Documentation de l'API SmartPulse

API de monitoring cardiaque - Documentation technique

Introduction

L'API SmartPulse est conçue pour une application de monitoring cardiaque. Elle permet aux patients d'envoyer des données de pouls en temps réel provenant d'un capteur (comme un Arduino ou similaire), tandis que les médecins peuvent prescrire des activités, définir des seuils de BPM (battements par minute), monitorer les sessions, et recevoir des notifications. Les utilisateurs sont divisés en deux rôles principaux : **PATIENT** et **MEDECIN**.

Fonctionnalités Principales

- **Authentification** : Enregistrement et login via JWT (JSON Web Token). Tous les endpoints protégés nécessitent un header `Authorization: Bearer <token>`.
- **Gestion des Utilisateurs** : Profils patients et médecins, assignation de médecins aux patients.
- **Seuils (Thresholds)** : Définition de limites BPM pour alerter en cas de dépassement.
- **Notifications** : Système de notifications push pour les utilisateurs (non lues/lues, marquage comme lues).
- **Activités** : Médecins prescrivent des activités aux patients, avec durée et monitoring.
- **Sessions Cardiaques** : Démarrage/arrêt de sessions de mesure, envoi de données BPM, summaries et historiques.
- **WebSockets** : Pour le streaming en temps réel des données BPM (utilisé pour des graphiques live sur le frontend).

Informations Générales

- **Base URL** : `/api` (tous les endpoints commencent par `/api/v1/` ou `/api/` selon le controller, mais la plupart sont sous `/api/v1/` pour versioning).
- **Authentification** : JWT requis pour la plupart des endpoints (sauf login/register). Utilisez `Authorization: Bearer <token>` dans les headers.
- **CORS** : Activé pour tous les origines (`*`) dans certains controllers (e.g., Notifications).
- **Erreurs Communes** :
  - 401 Unauthorized : Token invalide ou manquant.

  - 404 Not Found : Ressource non trouvée (e.g., patient/session inexistante).
  - 400 Bad Request : Données invalides dans le body.
  - 500 Internal Server Error : Erreurs serveur (rare, mais loggez-les).
- **Formats** : Toutes les requêtes/réponses sont en JSON. Dates au format ISO (e.g., `"2023-10-01"`).
- **WebSockets** : Utilisez une bibliothèque comme SockJS ou Stomp pour vous connecter. Endpoint de connexion : `/ws` (non explicitement défini dans le code, mais implicite via Spring WebSocket). Abonnez-vous à des topics comme `/topic/pulse` pour recevoir les données BPM en temps réel.
- **Conseils pour le Frontend** :
  - Utilisez Axios ou Fetch pour les appels HTTP.
  - Pour WebSockets, intégrez STOMP.js pour gérer les subscriptions.
  - Gérez les notifications avec un badge (comptez les non-lues) et un panneau pour les lister/marquage.
  - Pour les sessions cardiaques, affichez un graphique live via Chart.js, mis à jour via WebSocket.
  - Assurez-vous que les IDs (e.g., patientId) sont des strings ou longs selon le contexte (vérifiez les DTOs).

L'API est construite avec Spring Boot, utilisant des repositories JPA pour la persistence (base de données non spécifiée, assumez PostgreSQL ou similaire).

Catégories d'Endpoints

1. Authentification

Endpoints pour login et enregistrement. Pas d'auth requise.

POST /api/auth/login

Description : Authentifie un utilisateur (patient ou médecin) et retourne un JWT pour les appels subséquents. Utilisé pour connecter l'utilisateur au système.

Request Body (JSON) :

```
{
        "mail": "user@example.com",
        "password": "password123"
}
```

Response (200 OK) :

```
{
```

```
        "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Autres Infos : Si credentials invalides, retourne 401. Le token expire (durée non spécifiée, assumez 1h ; refresh si nécessaire via un autre endpoint non implémenté).

POST /api/auth/register/medecin

Description : Enregistre un nouveau médecin. Crée un utilisateur avec rôle MEDECIN et associe un profil Medecin.

Request Body (JSON) :

```
{
        "mail": "doc@example.com",
        "password": "password123",
        "nom": "Doe",
        "prenom": "John",
        "dateNaissance": "1980-01-01",
        "specialite": "Cardiologue"
}
```

Response (200 OK) :

```
{
        "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Autres Infos : Vérifie l'unicité du mail. Retourne 400 si mail existe.

POST /api/auth/register/patient

Description : Enregistre un nouveau patient. Crée un utilisateur avec rôle PATIENT et associe un profil Patient. Le medecinId est optionnel (null si non assigné).

Request Body (JSON) :

```
{
        "mail": "patient@example.com",
        "password": "password123",
        "nom": "Smith",
        "prenom": "Jane",
```

```
        "dateNaissance": "1990-05-15",
        "medecinId": 1
}
```

Response (200 OK) :

```
{
        "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Autres Infos : Si medecinId fourni, vérifie qu'il existe. Sinon, le patient peut être assigné plus tard.

POST /api/auth/register-test

Description : Endpoint de test pour créer un médecin hardcoded (pour dev/debug). Non pour production.

Request Body : Aucun.

Response (200 OK) : Texte brut, e.g., "Médecin créé → login : doc@exemple2.com / 123456"

Autres Infos : À désactiver en prod.

2. Gestion des Utilisateurs (Profils)

Endpoints pour gérer les profils. Requière JWT.

GET /api/user/me

Description : Récupère le profil de l'utilisateur connecté (patient ou médecin). Utilisé pour afficher les infos personnelles.

Request Body : Aucun.

Response (200 OK) :

```
{
        "UserId": 1,
        "mail": "user@example.com",
```

```
        "nom": "Doe",
        "prenom": "John",
        "dateNaissance": "1980-01-01",
        "role": "MEDECIN",
        "specialite": "Cardiologue",
        "medecinId": null,
        "id": 1
}
```


Autres Infos : Retourne 404 si user non trouvé.

GET /api/user/my-doctor

Description : Pour un patient connecté, récupère les infos de son médecin assigné. Utilisé pour afficher le contact du médecin.

Request Body : Aucun.

Response (200 OK) : Même format que UserProfileResponse (ci-dessus). 204 No Content si aucun médecin assigné.

Autres Infos : Uniquement pour patients.

GET /api/user/my-patients

Description : Pour un médecin connecté, liste tous ses patients. Utilisé pour le dashboard médecin.

Request Body : Aucun.

Response (200 OK) :


```
[
        {
                "UserId": 2,
                "mail": "patient@example.com",
                "nom": "Smith",
                "prenom": "Jane",
                "dateNaissance": "1990-05-15",
                "role": "PATIENT",
                "specialite": null,
                "medecinId": 1,
                "id": 2
```

```
        }
]
```

Autres Infos : Uniquement pour médecins.

PUT /api/user/update

Description : Met à jour le profil de l'utilisateur connecté (nom, prénom, etc.). Adapte selon le rôle.

Request Body (JSON) : Même format que UserProfileResponse, mais partiel si besoin.

```
{
        "nom": "NewNom",
        "prenom": "NewPrenom",
        "dateNaissance": "1980-01-01",
        "specialite": "Neurologue"  // Ignore pour patients
}
```

Response (200 OK) : UserProfileResponse mis à jour.

Autres Infos : Ne permet pas de changer mail/password ici.

3. Seuils (Thresholds)

Endpoints pour gérer les seuils BPM par patient. Requière JWT (typiquement médecin).

POST /api/v1/thresholds/set/{patientId}

Description : Définit ou met à jour les seuils min/max BPM pour un patient. Utilisé pour alerter si le pouls sort des bornes pendant une session.

Request Body : Aucun (paramètres dans URL/query).

Query Params : `min` (int, e.g., 60), `max` (int, e.g., 100).

Response (200 OK) :

```
{
        "id": 1,
        "patient": { /* Patient entity */ },
```

```
        "bpmMin": 60,
        "bpmMax": 100
}
```

Autres Infos : Crée si non existant. Throw exception si patient non trouvé.

GET /api/v1/thresholds/patient/{patientId}

Description : Récupère les seuils d'un patient.

Request Body : Aucun.

Response (200 OK) : Même que ci-dessus. 404 si non trouvé.

4. Notifications

Endpoints pour gérer les notifications. Requière JWT. CORS activé.

GET /api/v1/notifications/user/{userId}

Description : Liste toutes les notifications d'un utilisateur (triées par date descendante). Utilisé pour le panneau de notifications.

Request Body : Aucun.

Response (200 OK) :

```
[
        {
                "id": 1,
                "message": "Nouvelle activite prescrite...",
                "createdAt": "2023-10-01T12:00:00",
                "isRead": false
        }
]
```

Autres Infos : UserId peut être patientId ou doctorId (string).

GET /api/v1/notifications/user/{userId}/unread

Description : Liste uniquement les notifications non lues. Utilisé pour le badge de notification (comptez la longueur).

Request Body : Aucun.

Response (200 OK) : Même format que ci-dessus, filtré sur isRead=false.

PATCH /api/v1/notifications/{id}/read

Description : Marque une notification spécifique comme lue.

Request Body : Aucun.

Response (200 OK) : Vide. 404 si non trouvé.

POST /api/v1/notifications/user/{userId}/read-all

Description : Marque toutes les notifications d'un utilisateur comme lues. Utilisé quand l'utilisateur ouvre le panneau.

Request Body : Aucun.

Response (200 OK) : Vide.

5. Activités

Endpoints pour gérer les activités prescrites. Requière JWT.

POST /api/v1/activities/create

Description : Crée une nouvelle activité prescrite par un médecin à un patient. Notifie le patient.

Request Body (JSON) :

```
{
        "title": "Marche rapide",
        "description": "30 min par jour",
        "patientId": "patient123",
        "doctorId": "doctor456",
```

```
        "durationInMinutes": 30
}
```

Response (200 OK) :

```
{
        "id": 1,
        "title": "Marche rapide",
        "description": "30 min par jour",
        "patientId": "patient123",
        "doctorId": "doctor456",
        "durationInMinutes": 30,
        "completed": false
}
```

GET /api/v1/activities/{id}

Description : Récupère une activité par ID.

Request Body : Aucun.

Response (200 OK) : Même que ci-dessus. 404 si non trouvé.

POST /api/v1/activities/start-activity

Description : Démarre une session liée à une activité (pour patient). Programme une fin automatique après duration. Notifie potentiellement le médecin.

Request Body (JSON) :

```
{
        "patientId": "patient123",
        "activityId": 1
}
```

Response (200 OK) : Long (sessionId, e.g., 42).

Autres Infos : Refuse si activité déjà complétée. Utilise un scheduler pour arrêter après duration.

PUT /api/v1/activities/{id}

Description : Met à jour une activité. Notifie le patient.

Request Body (JSON) : Même que ActivityRequest.

Response (200 OK) : Activité mise à jour.

DELETE /api/v1/activities/{id}

Description : Supprime une activité. Notifie le patient.

Request Body : Aucun.

Response (200 OK) : Vide. 404 si non trouvé.

PATCH /api/v1/activities/{id}/close

Description : Clôture une activité (marque comme completed). Notifie le patient.

Request Body : Aucun.

Response (200 OK) : Message texte, e.g., "Activité clôturée avec succès." 404 si non trouvé.

GET /api/v1/activities/patient/{patientId}

Description : Liste les activités d'un patient (triées par ID descendante).

Request Body : Aucun.

Response (200 OK) : Liste d'activités.

GET /api/v1/activities/doctor/{doctorId}

Description : Liste les activités prescrites par un médecin.

Request Body : Aucun.

Response (200 OK) : Liste d'activités.

6. Sessions Cardiaques et Données de Pouls

Endpoints pour gérer les sessions de mesure. Requière JWT.

POST /api/v1/cardiac/start

Description : Démarre une session cardiaque générique (non liée à une activité). Utilisé pour des mesures ad-hoc.

Request Body (JSON) :

```
{
        "patientId": "patient123"
}
```

Response (200 OK) : Long (sessionId).

POST /api/v1/cardiac/receive

Description : Reçoit des données BPM du capteur (e.g., Arduino). Sauvegarde si session active, vérifie seuils, diffuse via WebSocket.

Request Body (JSON) :

```
{
        "bpm": 75,
        "status": "OK"
}
```

Response (200 OK) : Vide (no-content). Log si pas de session active.

Autres Infos : Appelé fréquemment (e.g., toutes les secondes). Déclenche notifications si seuils dépassés.

POST /api/v1/cardiac/stop

Description : Arrête manuellement la session active.

Request Body : Aucun.

Response (200 OK) : Message texte, e.g., "Mesure arrêtée avec succès."

GET /api/v1/cardiac/session/{sessionId}/data

Description : Récupère toutes les données BPM d'une session (triées par timestamp).

Request Body : Aucun.

Response (200 OK) :

```
[
        {
                "id": 1,
                "bpm": 75,
                "timestamp": "2023-10-01T12:00:00",
                "session": { /* Session ref */ }
        }
]
```

GET /api/v1/cardiac/session/{sessionId}/summary

Description : Récupère un résumé (moyenne, max, min) d'une session.

Request Body : Aucun.

Response (200 OK) :

```
{
        "averageBpm": 72.5,
        "maxBpm": 80,
        "minBpm": 65,
        "totalPoints": 10
}
```

GET /api/v1/cardiac/patient/{patientId}/sessions

Description : Liste les sessions d'un patient (triées par startTime descendante).

Request Body : Aucun.

Response (200 OK) : Liste de CardiacSession. 204 si vide.

7. WebSockets

Pas d'endpoint HTTP, mais configuration pour streaming.

Description Générale : Utilisé pour push en temps réel des données BPM pendant une session. Le frontend s'abonne pour recevoir des mises à jour live (e.g., pour un graphique).

Endpoint de Connexion : `/ws-cardiac` (via WebSocket handshake).

Topics :

- `/topic/pulse` : Reçoit les données BPM en temps réel.
  - Message Reçu (JSON) :

```
{
        "bpm": 75,
        "status": "OK"
}
```

Utilisation Frontend : Utilisez STOMP client :

```
const stompClient = new StompJs.Client({
        brokerURL: 'ws://your-api-url/ws'
});
stompClient.activate();
stompClient.subscribe('/topic/pulse', (message) => {
        const data = JSON.parse(message.body);
        // Mettre a jour graphique avec data.bpm
});
```

Autres Infos : Déclenché par `/api/v1/cardiac/receive`. Pas d'auth WebSocket par défaut (ajoutez si besoin via interceptor).

8. Test

Endpoint de test.

GET /test

Description : Test basique pour vérifier JWT.

Request Body : Aucun.

Response (200 OK) : "Test OK – si tu vois ça, le JWT marche !"

Annexes

— **Sécurité** : Tous les endpoints protégés vérifient le rôle via JWT. Assurez-vous que les patients ne peuvent pas accéder aux endpoints médecins (implémenté via @AuthenticationPrincipal).
— **Intégration Capteur** : Le capteur (e.g., Arduino) doit POST sur `/api/v1/cardiac/receive` avec auth (peut-être un token patient fixe).
— **Améliorations Potentielles** : Ajoutez pagination pour listes longues (e.g., sessions). Implémentez refresh token. Ajoutez plus de validations (e.g., BPM > 0).
— **Contact** : Si besoin de clarifications, basez-vous sur les codes fournis.