

Questionneur

Documentation d'Architecture Technique

Version 1.1 du 09/12/2025

Visas

Rôle	Nom	Date visa
Auteur	J. Le Fur	18/07/2022
Valideur Nat System	Déborah Kassabi	
Valideur ABM	David Vitte	

Historique des versions

Version	Date	Auteur	Description
1.0	18/07/2022	J. Le Fur	Version initiale du document
1.1	09/12/2025	A. Banckaert	Mise à jour montée de version Angular / Java

Table des matières

1. Introduction.....	4
1.1. Objectif du document	4
1.2. Contexte	4
2. Architecture de l'application	5
2.1. Vue d'ensemble de l'architecture.....	5
2.2. Structure du projet	7
2.3. Stack technique	5
2.3.1. Backend.....	5
2.3.2. Frontend.....	5
2.3.3. Base de données.....	6
2.4. Sécurité	7
3. Modularité et composition des livrables.....	8
3.1. Socle standard de l'Agence et livrables.....	8
3.2. Organisation du développement d'un Service REST.....	8
3.3. Organisation en package	9
4. Service RESTFul.....	10
4.1. API RESTFul niveau 2.....	10

1. Introduction

1.1. Objectif du document

L'Agence a défini une architecture cible pour l'ensemble de ses projets dans son Guide-architecture-v2.docx. Ce document vient en complément de ce guide : il n'a pas vocation à se substituer au guide car le projet QUEST suit le guide d'architecture de l'Agence.

Il illustre et précise certains points avec des exemples du projet QUEST.

Ce guide n'a pas vocation à être lu sans le Guide. Il est supposé que le lecteur dispose d'une version du Guide.

1.2. Contexte

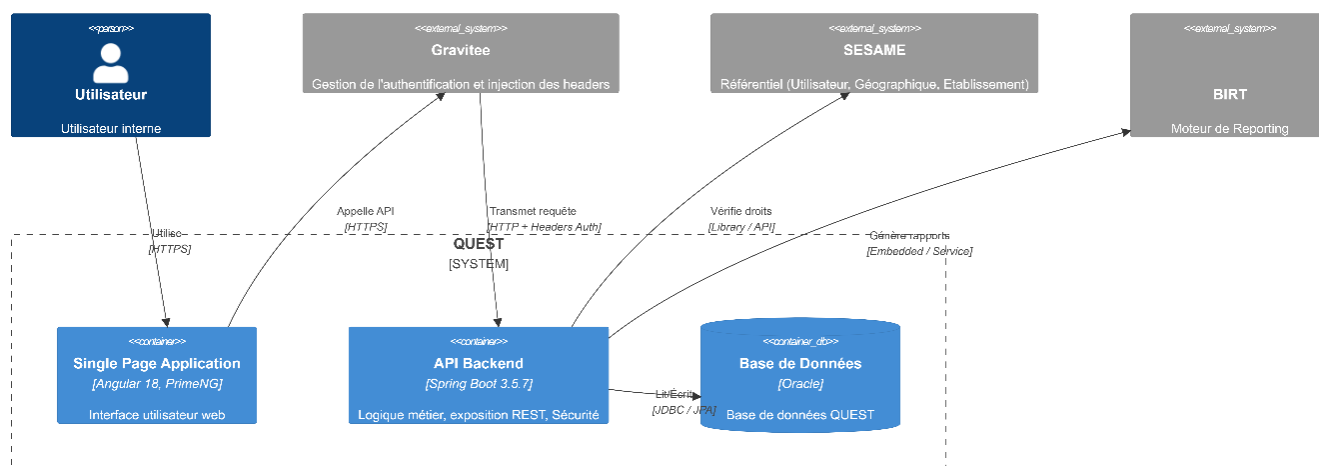
L'application Questionneur est déployée dans un environnement défini par l'Agence. L'accès se fait via une redirection depuis le portail SIPG. Après cette redirection, l'utilisateur s'authentifie à l'aide d'un jeton délivré par le serveur d'autorisation Keycloak et accède à l'API Gravitee, qui vérifie ce jeton transmis dans l'en-tête HTTP. L'authentification aboutit si les identifiants sont corrects dans la base de données Sésame. Une fois toutes les vérifications effectuées, la requête atteint le backend qui peut alors interroger la base de données POPP ainsi que le serveur SFTP pour récupérer les documents scannés.

De plus, l'application utilise des webservices Sesame pour la récupération des informations de l'utilisateur connecté, ainsi que pour les établissements et les données géographiques.

2. Architecture de l'application

2.1. Vue d'ensemble de l'architecture

Diagramme de Conteneurs - Système QUEST



2.2. Stack technique

2.2.1. Backend

Technologie	Version	Usage
Java	21	Langage de programmation
Spring Boot	3.5.7	Framework applicatif
Spring Data JPA	(via Spring Boot)	Abstraction ORM
Hibernate	(via Spring Boot)	Implémentation JPA
MapStruct	1.6.3	Mapping Objet-Objet (DTO <-> Entity)
SpringDoc OpenAPI	2.8.14	Documentation API (Swagger)
Log4j2	2.25.2	Journalisation

2.2.2. Frontend

Technologie	Version	Usage
Angular	18.2.4	Framework Frontend

PrimeNG	17.18.10	Bibliothèque de composants UI
Keycloak Angular	16.0.1	Intégration Sécurité
Keycloak-js	25.0.2	Librairie keycloak front
RxJS	7.8.1	Programmation réactive
Jest	29.7.0	Tests unitaires
PrimeIcons	7.0.0	Librairie PrimeNG d'icônes
PrimeFlex	3.3.1	Librairie PrimeNG pour le responsive design
Typescript	5.5.4	

2.2.3. Base de données

Technologie	Version	Usage
Oracle	12c	Driver ojdbc11

La base de données utilisée est celle de POPP. Cependant, pour des raisons de sécurité, un schéma particulier a été mis en place pour quest, que voici :

Nom de la table	Description
POD_INTEGRATION	Table des intégrations
POD_INTERROGATIONS	Table de suivi des interrogations
POD_ITR_RFU	Table de liens entre les interrogations et les refus
POD_PARAM	Table de paramétrage
POD_PJ	Table de pièce jointe
POD_PJ_PV	Table des PV de décès
POD_PJ_DEMANDEUR	Table de demande de refus
POD_REFUS	Table de refus
POD_TYPE_DEMANDE	Table de type de demande
POS_FONCTIONS	Table de fonctions
POS_MOTIF_REFUS	Table de référentiel des motifs de refus
POS_PROFILS	Table de profils

2.1. Structure du projet

Module	Description
quest -parent	POM parent gérant les versions et la configuration globale.
quest-data-services	Couche d'accès aux données principales (Repositories, Entities).
quest-rest-services	Couche d'exposition REST (Controllers, DTOs). Point d'entrée de l'application backend.
quest-front	Code source de l'application Angular.
quest-integration	Module indépendant contenant les ressources ainsi que le process de construction du livrable

2.2. Sécurité

La sécurité de l'application repose sur une authentification déléguée (via un SSO/Keycloak en amont).

Backend : Configuration personnalisée (RestSecurityConfig) utilisant un HeaderAuthProvider. L'application semble attendre des headers spécifiques (injectés par une gateway ou un proxy d'authentification) pour identifier l'utilisateur.

Frontend : Utilisation de keycloak-angular et keycloak-js pour la gestion des tokens et de la session utilisateur côté client.

Stateless : L'API est configurée en mode STATELESS (pas de session HTTP serveur).

3. Modularité et composition des livrables

L'Agence a défini son socle technique : il est basé sur une architecture de type API REST et repose sur les technologies Angular et Java.

3.1. Socle standard de l'Agence et livrables

Le projet POPP respecte le socle de l'Agence : les livrables du projet POPP s'intégreront à l'architecture mutualisée de l'Agence :

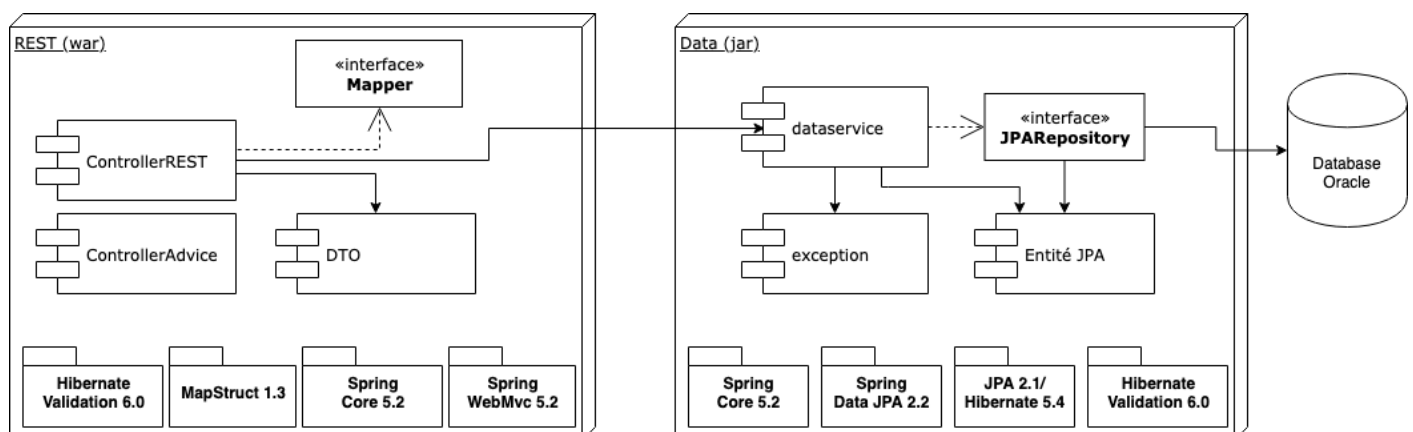
- un serveur Tomcat avec un JDK 21 pour le module du back end livré sous forme de war.
- un serveur Apache pour héberger et diffuser les applications fronts end Angular du projet
- un portail SIPG qui assure l'authentification centralisé des utilisateurs et le lancement des applications Angular
- un API Manager (Gravitee) permettant une gestion centralisée et protégée de l'accès aux services du Back.
- Un serveur d'autorisation Keycloak qui est couplé à l'API Manager et qui permet la fédération des mécanismes d'authentification.

A ce titre, le projet POPP mettra à disposition plusieurs war, plusieurs zip Angular et un document d'installation permettant de déployer et configurer les applications. Le guide d'installation indiquera également la configuration attendue de l'API Man.

3.2. Organisation du développement d'un Service REST

Pour le projet POPP, le développement d'un service REST sera décomposé en deux projets Java :

- le projet REST qui produit un war
- le projet DataService qui produit un jar utilisé et inclus dans le war.



Cette approche permet d'assurer l'isolation de chaque couche et d'assurer une plus grande pérennité de la partie métier en la découplant de la couche communication.

De plus, si nécessaire le jar du projet DataService est susceptible d'être utilisé dans un autre projet REST si cela semble opportun.

3.3. Organisation en package

Ce point n'est pas abordé dans le guide d'architecture mais dans le guide de développement.

L'organisation des packages Java pour le projet sera la suivante :

Pour le war quest-rest-service-1.X.X.war (partie REST du projet)

```
com.abm.quest.rest.dto  
com.abm.quest.rest.controller  
com.abm.quest.rest.mapper
```

Pour le jar correspondant (la partie dataservice du projet)

```
com.abm.quest.data.repository  
com.abm.quest.data.model  
com.abm.quest.data.service  
com.abm.quest.data.core.exception  
com.abm.quest.data.core.service  
com.abm.quest.data.core.util.converter
```

4. Service RESTFul

4.1. API RESTFul niveau 2

L'application QUEST est conforme à la demande de l'Agence : elle respecte les préconisations du niveau 2 comme demandé dans le chapitre 6.1.

Dans ce niveau l'API utilise les verbes HTTP. soit :

- `POST` pour la création d'une nouvelle ressource
- `GET` pour la recherche d'une ressource par son id ou d'une collection avec des critères passés dans la query string .
- Le projet QUEST utilisera principalement la méthode `PATCH` pour les mises à jour : les mises à jour partielles seront favorisées et seront dans la mesure du possible, couplées aux objets *Embeddable*. Par exemple la mise à jour de l'état d'une intégration pourra être : `PATCH integrations/{id}/etat` et ne transmettra que la partie de l'état dans le body.
- `DELETE` pour les suppressions si cela est nécessaire

QUEST utilise des codes de réponse adaptées :

- 200 si tout s'est bien passé
- 201 lors d'une création avec un `POST`
- 204 lors de la suppression par un `DELETE`
- 206 si le `GET` renvoie une liste paginée
- 400 pour les erreurs utilisateurs autres que 401, 403 et 404
- 401 si l'utilisateur n'est pas authentifié
- 403 si l'utilisateur n'a pas accès à la ressource
- 404 si la ressource n'a pas été trouvée
- 500 pour toutes les erreurs techniques