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

|  |
| --- |
| Nghogué Taptué Franck Roddier  13/07/2025  Sous la supervision de Pr. DJOTIO Thomas |

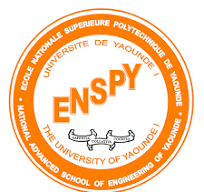


Table des matières

[**INTRODUCTION :** 2](#_Toc203312640)

[**I.** **ANALYSE DES EXIGENCES :** 3](#_Toc203312641)

[**I.1 Exigences fonctionnelles :** 3](#_Toc203312642)

[**I.2 Exigences non-fonctionnelles :** 3](#_Toc203312643)

[**II.** **ANALYSE ET CONCEPTION UML :** 4](#_Toc203312644)

[**II.1 Diagramme de Classes :** 4](#_Toc203312645)

[**II.2 Diagramme de package** 5](#_Toc203312646)

[**II.3 Diagramme de Cas d’Utilisation :** 6](#_Toc203312647)

[**II.4 Diagrammes de Séquence :** 7](#_Toc203312648)

[II.4.1 Réserver un token : 7](#_Toc203312649)

[II.4.2 Générer code QR : 8](#_Toc203312650)

[II.4.3 Scanner code QR : 9](#_Toc203312651)

[**II.5 Diagramme d’état-transition :** 10](#_Toc203312652)

[**II.6 Diagrammes d’activités :** 11](#_Toc203312653)

[**II.7 Diagramme de Déploiement :** 13](#_Toc203312654)

[**III.** **ARCHITECTURE GENERALE :** 14](#_Toc203312655)

[**IV.** **ENDPOINTS REST :** 15](#_Toc203312656)

[**V.** **SECURITE ET CONFORMITE :** 16](#_Toc203312657)

[**VI.** **PLANS DE TESTS :** 17](#_Toc203312658)

[**VII.** **DEPLOIEMENT ET ENVIRONNEMENT :** 18](#_Toc203312659)

[**VIII.** **SCENARIOS D’UTILISATION :** 19](#_Toc203312660)

[**CONCLUSION :** 20](#_Toc203312661)

# **INTRODUCTION :**

Depuis l’essor, d’Internet, puis des nouvelles technologies, et avec la popularisation des techniques de numérisation, il est de plus en plus urgent de mettre au point de nouveaux systèmes qui permettent la gestion « moderne » des transactions/échanges. Les services échangés reposent désormais sur des « tickets de service », des « billets », « factures », « reçus », que l’on peut qualifier de « documents ». Cependant, en même temps que les nouvelles technologies ont émergées de nouvelles menaces, l’usurpation d’identité, le vol d’information confidentielles, les arnaques … toutes regroupées sur la bannière du terme « cybercriminalité ». Les « documents » qui servent de garantie pour l’échange de services sont alors menacés de détournement et/ou de falsification, à des fins malveillantes. Notre projet « Authentification de documents Handshake » (ou QRAPI) a vu le jour dans le cadre de la lutte contre ces méfaits. Son but est de permettre la vérification de l’authenticité de différents « documents », ainsi que leur intégrité.

# **ANALYSE DES EXIGENCES :**

## **I.1 Exigences fonctionnelles :**

Notre système vise à permettre la vérification de l’intégrité et l’authenticité des documents. Pour que ce but soit atteint, il est nécessaire qu’il puisse assurer les fonctions suivantes :

* **Générer des codes QRs :** ceux-ci seront ajoutés aux documents
* **Valider les codes QRs :** les informations résultantes du scan du code QR seront vérifiées.
* **Conserver une trace des validations :** il faut que les différentes validations de code QR soient retraçables

## **I.2 Exigences non-fonctionnelles :**

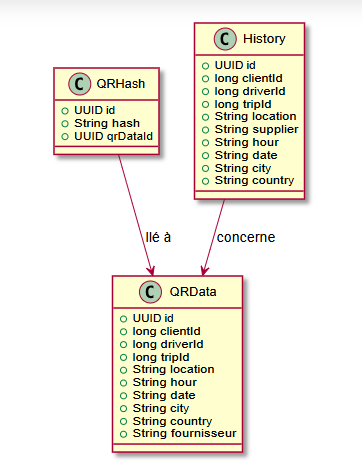
Il s’agira ici de recenser les spécifications sur le système (l’API) utiles pour son bon fonctionnement :

* **Authentification des fournisseurs :** il faut que chaque fournisseur (chaque application tiers) qui utilise l’API puisse générer sa clé d’API qui lui permettra d’utiliser le système
* **Hashage et chiffrement des codes QR :** le code QR ne doit pas contenir de texte clair et compréhensible de tous (pour améliorer la sécurité)
* **Liberté du choix d’une clé :** il faut que chaque fournisseur choisisse sa propre clé secrète qui sera utilisée pour la génération et la validation de codes QR

# **ANALYSE ET CONCEPTION UML :**

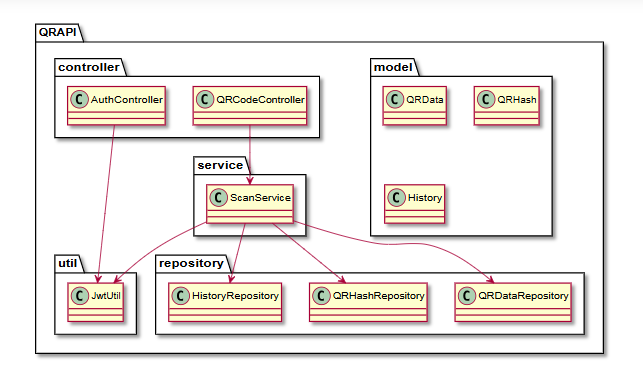
## **II.1 Diagramme de Classes :**

L’on peut distinguer 3 classes, que nous utiliserons (manipulerons) tout au long de notre projet :

* **QRData :** qui stocke les données d’un « document », nous avons choisi un format client/chauffeur car il illustre facilement le modèle bénéficiaire/distributeur de service. Les localisations spécifiées représentent le lieu de génération du document qu’elle représente.
* **QRHash :** stocke le hash des données d’un QRData, ainsi qu’une clé étrangère pour la QRData associé
* **History :** permet de conserver une trace des QRData (donc des documents) qui ont été « validés ». Elle présente la même structure que QRData. Mais les localisations spécifiées représentes le lieu de la validation du document représenté par QRData**

## **II.2 Diagramme de package**

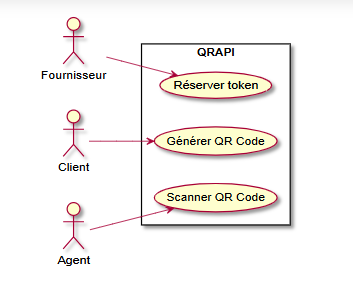
Ici la représentation de la répartition des fichiers du projet QRAPI, ainsi que les interactions entre eux (les interactions du type « qui utilise qui ») :



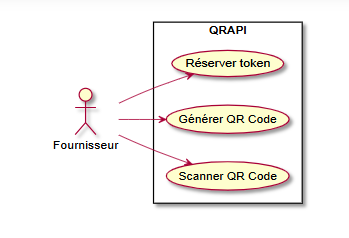
## **II.3 Diagramme de Cas d’Utilisation :**

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

* **Réserver token :** Le fournisseur exécute une requête à l’API et reçoit un token qui lui servira pour ses futurs usages de l’API
* **Générer un QR :** Le client, par le biais de l’application du fournisseur, exécute une requête, avec en paramètres les données de la transaction/du service garantit par son « document » à l’API et reçoit l’image du QR Code à insérer dans son « document »
* **Scanner un QR :** L’agent scanne un QR code sur un document et utilise le résultat du scan pour faire une requête à l’API, par le biais de l’application du fournisseur.



Par la suite, nous remarquerons que les rôles Client, et Agent, ne sont présents que l’application du fournisseur. Ainsi on peut mettre le diagramme de cas d’utilisation sous cette forme (parce que c’est le fournisseur qui réalise « réellement » ces cas d’utilisation) :

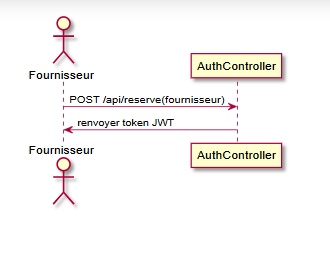


## **II.4 Diagrammes de Séquence :**

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

### II.4.1 Réserver un token :

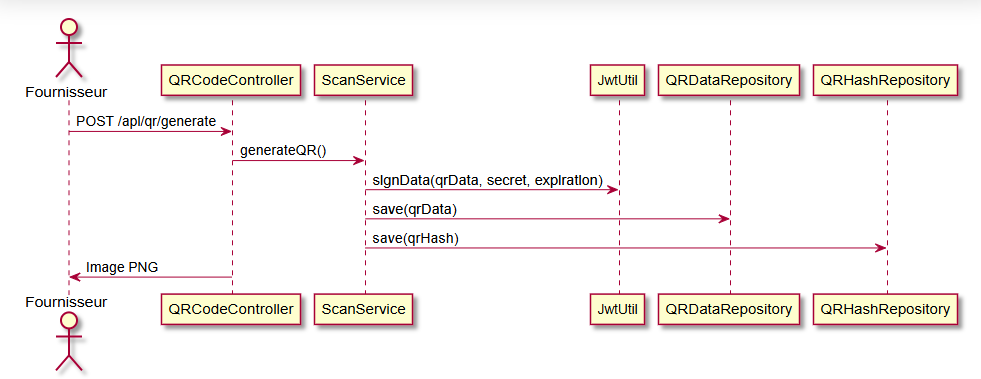
Le fournisseur fournit en paramètre d’une requête post /api/reserve son nom et se voit retourner un token, qui lui sera utile pour toute les autres cas d’utilisation de l’API.



Réserver token

### II.4.2 Générer code QR :

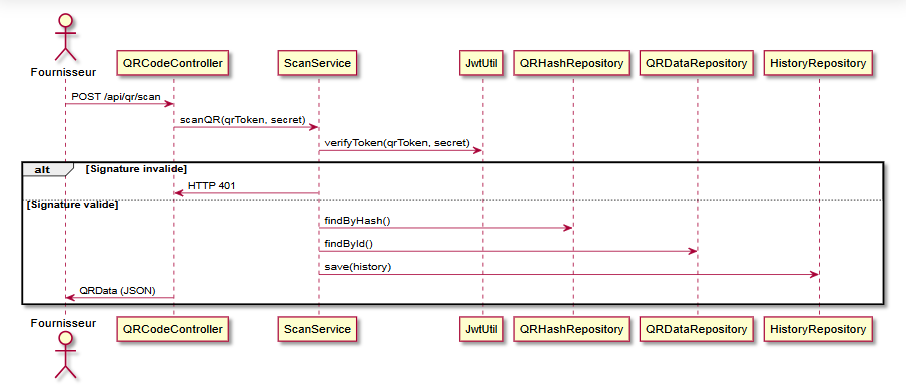
Le fournisseur exécute une requête POST /api/qr/generate avec en paramètre sa clé secrète (secret), et le temps d’expiration de son document (expirationMillis), et les données du « document » (sous le format QRData). Une entité QRData est crée à partir de là, pendant que son contenu est hashé sous une entité QRHash, puis chiffrée sous forme de token, en incluant le temps d’expiration, qui sera converti en code QR. Au final une image PNG est retournée au fournisseur, le QR Code, à insérer dans le « document ».



Générer QR code

### II.4.3 Scanner code QR :

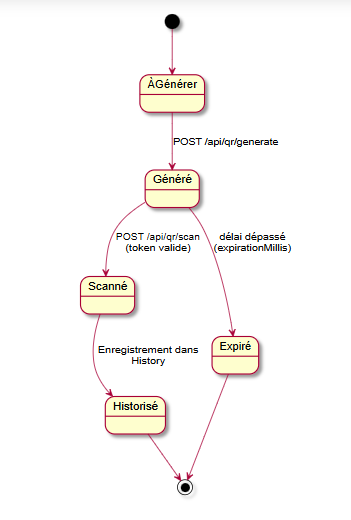
Le fournisseur scanne le code QR d’un document, récupère le chiffré contenu dans celui-ci, et le passe, avec sa clé secrète et les paramètres d’historique (la localisation du scan), en paramètre à l’API, qui s’occupe de vérifier le chiffré, ainsi que sa validité, en cas d’échec (token invalide ou expiré) retourne une http-401, et en cas de succès, retrouve le hash en déchiffrant le token, avant de retrouver l’entité QRHash correspondante. A partir de celle-ci on retrouve la QRData. Puis on utilise les informations de QRData pour compléter et stocker l’historique, et on renvoie au fournisseur QRData.



Scanner QR code

## **II.5 Diagramme d’état-transition :**

Le cycle de vie d’un QRCode (qui sera inclus dans un document) passe par plusieurs stades :

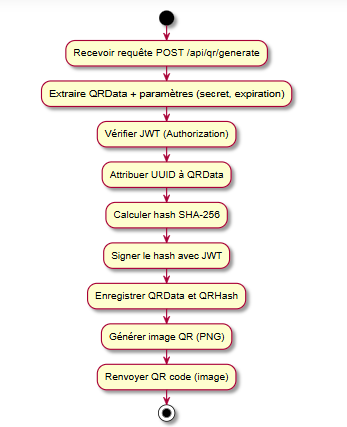
* **A Générer :** Le « document » qui garantit le service est établit (ou presque) et on dispose de toutes les informations nécessaires au QRCode
* **Généré :** Le fournisseur a utilisé /api/qr/generate pour chiffrer les informations du document et les mettre au format QR Code
* **Expiré :** Une fois que le délai spécifié par expirationMillis est dépassé
* **Scanné :** Une fois que le fournisseur le passe en paramètre (du moins son contenu) à l’API avec /api/qr/scan
* **Historisé :** Lorsqu’il est valide et son historique associée a été établie
* 

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

## **II.6 Diagrammes d’activités :**

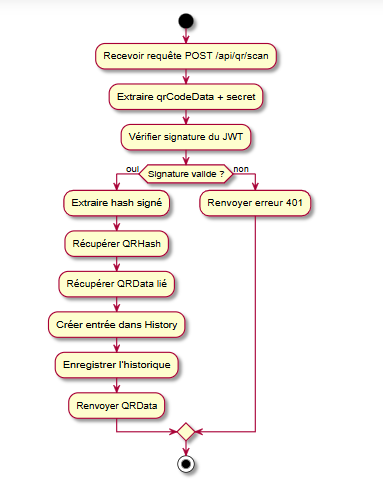
Ici les différentes activités que le système (la QRAPI) effectue au long de son fonctionnement lors de :

* **La génération de QRCode :**



Générer QR Code

* **Le scan de QRCode :**

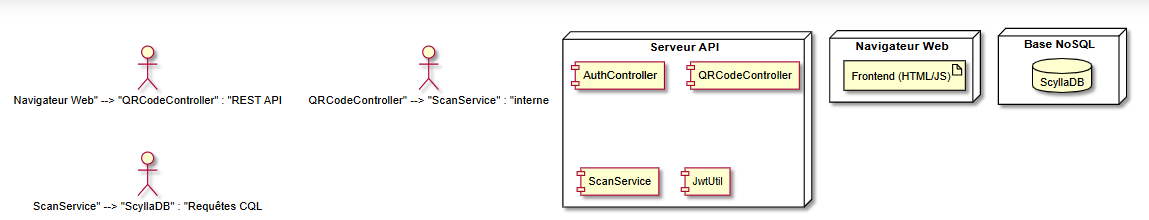


Scanner QR Code

## **II.7 Diagramme de Déploiement :**

**Composants** :

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



# **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

# **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 |

# **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 si signature invalide ou token expiré

# **DEPLOIEMENT ET ENVIRONNEMENT :**

* 📦 **Langage** : Java 17+
* ⚙️ **Build** : Maven (spring-boot-starter-\*, spring-data-cassandra)
* 🛢️ **Base** : ScyllaDB
* 🐳 **Docker** recommandé pour ScyllaDB
* 🧱 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 :**

Il était question pour nous de répondre à la menace que représente l’émergence de techniques cybercriminelles dans le cadre de « l’authentification Handshake de documents ». Au terme de notre travail nous avons conçu une API qui permettait, par le biais de codes QR, d’établir l’authenticité et l’intégrité de « documents » (qui représentent un service ou une transaction), tout en incluant un certain degré de liberté (le libre choix de la clé secrète du fournisseur) dans la génération de ceux-ci. Nous cherchons continuellement des voies d’amélioration de son principe de conception.