ProSanteConnect – POC Copier-coller

Guide d'Implémentation

« Henix / TRA ANS »

ANS | Agence du numérique en santé
9, rue Georges Pitard
75015 PARIS

Sommaire

1 Contexte du projet et objectifs du document

Ce document a pour objectif de décrire l'architecture technique et logicielle utilisée par le POC Copier-Coller. De manière symbolique et schématique, ce document décrit les différents logiciels, leurs interrelations et leurs interactions. Le modèle d'architecture, ne décrit pas ce que doit réaliser le système mais plutôt comment il doit être conçu.

L'agence souhaite proposer une solution permettant à des fournisseurs de service (FS), intégrés ou non dans un même Système d'information, mais ayant en commun le recours à ProSanteConnect comme fournisseur d'identité, de pouvoir partager des données dans le cadre d'un « contexte ProSanteConnect ».

Fonctionnellement, cette solution permettrait aux applications l'implémentant de disposer d'un service de « copier-coller », l'authentification ProSanteConnect jouant à la fois le rôle de contrôle de l'autorisation d'accès aux données partagées, et celui de l'indexation de celles-ci (un Professionnel de Santé n'ayant accès qu'aux données qu'il ait lui-même partagées).

L'objectif prioritaire de ce projet est de démontrer la faisabilité technique d'une solution de partage de contexte adossée à ProSanteConnect, en proposant un exemple d'implémentation en Open Source, de nature à inspirer des fournisseurs de service.

Il s'est donc agi de proposer une solution légère et la plus portable possible, pour la rendre implémentable dans des applications existantes avec un degré de complexité réduit.

Pour réaliser cet objectif, un webservice de type API REST a été implémenté devant un serveur de cache Redis, ainsi que deux proto applications, dont l'usage se limite à permettre de visualiser facilement le comportement générable côté client.

Chaque fournisseur de service potentiellement intéressé pouvant avoir ses propres caractéristiques d'implémentation, le choix a été fait dans le cadre de ce POC d'une architecture distribuée permettant d'isoler les comportements :

- Le partage de contexte proprement dit (API avec méthodes GET/PUT sur un serveur de cache, contrôle d'accès, contrôle de format).
- L'utilisation côté FS de ce partage (génération des requêtes API, implémentation dynamique du partage côté navigateur).

Les sources des quatre applications développées sont disponibles sur un dépôt distant Github (Prosanteconnect).

2 Architecture technique

2.1 Fonctionnement de l'application

2.1.1 Fonctionnement général

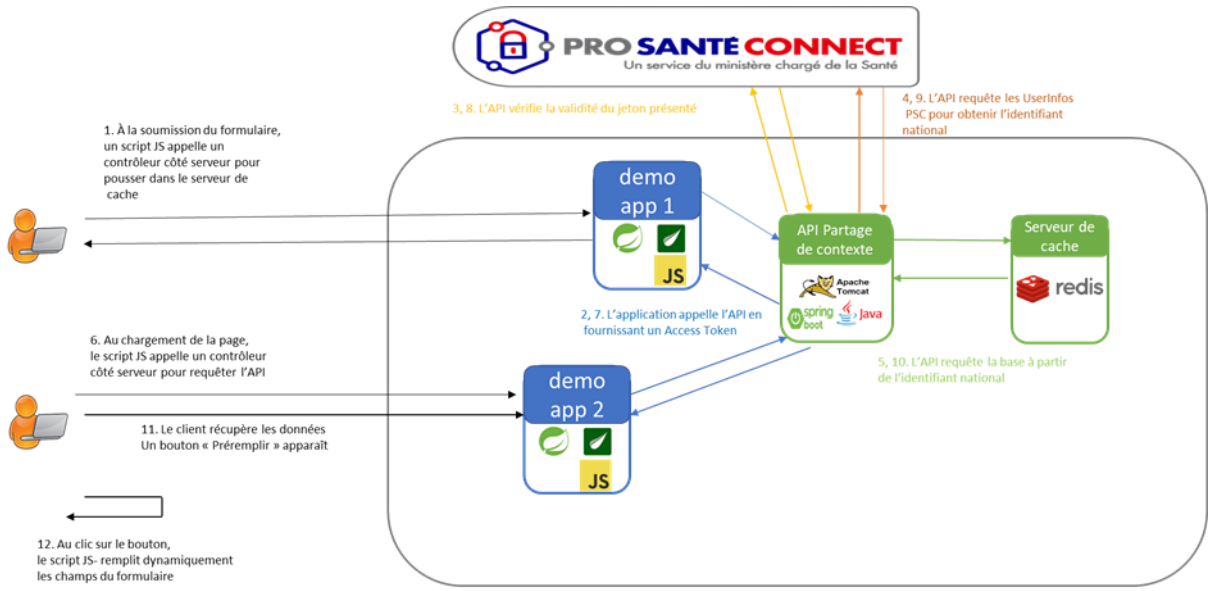
La solution retenue pour l'API Proxy est la suivante :

- Un serveur de cache Redis, avec rétention des données de 15 minutes et sans persistance.

- Une API devant ce serveur, développée en Java / Spring Boot selon le standard OpenAPI. Cette API

- Deux applications de démonstration, développées en Java / HTML / JavaScript. Cependant, la solution proposée n'est pas adhérente au langage de programmation côté serveur, car elle repose sur l'exécution d'un script JS et un appel à une API REST. Entre les deux, la logique côté serveur (ajout de l'access token ProSanteConnect) est relativement aisée à implémenter dans le langage de son choix.

2.1.2 Schéma d'architecture virtualisé



Récapitulatif des flux

2.1.3 Fonctionnement détaillé

2.1.3.1 Serveur de cache

Le cache est assuré par une base de données clé-valeur en mémoire pour des raisons de performance. Le choix s'est porté sur un serveur Redis conteneurisé à partir de l'image officielle Redis pour les raisons suivantes :

- La vitesse d'exécution, notamment pour les accès en lecture (les données contextuelles disponibles doivent être accessibles quasiment sans latence au chargement d'une page côté client pour une expérience utilisateur fluide).

- La possibilité de configurer au niveau du serveur la durée de rétention des objets, ce qui permet de maintenir un cache souple et adapté à l'expérience utilisateur. Dans le cadre de ce POC, la durée de configuration a été configurée à **15 minutes**.

- La stabilité applicative, Redis étant la solution leader sur le cache clé-valeur en mémoire.

2.1.3.2 API de partage de contexte

Les opérations sur le serveur sont opérées à partir d'une API, seul point d'entrée sur le cache.

L'API a été développée en Java / Spring Boot, en s'appuyant sur la spécification OpenAPI 3.0. L'API est décrite dans un fichier ***psc-copier-coller-api.yaml*** présent dans les sources. Ce fichier décrit le modèle objet utilisé par l'API ainsi que l'ensemble des endpoints exposés. Il peut être utilisé pour générer facilement un client dans différents langages.

La connexion avec le serveur de cache est gérée via un fichier de Spring properties. Dans le cadre de ce POC, le déploiement étant effectué en conteneur Docker via l'orchestrateur Nomad, ces propriétés sont déclarées dans un ConsulTemplate embarqué dans le descripteur de job Nomad présent à la racine du projet.

À ce jour, les opérations permises par l'API sont les suivantes :

- Obtenir les données contextuelles associées à un Professionnel de Santé.

- Créer / mettre à jour ces données contextuelles.

- Obtenir l'ensemble des formats de données acceptées par l'API, sous forme d'une liste schémas respectant la spécification JSON-schema.

- Obtenir l'un de ces formats en particulier.

Deux niveaux de sécurité ont été implémentés :

- L'accès aux endpoints permettant la manipulation des données contextuelles est soumis à la présentation par le client d'API d'un jeton d'authentification (access token) ProSanteConnect valide. Dans le cadre de ce POC il s'agit d'un jeton ProSanteConnect ***Bac à sable***.

- Lors de la création / mise à jour de données contextuelles, un contrôle de conformité est effectué sur le format des données fournies en entrée. Celles-ci doivent en effet respecter le format défini par un descripteur JsonSchema présent dans un répertoire de l'application.

Cette mécanique a pour objectif de prendre en charge la définition d'un standard de façon centralisée, au niveau de l'API, et de décharger les clients des contrôles de cohérence et de conformité.
En l'état actuel du POC, ces schémas doivent être disponibles sur le dépôt de sources pour être chargés au lancement de l'application. Une évolution éventuelle pourrait consister en l'ajout d'une couche de persistance des schémas et l'implémentation de méthodes permettant d'ajouter ou de modifier un JSON-schema.

2.1.3.3 Les applications de démonstration

Deux applications sont livrées en sus de l'API et du serveur de cache, à de pures fins de démonstration. Elles ne relèvent pas à proprement parler du concept qu'on souhaite vérifier par l'intermédiaire de ce POC, mais elles concourent à illustrer un exemple de comportement implémentable.

À ce titre, elles sont volontairement rudimentaires : une page d'accueil avec connexion ProSanteConnect, pour l'obtention d'un jeton d'authentification valide, et une simple page de formulaire, sans traitement métier, destinée à permettre l'implémentation du comportement côté client.

Le choix a été fait de les développer en SpringBoot pour déléguer à Spring Security la gestion de l'authentification ProSanteConnect, mais n'importe quel langage est compatible, dès lors que l'application est en mesure de prendre en charge l'authentification PSC.

Le fonctionnement implémenté est le suivant : deux contrôleurs s'exécutant côté serveur permettent de propager l'access token requis par l'API et d'appeler les endpoints de l'API (en GET pour le coller, en PUT pour le copier). Ces contrôleurs sont appelés côté client par un script javascript, disponible dans un fichier dédié, appelé depuis la page de formulaire.

Ce script s'appuie en outre sur un mapping décrit dans un fichier au format json pour faire le lien entre des éléments du DOM, comme des champs de formulaires, avec des attributs des données contextuelles obtenues de l'API.

En troisième partie de ce document, un guide d'implémentation explicite le rôle de chaque fichier et le moyen de les intégrer à une application.

2.1.3.4 Description du workflow

La mécanique de l'application est double (copier / coller), chaque fournisseur de service ayant la liberté d'implémenter un versant, l'autre ou les deux.

Les deux applications de démonstration jouent le rôle de client de l'API. Elles sollicitent celle-ci en fournissant un jeton d'authentification ProSanteConnect valide.

Lorsqu'elle est appelée, l'API procède selon les étapes suivantes :

- La validité du jeton d'authentification est vérifiée auprès de ProSanteConnect. En cas de jeton invalide, un code http

401.UNAUTHORIZED est retourné. En l'absence de jeton, ou en présence d'un jeton mal formé, un code http ***403.FORBIDDEN*** est retourné.

- À l'étape du copier, une vérification de la conformité des données soumises avec un format de référence est opérée. Ce format de référence est spécifié dans la requête entrante. Cette vérification permet de disposer de formats de référence connus des applications implémentant cette solution, et de les dispenser de l'implémentation des contrôles de conformité.

- L'API procède ensuite aux opérations sur le serveur de cache.

Les applications utilisant la solution de partage de contexte sont libres de l'implémentation des appels à l'API. Pour simplifier cette étape, le choix a été fait dans le cadre de ce POC de développer un script JS portable dans tous les environnements. Les méthodes de ce script appellent des contrôleurs internes de l'application appelante. Ce sont ces contrôleurs qui ont pour charge de s'interfacer avec ProSanteConnect pour la génération du jeton d'authentification, puis d'appeler les endpoints de l'API.

2.1.4 Test sur l'environnement POC – ProSanteConnect

Il est possible de tester la solution sur l'environnement ProSanteConnect POC, aux adresses suivantes :

- https://prosanteconnect.share-context.henix.asipsante.fr/
- https://application-1.henix.asipsante.fr/
- https://application-2.henix.asipsante.fr/

Les prérequis au bon fonctionnement de l'application sont les suivants :

- L'utilisateur doit être en mesure de fournir une identité enregistrée auprès de ProSanteConnect, dans le cas présent une identité de test (ProSanteConnect ***Bac à sable***).

3 Guide d'implémentation

Plusieurs possibilités s'offrent au moment de choisir une implémentation de cette solution.

En premier lieu, le présent POC peut être conçu comme une source d'inspiration générale sur ce qu'il est possible de mettre en place, pour la réimplémenter en intégralité, conformément aux besoins spécifiques rencontrés.

Il est également envisageable de tester rapidement ce qu'il est possible de faire sur une application existante en utilisant l'API du POC et en n'implémentant que la partie cliente. Cette seconde option est décrite ci-dessous.

3.1 Pré-requis

Au préalable, un client doit être enregistré auprès du fournisseur d'identité ProSanteConnect. Les credentials de ce client devront être fournis, aussi bien par l'API si celle-ci est réimplémentée, que par les applications la consommant.

3.2 L'implémentation côté applications clientes

3.2.1 Architecture du dépôt de sources

L'ensemble des composants du POC sont sourcées dans un dépôt unique, comportant cinq dossiers : un répertoire pour chaque composant (le serveur de cache, l'API et chacune des deux applications de démonstration) et un répertoire de ressources.

Le packaging de la solution ayant été prévu, dans le cadre du POC, pour un déploiement sous forme de conteneurs Docker pris en charge par l'orchestrateur Nomad sur un cloud public, les répertoires de l'API et des applications de démonstration contiennent les sources du projet ainsi que les fichiers permettant d'assurer le build et le déploiement : un pom.xml dans le cas des projets Java/SpringBoot, et surtout un Dockerfile et un descripteur de job Nomad.

Il n'y a pas de Dockerfile dans le répertoire du serveur de cache, celui-ci étant déployé à partir de l'image officielle. Seul le descripteur de job Nomad est présent.

À la racine du projet se trouve un fichier waypoint.hcl, destiné à être lu par le logiciel Waypoint pour piloter la build et le déploiement par Nomad des quatre composants applicatifs.

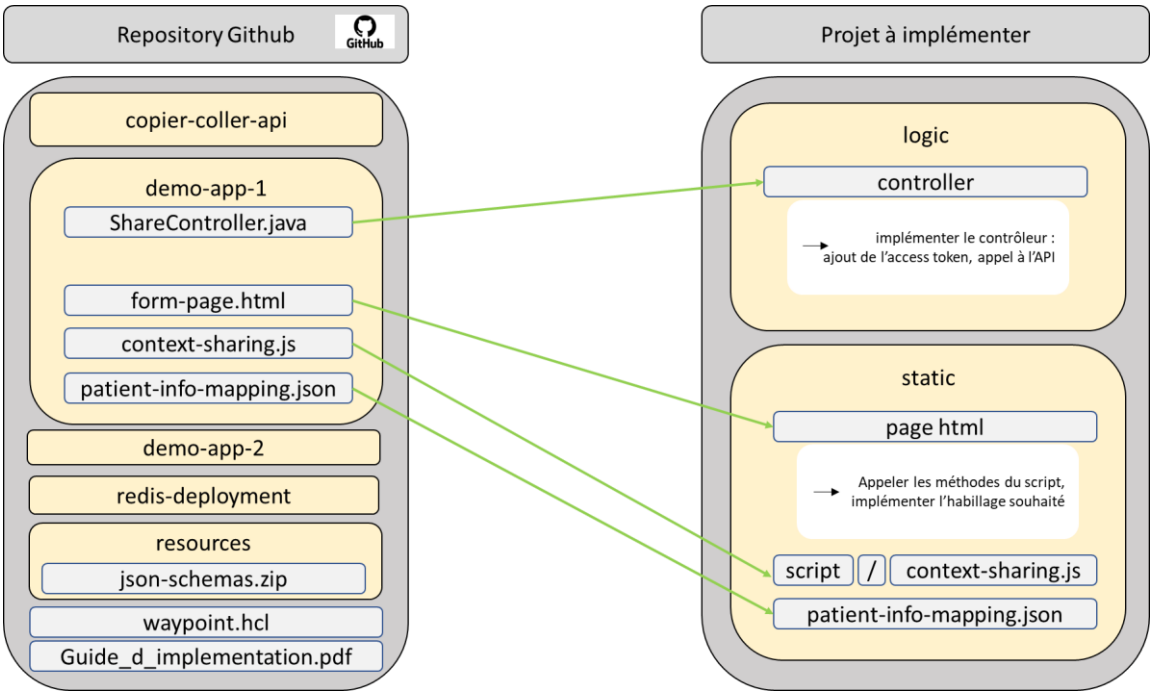
Dans un autre cadre de déploiement, ces fichiers sont donc sans objet. Il importe cependant de prendre en compte que les paramètres configurables (URLs de connexion, credentials ProSanteConnect, etc.) le sont via les descripteurs Nomad. Il appartiendrait alors à l'implémenteur de passer ces paramètres de configuration par la voie qui lui paraîtra adaptée à son contexte de déploiement.

Enfin, un répertoire ***resources*** contient des exemples de JSON-schemas et surtout une archive au format zip contenant certains d'entre eux. Dans le cadre d'un déploiement via Nomad, le contenu de cette archive est dézippé et copié dans un répertoire accessible du conteneur de l'API. En se passant de Nomad, il faudrait également s'assurer de mettre cette archive à disposition de l'API : elle *contient l'ensemble des JSON-schemas considérés comme valides*, et contre lesquels l'API effectue *le contrôle de conformité des données contextuelles* fournies lors d'une opération de partage.

Les endpoints de l'API étant exposés publiquement, il est possible de tester l'implémentation d'une application cliente tout en s'appuyant sur l'API et le serveur de cache déployés sur la plateforme POC de l'ANS.

3.2.2 Principes généraux d'implémentation

3.2.2.1 Schéma d'implémentation général



L'une des applications de démonstration peut être utilisée comme modèle.

Dans les sources du code applicatif, elle contient un contrôleur s'exécutant côté serveur, ShareController.java, contenant les deux méthodes permettant de copier et coller.

À la racine des sources statiques, elle contient un fichier ***patient-info-mapping.json*** et un répertoire ***script*** dans lequel est placé un fichier intitulé ***context-sharing.js***.

Dans le répertoire ***templates*** est placé le code des pages html.

Le script ***context-sharing.js*** contient la logique métier via trois méthodes permettant d'obtenir, coller et copier des données contextuelles.

Le fichier ***patient-info-mapping.json*** est un exemple de fichier de mapping qui est utilisé par les méthodes du script pour assurer le lien entre des éléments du DOM et les données contextuelles.

3.2.2.2 Les contrôleurs côté serveur

Côté serveur, deux méthodes sont implémentées au sein du contrôleur ShareController : ***putContextInCache()*** et ***getContextInCache()***. Ces deux méthodes jouent le rôle de proxy pour transmettre à l'API des requêtes générées côté client, en propageant un header supplémentaire ***AUTHORIZATION*** contenant le jeton d'authentification ProSanteConnect.

Le maintien de ce jeton côté serveur correspondant à une exigence du fournisseur d'identité ProSanteConnect, il est nécessaire d'implémenter ces deux méthodes, qui n'ont pas d'autre logique métier.

3.2.2.3 Les fichiers de référence côté client

Côté client, deux (a minima) fichiers sont à ajouter : le script Javascript qui contient la logique métier, et un ou plusieurs fichiers de mapping.

Le script ***context-sharing.js*** contient trois méthodes :

- Une pour requêter l'API pour obtenir les données contextuelles associées au Professionnel de Santé connecté, s'il y en a de disponibles.

- Une pour remplir les champs d'un formulaire avec les valeurs retournées par la méthode précédente, en s'appuyant sur un fichier de mapping fourni en paramètre de la méthode (ici ***patient-info-mapping.json***) : c'est le « coller ».

- Une pour extraire les données du formulaire, et les pousser vers l'API, en s'appuyant là encore sur un fichier de mapping fourni en paramètre de la méthode : c'est le « copier ».

Ce script est conçu pour être portable, et de pouvoir être exécuté avec des fichiers de mapping différents suivant les formulaires de l'application : le choix du mapping de référence s'effectue à l'appel de la méthode, dans le document html.

Le (ou les) fichier(s) de mapping adaptés au contexte de l'application. Leur format doit respecter le même modèle que le fichier d'exemple : une liste d'attributs au format JSON permettant de mapper à gauche des id d'éléments du DOM et à droite des attributs des données contextuelles reçues de l'API, en respectant le modèle objet défini par le schéma.

3.2.2.4 L'implémentation côté vue

Pour « copier », il convient d'ajouter à la méthode de soumission du formulaire un appel à la méthode ***putInCache()*** du script en lui fournissant le nom du schema de référence sur lequel l'API effectuera son contrôle de conformité et le chemin relatif du fichier de mapping utilisé.

Pour « coller », les deux opérations se déroulent en deux temps : au chargement de la page, une requête Ajax permet de récupérer les données disponibles auprès de l'API. Si aucune donnée n'est disponible, le comportement de la page n'est pas altéré. S'il en existe, un bouton de pré remplissage apparaît. L'utilisateur a alors la possibilité de coller le contenu de ces données dans le formulaire.

La requête initiale doit s'effectuer au chargement de la page. Elle est donc appelée par la méthode ***getFromCache()*** appelée dès le chargement du Body.

La méthode ***fillForm()***, elle doit être appelée au clic sur le bouton de pré remplissage, en fournissant en paramètre le chemin relatif du fichier de mapping requis pour ce formulaire.