **Optimisation de l'Utilisation de la Mémoire :** Réduisez la consommation de mémoire en gérant efficacement les objets (instances de classes en programmation, représentant des entités avec des attributs et des méthodes) et en nettoyant ceux qui ne sont plus utilisés (par exemple, via une bonne gestion des collecteurs de déchets en Java, mécanisme automatique pour libérer la mémoire occupée par des objets qui ne sont plus référencés).
- **Limitation de la Taille des Bibliothèques :** N'incluez que les bibliothèques (ensembles de code préécrit que les développeurs peuvent réutiliser) strictement nécessaires pour réduire la taille globale de l'application et donc la quantité de ressources nécessaires pour la maintenir et l'exécuter.

3. **Utilisation des Ressources Serveur**

    - **Scalabilité Dynamique :** Implémentez des systèmes qui permettent d'ajuster dynamiquement les ressources serveur (capacité de traitement et de stockage d'un serveur) en fonction de la demande (scalabilité horizontale/verticale, ajout ou retrait de ressources serveur pour répondre aux besoins) afin d'éviter de laisser des serveurs tourner inutilement à pleine capacité.
    - **Serveurs Basse Consommation :** Si possible, hébergez l'application sur des serveurs qui utilisent des énergies renouvelables (sources d'énergie qui se régénèrent naturellement, comme l'énergie solaire ou éolienne) ou des centres de données (installations où sont regroupés des serveurs et des systèmes de stockage) à faible impact environnemental.
4. **Gestion de l'Infrastructure Cloud**

    - **Choix de Fournisseurs Cloud Écologiques :** Optez pour des fournisseurs cloud (entreprises qui offrent des services informatiques via Internet) qui s'engagent dans des pratiques de gestion durable (approche visant à minimiser les impacts environnementaux tout en répondant aux besoins des utilisateurs), comme AWS avec ses initiatives d'énergies renouvelables, ou Google Cloud Platform avec ses data centers écoénergétiques (conçus pour minimiser la consommation d'énergie).
    - **Optimisation des Machines Virtuelles :** Utilisez des instances de machines virtuelles (ordinateurs simulés au sein d'un environnement physique, permettant de faire fonctionner plusieurs systèmes d'exploitation sur un même matériel) adaptées à la charge de travail pour éviter le surprovisionnement (allocation de plus de ressources que nécessaire, ce qui peut entraîner un gaspillage).

5. **Surveillance et Analyse Continue**
    - **Monitoring de la Consommation Énergétique :** Implémentez des outils de monitoring (systèmes de surveillance continue des performances et de l'utilisation des ressources) pour suivre la consommation énergétique de l'application en production (environnement où l'application est effectivement utilisée par les utilisateurs finaux), afin de pouvoir ajuster et optimiser les ressources en temps réel.
    - **Analyse de l'Impact Carbone :** Mettez en place des métriques (mesures quantitatives utilisées pour évaluer les performances ou l'efficacité d'un système) pour estimer et analyser l'empreinte carbone (quantité de gaz à effet de serre émis directement ou indirectement par une activité) associée au fonctionnement de l'application, en tenant compte des aspects comme la consommation d'énergie des serveurs, la transmission des données, etc.

6. **Adoption de Bonnes Pratiques de Développement**
    - **Revue de Code Axée sur l'Efficacité :** Lors des revues de code (processus où les développeurs examinent le code source écrit par leurs pairs pour en vérifier la qualité et la conformité), évaluez les modifications non seulement sur la qualité et la fonctionnalité, mais aussi sur leur impact potentiel en termes d'efficacité énergétique.
    - **Formation et Sensibilisation des Développeurs :** Sensibilisez l'équipe de développement aux principes du Green Code pour qu'ils intègrent systématiquement ces pratiques dans leur travail quotidien.

Conclusion

L'adoption du Green Code dans ce projet permettra non seulement d'améliorer l'efficacité des ressources (utilisation optimale des ressources matérielles et logicielles disponibles) et de réduire les coûts opérationnels (dépenses liées au fonctionnement continu de l'application), mais aussi de contribuer à la réduction de l'empreinte carbone globale de l'application (quantité totale de dioxyde de carbone et autres gaz à effet de serre émis pendant le cycle de vie de l'application). Cette approche est essentielle pour aligner le développement logiciel avec les objectifs de durabilité environnementale.