|  |
| --- |
| ENSPY |
| Authentification Handshake |
| Rapport de conception de la QRAPI |

|  |
| --- |
| Nghogué Taptué Franck Roddier  05/07/2025  Sous la supervision de Pr. DJOTIO Thomas |

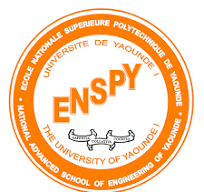


Table des matières

[**INTRODUCTION :** 2](#_Toc203215898)

[**OBJECTIFS :** 3](#_Toc203215899)

[**ARCHITECTURE GENERALE :** 4](#_Toc203215900)

[**MODELE DE DONNEES :** 5](#_Toc203215901)

[**4.1 Tables principales :** 5](#_Toc203215902)

[**ENDPOINTS REST :** 7](#_Toc203215903)

[**DIAGRAMMES UML :** 8](#_Toc203215904)

[**5.1 Diagramme de Classes :** 8](#_Toc203215905)

[**5.2 Diagramme de package** 9](#_Toc203215906)

[**5.2 Diagramme de Cas d’Utilisation :** 10](#_Toc203215907)

[**5.2 Diagrammes de Séquence :** 11](#_Toc203215908)

[**5.3 Diagramme d’état-transition :** 13](#_Toc203215909)

[**5.3 Diagrammes d’activités :** 14](#_Toc203215910)

[**5.5 Diagramme de Déploiement :** 16](#_Toc203215911)

[**SECURITE ET CONFORMITE :** 17](#_Toc203215912)

[**PLANS DE TESTS :** 18](#_Toc203215913)

[**DEPLOIEMENT ET ENVIRONNEMENT :** 19](#_Toc203215914)

[**SCENARIOS D’UTILISATION :** 20](#_Toc203215915)

[**CONCLUSION :** 21](#_Toc203215916)

# **INTRODUCTION :**

Ce document présente le rapport de conception et de développement de l'API **QRAPI**, conçue pour la génération, la validation, et la traçabilité de **QR codes sécurisés**, intégrant des données relatives à des courses (chauffeur/client). L’API s’intègre dans une architecture sécurisée utilisant JWT, SHA-256 et une base NoSQL ScyllaDB (compatible Cassandra).

# **OBJECTIFS :**

* **Génération de QR codes signés** avec les données d’une course.
* **Vérification sécurisée** des QR codes scannés via signature JWT.
* **Traçabilité** des opérations via enregistrement d’un historique.
* **Protection des accès** via un système d’authentification avec tokens JWT.
* **Architecture modulaire et scalable**, prête pour une montée en charge.

# **ARCHITECTURE GENERALE :**

* Spring Boot pour la structure REST.
* Contrôleur : QRCodeController
* Service métier : ScanService
* Outils de sécurité : JwtUtil, JWT signé HS256
* Base de données : ScyllaDB, avec tables :
  + qr\_data (informations course)
  + qr\_hash (hash signé)
  + history (traces de scan)
* Outils : ZXing (QR), JJWT (JWT), Spring Security, Docker

# **MODELE DE DONNEES :**

## **4.1 Tables principales :**

**Tables ScyllaDB :**

* qr\_data (

id uuid PRIMARY KEY,

client\_id bigint,

driver\_id bigint,

trip\_id bigint,

location text,

hour text,

date text,

city text,

country text,

fournisseur text

);

* qr\_hash (

id uuid PRIMARY KEY,

hash text,

qr\_data\_id uuid

);

* history (

id uuid PRIMARY KEY,

client\_id bigint,

driver\_id bigint,

trip\_id bigint,

location text,

supplier text,

hour text,

date text,

city text,

country text

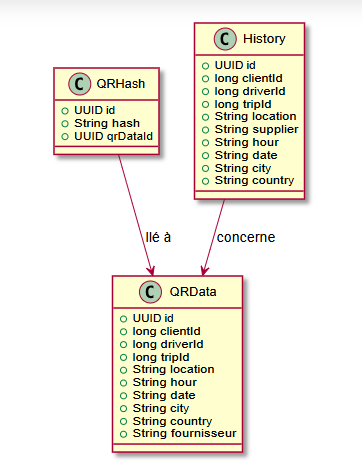
);

# **ENDPOINTS REST :**

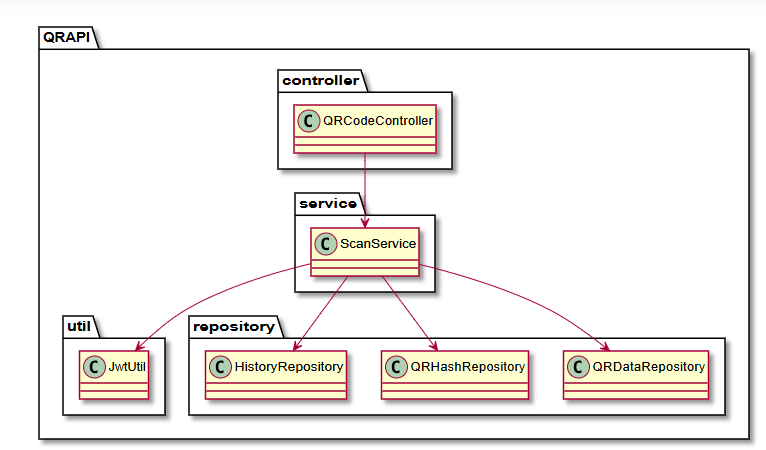
| **Endpoint** | **Méthode** | **Authentification** | **Description** |
| --- | --- | --- | --- |
| /api/reserve | POST | ❌ | Réserver un token JWT (non inclus dans code) |
| /api/qr/generate | POST | ✅ | Générer un QR code sécurisé (image PNG) |
| /api/qr/scan | POST | ✅ | Scanner et valider un QR code, enregistrer l’historique |

# **DIAGRAMMES UML :**

## **5.1 Diagramme de Classes :**

**

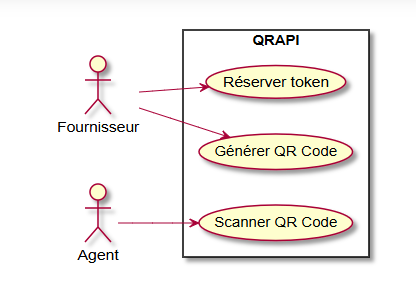
## **5.2 Diagramme de package**



## **5.2 Diagramme de Cas d’Utilisation :**

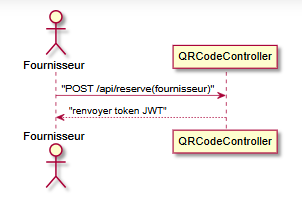
**Acteur principal** : Fournisseur/Agent  
**Cas** :

* Réserver token
* Générer un QR
* Scanner un QR

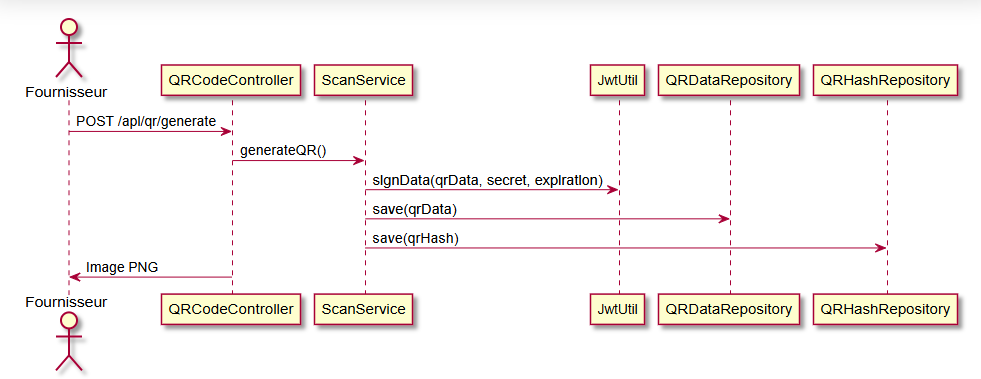


## **5.2 Diagrammes de Séquence :**

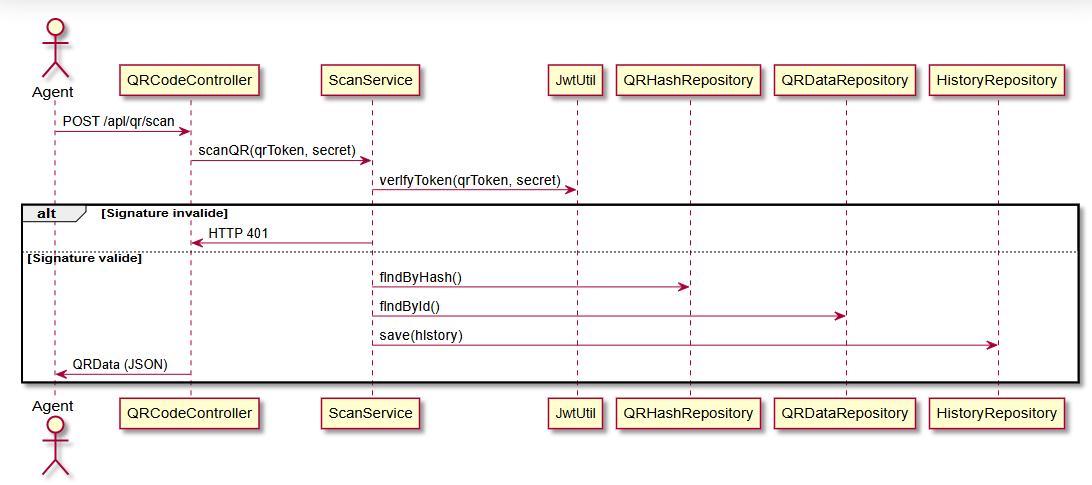
**Acteurs** : Fournisseur/Agent → API  
**Description** : Interaction entre l’utilisateur et le contrôleur pour générer un QR.



Réserver token

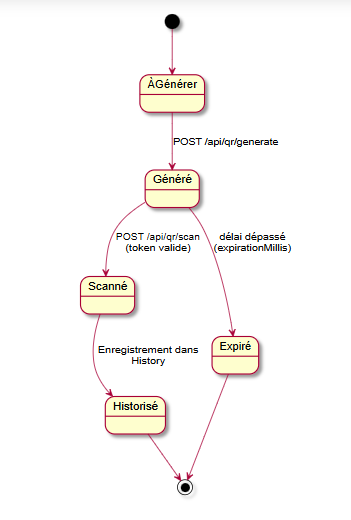


Générer QR code



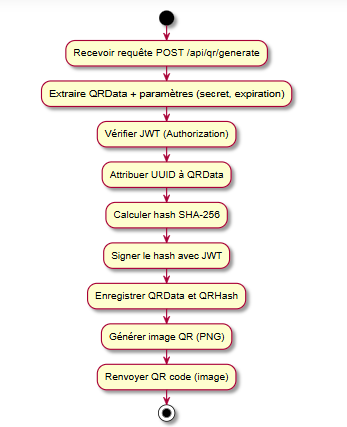
Scanner QR code

## **5.3 Diagramme d’état-transition :**

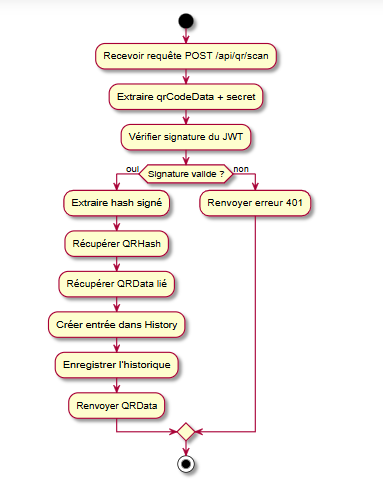


Cycle de vie d’un QR Code au sein de l’API

## **5.3 Diagrammes d’activités :**



Générer QR Code

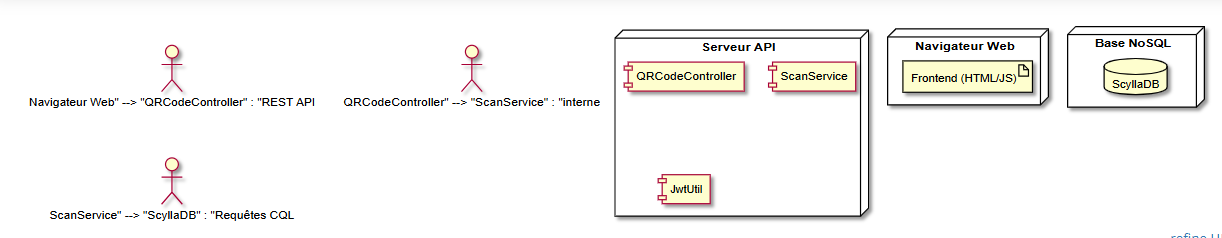


Scanner QR Code

## **5.5 Diagramme de Déploiement :**

**Composants** :

* Application Spring Boot (QRAPI)
* Base ScyllaDB/Cassandra
* Navigateur utilisateur



# **SECURITE ET CONFORMITE :**

* 🔐 **JWT** obligatoire pour /generate et /scan
* 🔏 **Hash SHA-256** intégré dans le QR code, signé
* 🕒 **Expiration des tokens** configurable
* ❌ **Pas de données sensibles exposées**
* ✅ CORS configuré finement

# **PLANS DE TESTS :**

* ✅ **Unitaires** : JwtUtil, ScanService, encodage
* ✅ **Intégration** : contrôleur ↔ service ↔ DB
* ✅ **Charge** : stress tests sur /generate et /scan
* ✅ **Sécurité** : injection, falsification, signature incorrecte
* ✅ **Suivi d’erreurs** : code 401/400 si signature invalide ou token expiré

# **DEPLOIEMENT ET ENVIRONNEMENT :**

* 📦 **Langage** : Java 17+
* ⚙️ **Build** : Maven (spring-boot-starter-\*, spring-data-cassandra)
* 🛢️ **Base** : ScyllaDB (ou Cassandra local)
* 🐳 **Docker** recommandé pour Cassandra
* 🧱 application.properties (ou bean de configuration) avec spring.data.cassandra.\*
* 🔐 Token via /api/reserve (JWT HS256, 32+ caractères)

# **SCENARIOS D’UTILISATION :**

* **Génération de QR**
  + Acteur : Fournisseur
  + Conditions : JWT valide
  + Résultat : QR signé, image PNG, enregistré en DB
* **Scan de QR**
  + Acteur : Agent
  + Conditions : QR valide et JWT valide
  + Résultat : Données de course affichées, enregistrement dans history

# **CONCLUSION :**

Le projet **QRAPI** constitue une solution robuste pour la sécurisation des échanges via **QR codes signés**, en particulier dans un contexte de transport ou de logistique. L’architecture modulaire, le respect des principes SOLID, la scalabilité via ScyllaDB, et les mécanismes de traçabilité garantissent un système prêt pour l’usage en production.