École Nationale Supérieure Polytechnique de Yaoundé
National Advanced School of Engineering of Yaoundé

Département de Génie Informatique
Department of Computer Engineering

PROJET D'ADMINISTRATION RÉSEAUX

Thème : *Modélisation d'une solution de non répudiation : Handshake basé sur le code QR.*

Réalisé par les étudiants :

DJONGO FOKOU Ariel Sharon	21P360
DJOKO DJODOM Syntia Loana	21P038
MBIAMY NGAMENI Steven Loic	23P770
NGHOGUE TAPTUE Franck Roddier	21P279
Antoine Emmanuel ESSOMBA ESSOMBA	23P750

Sous la supervision de :

Prof. Thomas DJOTIO
M. Juslin KUTCHE

Année académique : 2024-2025

Table des matières

Table des figures

Introduction

Dans un contexte de plein croissance des services numériques, la sécurité et la fiabilité des transactions en ligne deviennent primordiales. Dans ce contexte, la non-répudiation est un concept essentiel garantissant qu'une partie ne puisse nier avoir pris part à une transaction après qu'elle eut lieu. Ce projet propose la mise en œuvre d'un système de non-répudiation pour une application de commande de taxis, en utilisant un mécanisme de handshake basé sur la génération et la vérification de codes QR.L'objectif principal de ce projet est de garantir qu'un passager et un chauffeur puissent prouver, sans équivoque, leur engagement dans une course. Grâce à ce mécanisme, le passager est capable de confirmer l'identité du chauffeur assigné par le système en scannant un code QR unique généré lors de la commande. Ce processus assure la correspondance exacte entre le chauffeur et le passager tout en prévenant toute tentative de fraude ou de substitution, assurant ainsi la sécurité et la confiance dans le service. Ce projet met en œuvre plusieurs concepts clés de la sécurité informatique et des systèmes distribués, notamment : L'authentification et La non-répudiation : Assurer que chaque partie impliquée ne puisse contester sa participation. Dans le cadre de ce projet on résout le problème de la vérification des identités à travers le scan des QR codes. Le résultat final est un système qui garantit la transparence, la sécurité, et la confiance dans le processus de commande de taxi.

1 Conception du projet

1.1 Diagramme de contexte

Ce système gère les interactions entre les passagers, les chauffeurs et les services liés aux QR codes.

Acteurs :

- **Passager** : L'acteur qui initie la commande de taxi, scanne le QR code, et valide la prise en charge. Cet acteur interagit avec l'application pour notamment :
  - Faire une demande de taxi
  - Recevoir le QR code du chauffeur
  - Scanner le QR code à l'arrivée du chauffeur
- **Chauffeur** : Cet acteur reçoit la demande de course, se voit attribuer un QR code par l'application, et attend la confirmation du passager lors de la prise en charge. Les interactions incluent :
  - Réception des informations de la course
  - Affichage du QR code à scanner par le passager
  - Réception de la confirmation de la course après la validation

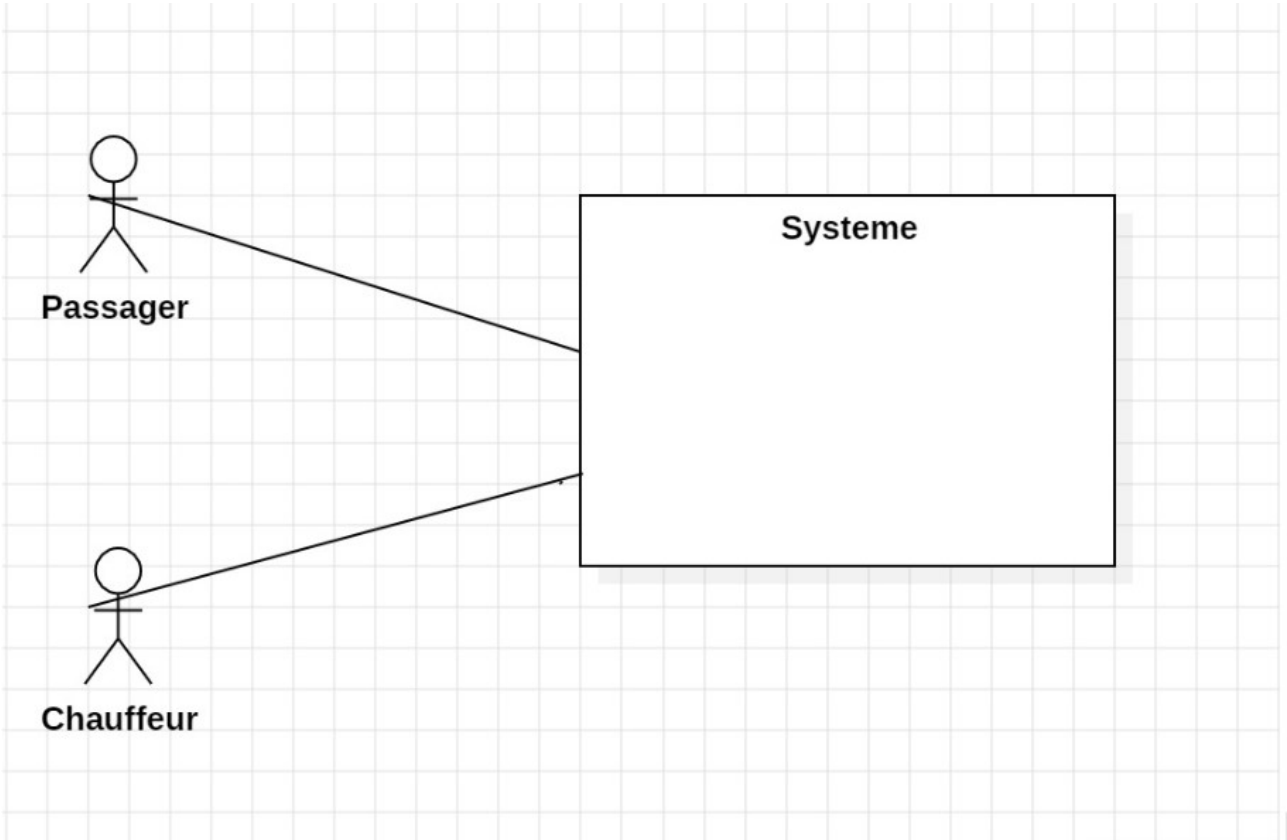

FIGURE 1 – Diagramme de contexte

1.2 Diagramme de package

Ce diagramme de package représente l'architecture simplifiée du système d'authentification et de commande pour une application mobile dédiée aux chauffeurs et clients.

1.2.1 Système d'Identification

— **Responsabilité** : Ce sous-système est responsable de toutes les opérations liées à l'authentification des utilisateurs.
— **Méthodes incluses :**
  — Authentifier les utilisateurs et gérer leur connexion : authentifierUtilisateur()
  — Vérifier les informations d'identité des utilisateurs : Dans le cadre du processus KYC (Know Your Customer)
  — Vérifier les informations d'identité des utilisateurs : Dans le cadre du processus KYC (Know Your Customer)

1.2.2 Gestion de Commande

— **Responsabilité** : Ce sous-système gère le processus de commande entre clients et chauffeurs.
— **Méthodes incluses :**
  — Passer des commandes pour les clients
  — Valider les commandes pour les chauffeurs
  — Gérer l'état des commandes, incluant le suivi et les modifications possibles

Les deux sous-systèmes interagissent étroitement pour offrir une expérience utilisateur fluide, en assurant à la fois une identification sécurisée et un traitement efficace des commandes.

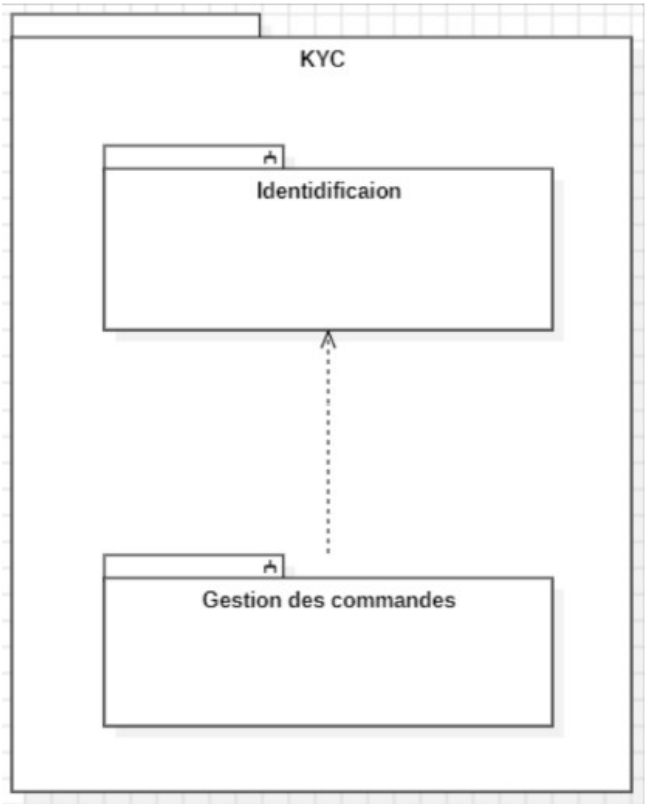

FIGURE 2 – Diagramme de package

1.3 Diagramme de Classe

Le diagramme de classes représente la structure des entités principales d'un système de gestion de commandes de taxi. Il comprend quatre classes clés : **Personne**, **Client**, **Chauffeur**, et **Commande**.

1.3.1 Classe : Personne

La classe Personne est une classe générale qui définit les attributs et comportements communs à toute personne impliquée dans le système. Cette classe peut être spécialisée en Client ou Chauffeur selon le rôle de la personne dans le processus.

- **Attributs :** Cette classe possède des attributs généraux, tels que le nom, prenom, et d'autres informations d'identification.
- **Méthodes :**Les méthodes couvrent les actions que le client peut entreprendre, comme passer une commande ou consulter ses commandes passées.

1.3.2 Classe : Client

La classe Client est une spécialisation de la classe Personne, représentant un utilisateur du système qui passe des commandes de taxis. Elle contient des informations spécifiques sur le client et les actions qu'il peut effectuer.

- **Attributs :** Les attributs incluent des informations relatives au client, telles que son id.
- **Méthodes :** Les méthodes couvrent les actions que le client peut entreprendre, comme passer une commande ou consulter ses commandes passées.

1.3.3 Classe : Chauffeur

La classe Chauffeur est également une spécialisation de la classe Personne, mais elle représente le rôle du conducteur de taxi dans le système. Elle contient des informations propres à un chauffeur et ses interactions avec les commandes.

- **Attributs :** Les attributs incluent les informations spécifiques du chauffeur, comme son véhicule et ses qualifications.
- **Méthodes :**Les méthodes permettent de gérer le processus de commande, du début à la fin, y compris la validation et la notification des parties concernées.

1.3.4 Classe : Commande

La classe Commande représente une transaction dans le système, où un Client passe une commande et un Chauffeur est assigné pour l'exécuter. Elle sert de lien entre les classes Client et Chauffeur.

- **Attributs :** Les attributs incluent les détails de la commande, tels que la date, le lieu de départ, et le chauffeur assigné.
- **Méthodes :**Les méthodes permettent de gérer les actions liées à l'activité du chauffeur dans le système, comme l'acceptation d'une commande.

Relations :

- **Héritage :** La classe Personne est la classe parent des classes Client et Chauffeur. Cette relation d'héritage permet aux clients et aux chauffeurs de partager des caractéristiques communes tout en ayant des spécificités propres.
- **Association :**La classe Commande est liée à la classe Client et à la classe Chauffeur. Chaque commande est associée à un client qui la passe et à un chauffeur qui la réalise.

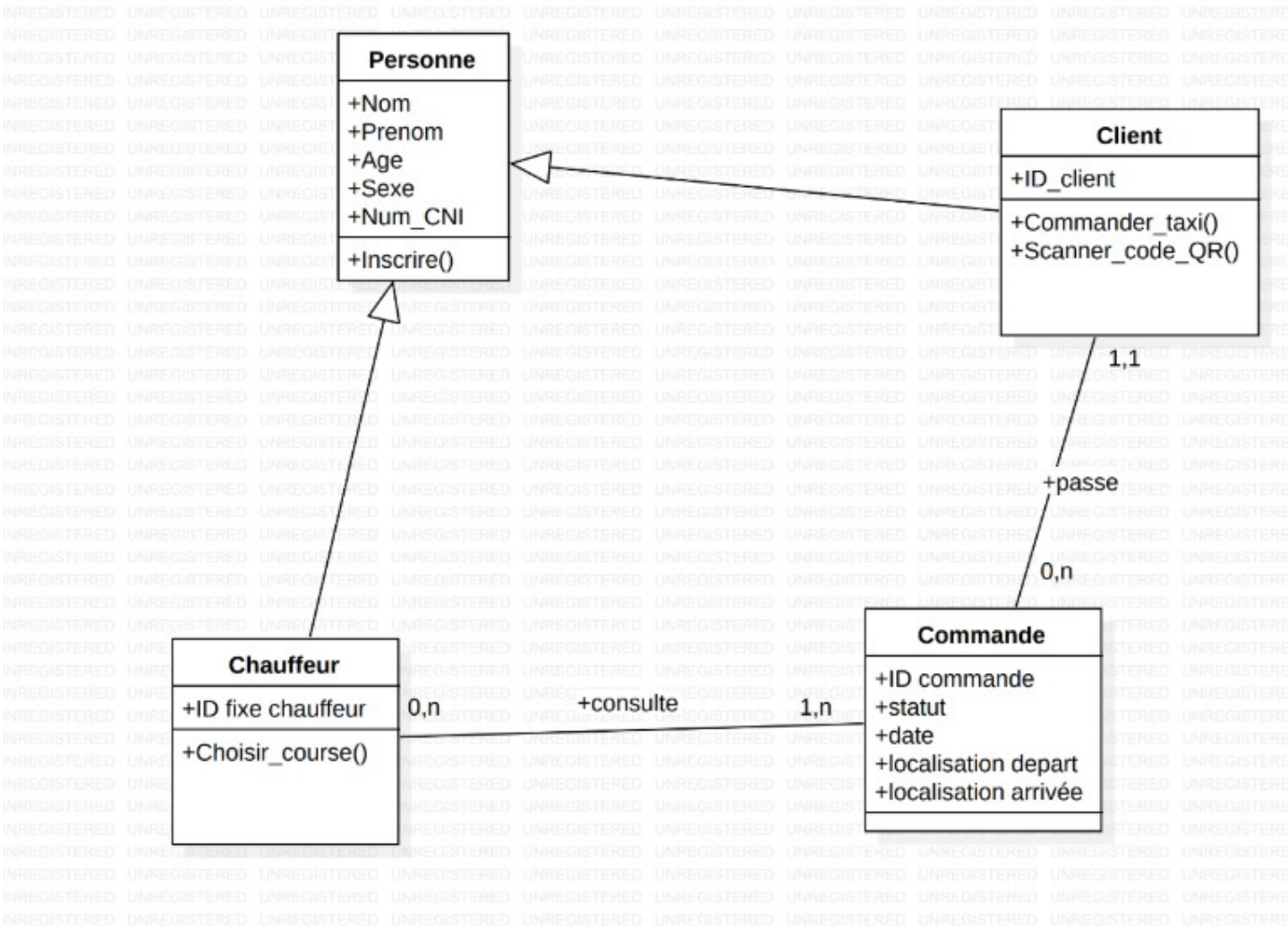

FIGURE 3 – Diagramme de classe

1.4 Diagramme de Cas d'Utilisation

Générer un code QR

- **Acteur principal :**Client
- **Description :**Un client connecté peut générer un code QR. Lorsqu'il génère ce code, un ID aléatoire de 64 chiffres est créé et associé à son IDChauffeur, puis stocké dans une table temporaire appelée QRCode. Le code QR doit être scanné par le client dans un délai de 20 secondes ou il est supprimé. Le chauffeur peut également fermer le code QR manuellement avant la fin des 20 secondes.
- **Préconditions :**Le chauffeur doit être connecté.
- **Postconditions :**Le code QR est généré et stocké temporairement. S'il est scanné dans les 20 secondes, la vérification est réussie. Sinon, il est supprimé.

Scanner le code QR

- **Acteur principal :**Fournisseur
- **Description :**Un fournisseur connecté peut scanner le code QR d'un chauffeur. Le système récupère l'ID aléatoire et l'IDChauffeur, puis vérifie leur présence dans la table temporaire QRCode. Si les informations sont présentes, le client reçoit une confirmation de succès. Sinon, un échec est renvoyé.

— **Préconditions :**Le client doit être connecté et le code QR du chauffeur doit être actif.
— **Postconditions :**Le scan est validé si le code est présent dans la table QRCode.

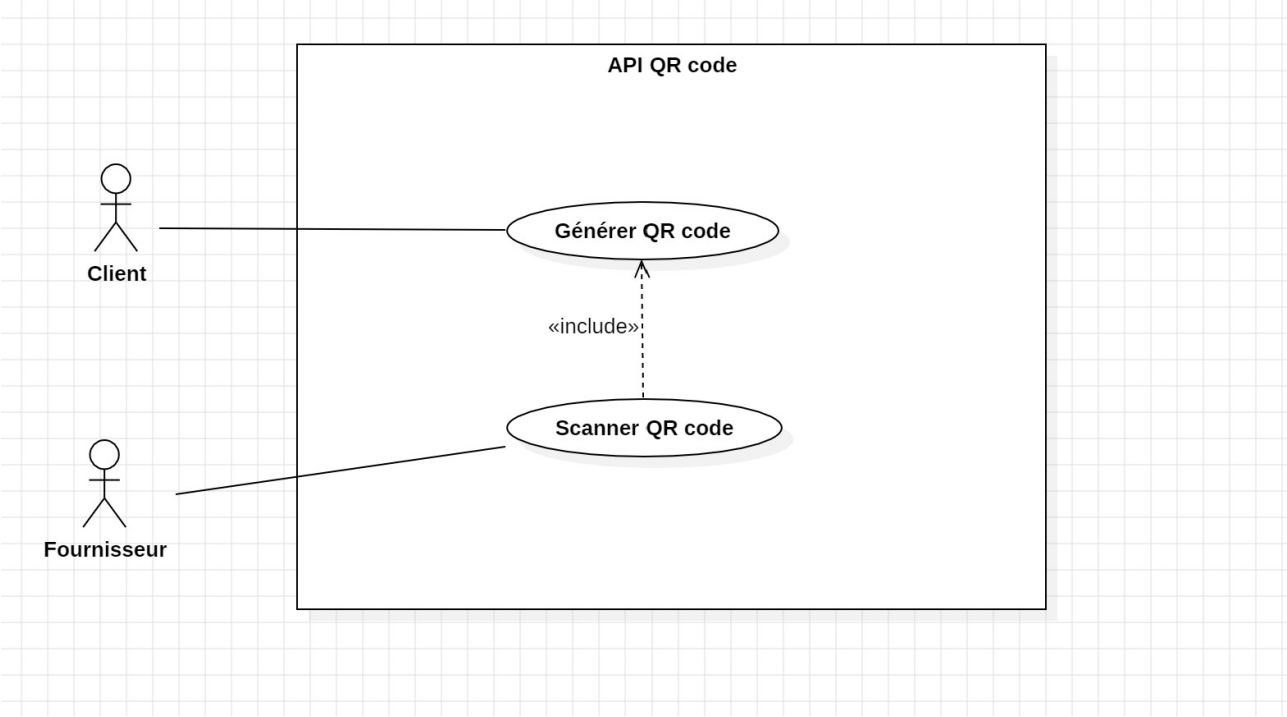

FIGURE 4 – Diagramme de cas d'utilisation

1.5 Diagramme d'activité

Choisir une course .
Étape 1 : Connexion du chauffeur
— **Chauffeur se connecte :** Le processus commence par la tentative de connexion du chauffeur à la plateforme. Le chauffeur doit entrer ses identifiants pour accéder à l'application

.
Étape 2 : Vérification de la connexion
— **Connexion réussie ?** : L'application vérifie si les identifiants sont corrects. Si la connexion est réussie, l'activité continue, sinon l'application affiche un message d'erreur et le processus se termine.

.
Étape 3 : Consultation des courses disponibles
— **Accéder à la liste des courses disponibles :** Une fois connecté, le chauffeur peut accéder à la liste des courses commandées par les clients. Ces courses incluent des informations comme le point de départ, la destination, l'heure de départ, et le tarif proposé par le client.

.
Étape 4 : Choix de la course
— **Consulter les courses** : Le chauffeur parcourt la liste des courses disponibles.
— **Choisir une course :** Le chauffeur sélectionne une course à accepter. Cela indique qu'il est prêt à prendre en charge cette course spécifique.

.
Étape 5 : Génération du QR Code

— **Course choisie ?** : Si le chauffeur a sélectionné une course, le système l'assigne à ce chauffeur. L'activité continue vers la génération du QR Code..
— **Générer le QR Code :** Le système génère un QR Code unique pour cette transaction, qui contient un identifiant aléatoire de 64 chiffres et l'ID du chauffeur.

.
Étape 6 : Affichage et validation du QR Code
— **Afficher le QR Code pour 20 secondes :** Le QR Code est visible pour une période limitée (20 secondes), durant laquelle le client doit le scanner.

.
Étape 7 : Scan et validation du QR Code
— **Client scanne QR Code ? :** Si le client scanne le QR Code dans le délai imparti, le système vérifie si les informations sont correctes (le couple ID aléatoire + IDChauffeur).
— Si le QR Code est validé, la course est confirmée.
— Si le QR Code n'est pas scanné à temps ou si le chauffeur clique sur "fermer", le QR Code est fermé, et le couple d'identifiants est supprimé de la base de données.

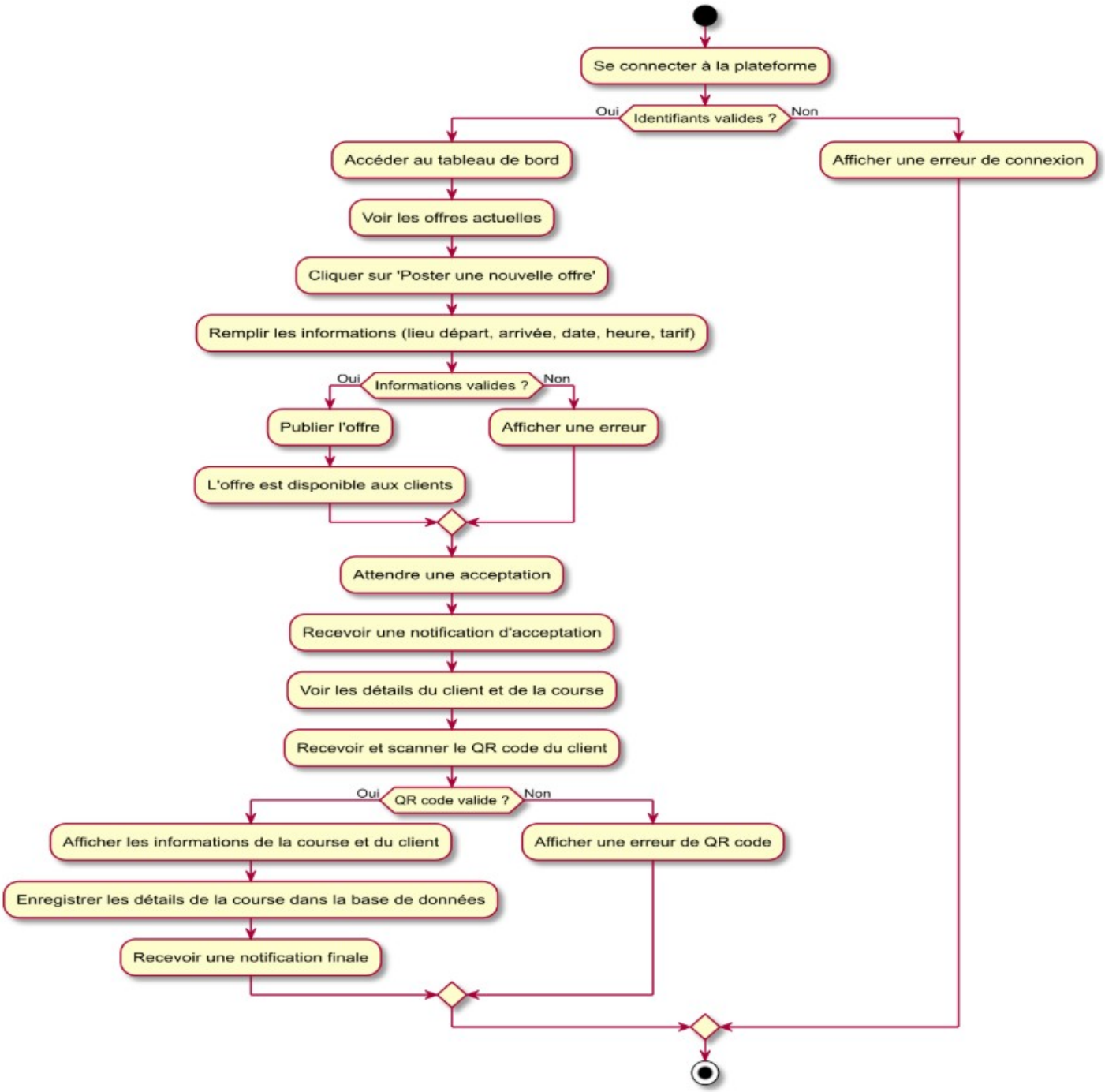

FIGURE 5 – Diagramme d'activité Choisir une course

1.6 Diagramme de séquence

Cas d'utilisation "Scanner code QR"

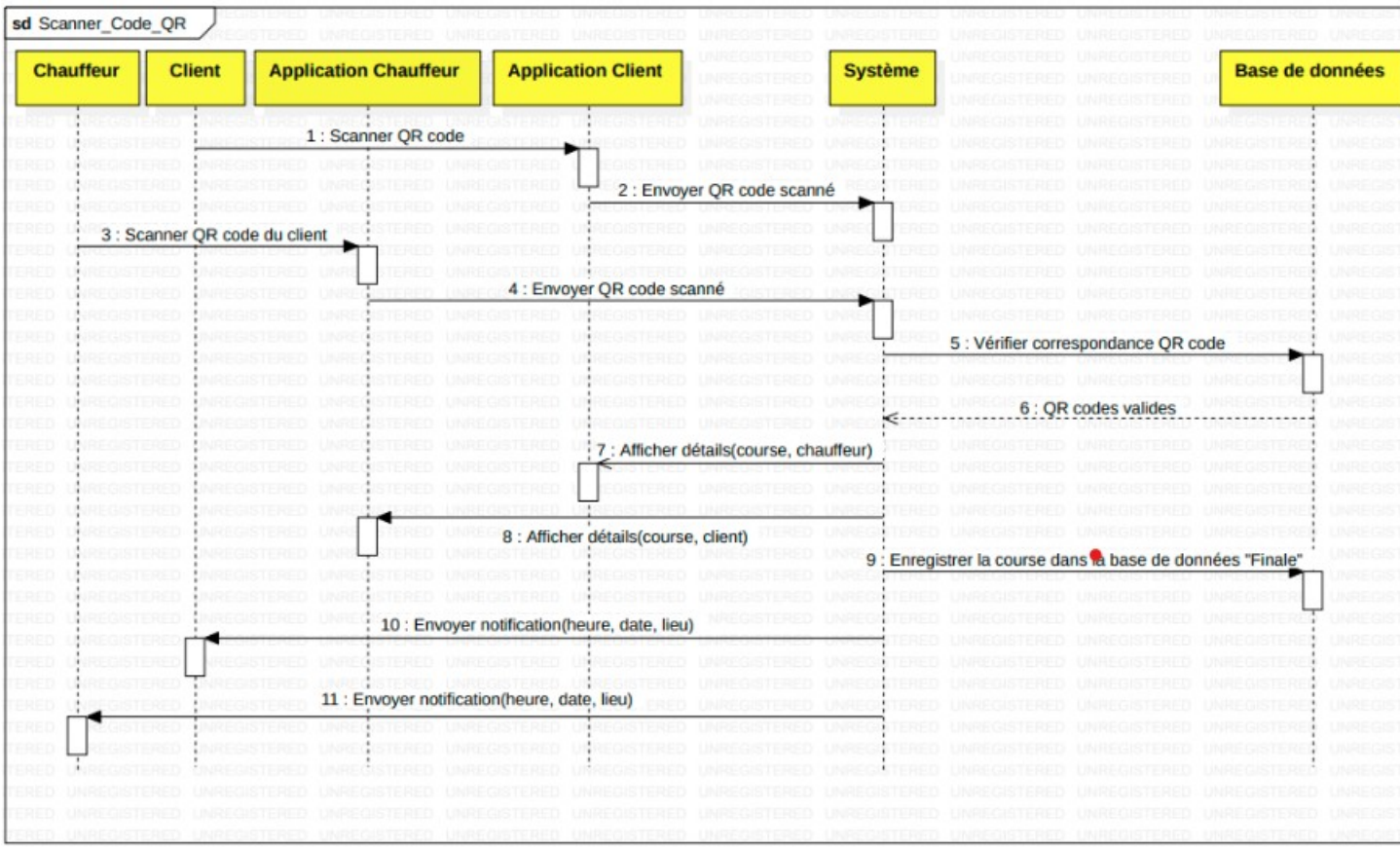

FIGURE 6 – Diagramme de séquence Scanner code QR

1.7 Diagramme d'etat transition

Chauffeur Initialement, après s'être connecté, le chauffeur est dans l'attente des courses, puis il reçoit différentes courses et en choisi une ; ensuite lorsqu'il rencontre le client, il génère un code QR qui sera scanné par le client et la course est ainsi validée.

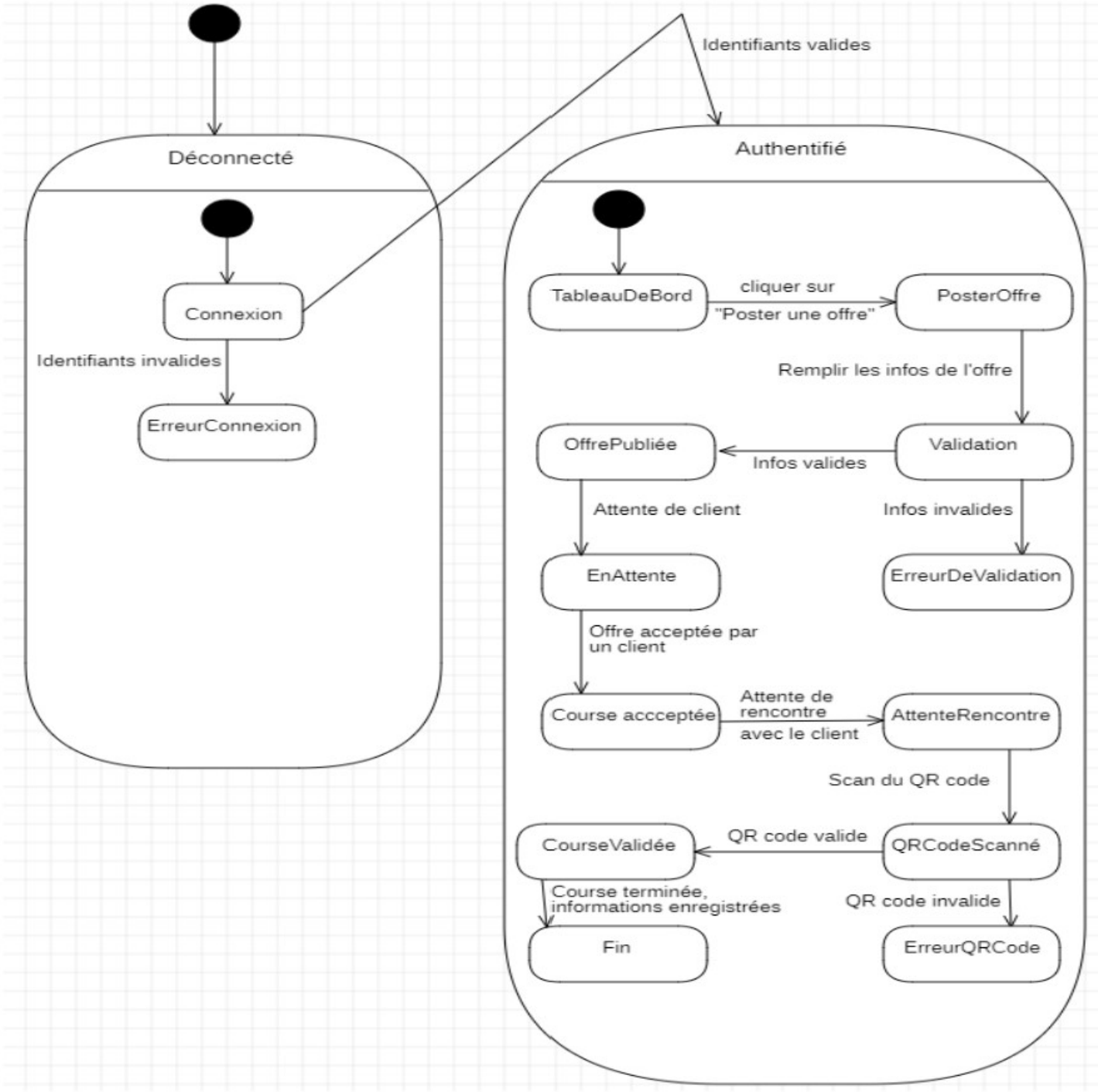

FIGURE 7 – Diagramme d'état transition du chauffeur

Client Après s'être connecté, le client entre ses paramètres puis commande une course, la course est ainsi ajoutée à la liste des courses du chauffeur.

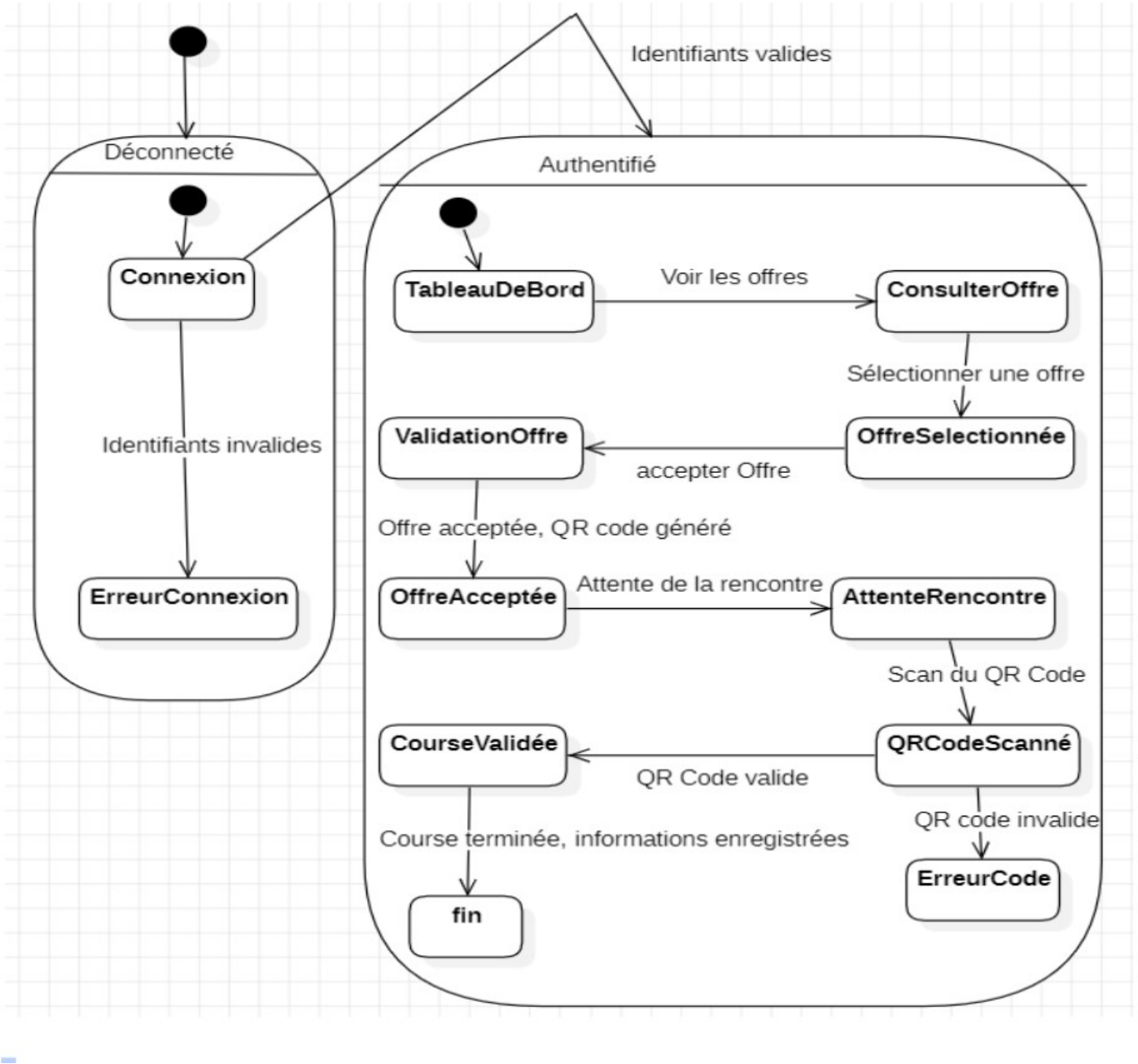

FIGURE 8 – Diagramme d'état transition du client

1.8 Diagramme de déploiement

Composants déployés :

- **Application Client (sur l'appareil du client) :** Gère les interactions du client, telles que la commande de course et le scan des QR Codes.
- **Application Chauffeur (sur l'appareil du chauffeur) :** Gère les interactions du chauffeur, comme la consultation des courses et la génération des QR Codes.
- **API Backend (sur le serveur web) :** Fournit toutes les fonctionnalités métier du système, telles que la gestion des commandes, la génération des QR Codes, et la validation des courses.
- **Base de données (sur le serveur de base de données)** : Stocke et gère toutes les informations liées aux utilisateurs (clients et chauffeurs), aux commandes, et aux QR

Codes.
— **Générateur de QR Code (sur un service externe ou un module dédié) :** Produit le QR Code unique à partir des identifiants fournis par le système.

Communications entre les nœuds :
— **Application Client - Serveur Web :** L'application du client envoie des requêtes au serveur pour la création et la validation de courses, ainsi que pour le scan des QR Codes.
— **Application Chauffeur - Serveur Web :** L'application du chauffeur envoie des requêtes au serveur pour consulter les courses et demander la génération d'un QR Code.
— **Serveur Web - Base de données :** Le serveur web interagit avec la base de données pour gérer les informations des utilisateurs, des courses, et des QR Codes.
— **Serveur Web - Générateur de QR Code :** Le serveur web demande au générateur de QR Code de produire un code unique pour chaque course assignée à un chauffeur.

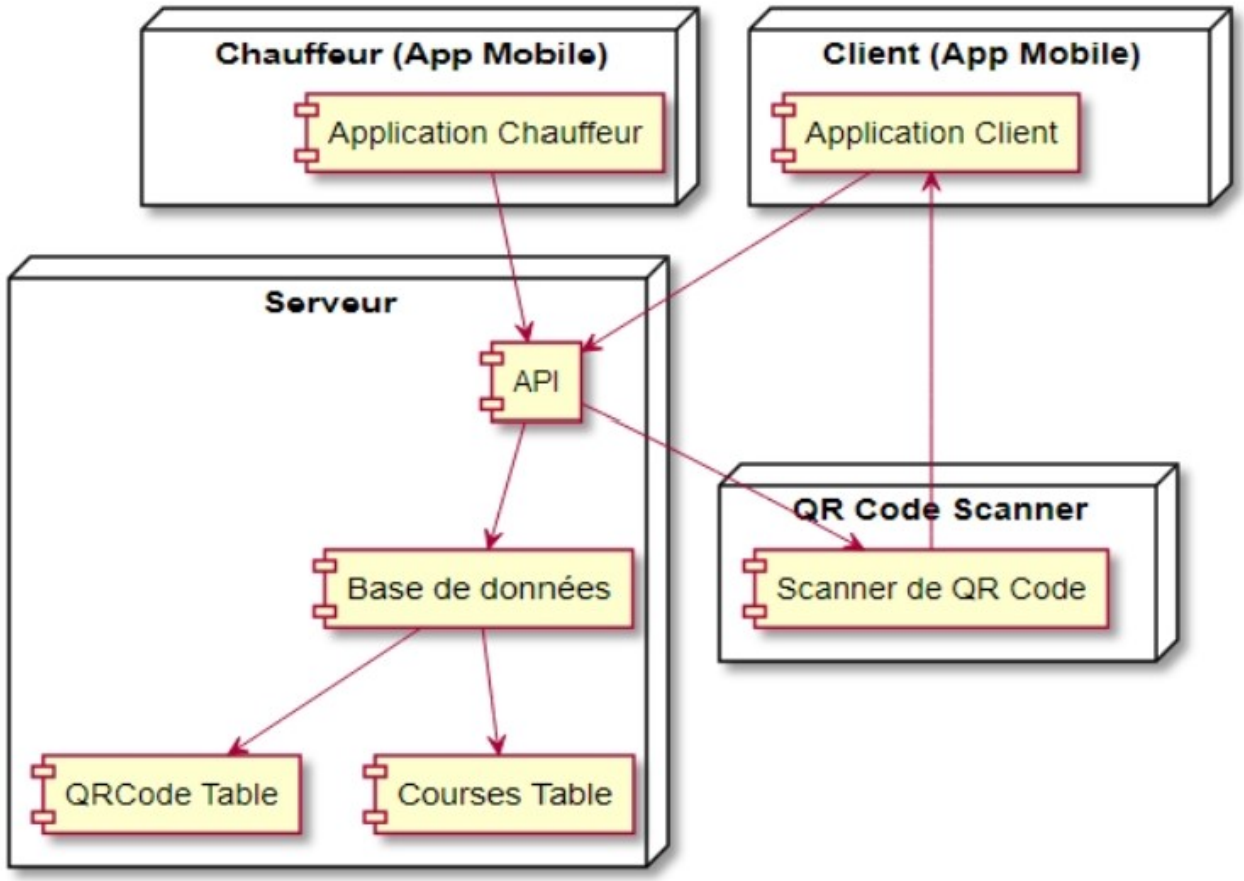

FIGURE 9 – Diagramme d'état de déploiment

2 Implémentation du projet

2.1 Fonctionnalités implémentées

Dans le cadre de la réalisation de notre projet, nous avons mis en place plusieurs fonctionnalités essentielles pour assurer une expérience fluide et sécurisée aux utilisateurs. Ces fonctionnalités s'articulent principalement autour de la gestion des rôles des utilisateurs et d'un mécanisme de non-répudiation basé sur les codes QR.

1. **L'introduction de la notion de rôle lors de l'authentification.**
   L'une des premières fonctionnalités que nous avons intégrées est la gestion des rôles des utilisateurs dès l'authentification. Cette distinction des rôles est essentielle pour déterminer les droits et les actions disponibles pour chaque utilisateur au sein de l'application. Nous avons défini deux types de rôles :
   — **Le Chauffeur** : Il s'agit de l'utilisateur ayant la capacité de publier des courses sur la plateforme. Il peut entrer des informations relatives à son trajet, comme le point de départ, la destination, l'horaire prévu, ainsi que le tarif de la course. Une fois la course publiée, elle devient visible aux clients potentiels.
   — **Le Client :** Ce rôle est attribué aux utilisateurs cherchant à réserver une course. Un client peut consulter la liste des courses disponibles, filtrer les résultats en fonction de ses préférences et sélectionner une course qui lui convient. Une fois la réservation effectuée, le chauffeur concerné est notifié et peut accepter ou refuser la demande.
   Cette gestion des rôles permet d'offrir une interface plus intuitive et des fonctionnalités adaptées aux besoins spécifiques de chaque type d'utilisateur.
2. **Génération et scan de codes QR pour une solution de non-répudiation.**
   Une problématique majeure dans le domaine des applications de transport est la vérification de l'identité des parties impliquées dans une transaction. En d'autres termes, lorsqu'un client et un chauffeur se rencontrent, il est essentiel de s'assurer qu'ils sont bien ceux qui ont validé la course via l'application. Deux questions essentielles se posent alors :
   — **Pour le client :** "Est-ce bien le chauffeur que j'ai choisi lors de la réservation ?"
   — **Pour le chauffeur :** "Est-ce bien le client qui a réservé ma course ?"
   Pour répondre efficacement à ces interrogations et éviter tout litige, nous avons mis en place un mécanisme de non-répudiation basé sur un système de codes QR :
   - Lorsqu'un client sélectionne une course et que le chauffeur accepte, un identifiant unique est généré pour cette transaction.
   - Lors de la rencontre physique entre le chauffeur et le client, le chauffeur génère un code QR contenant cet identifiant unique et ses informations.
   - Le client scanne ce code QR avec son application mobile.
   - Si les informations correspondent à la réservation effectuée, le système valide automatiquement la transaction et enregistre cette confirmation.

   Grâce à ce procédé, nous nous assurons que seule la personne ayant réservé la course peut y accéder, et qu'aucun tiers ne peut usurper l'identité du chauffeur ou du client. Ce mécanisme apporte une solution efficace à la problématique de la non-répudiation et améliore la confiance des utilisateurs dans l'application.

2.2 Environnement de développement

Lors de la conception et du développement de notre projet, nous avons sélectionné des technologies modernes et robustes afin d'assurer une architecture performante, évolutive et sécurisée. Notre choix s'est porté sur les solutions suivantes :

1. **Next.js (Frontend)**
   Next.js est un framework basé sur React qui apporte des fonctionnalités avancées telles que le rendu côté serveur (SSR), la génération statique (SSG) et une gestion optimisée du routage. Nous l'avons choisi pour plusieurs raisons :
   — **Performance accrue** : grâce au rendu côté serveur et à la génération statique, les pages se chargent plus rapidement.
   — **Expérience développeur améliorée** : avec une architecture modulaire et des outils intégrés facilitant le développement.
   — **SEO optimisé** : les pages sont mieux indexées par les moteurs de recherche par rapport aux applications React classiques.
   — **Facilité d'intégration avec des API** : permettant une communication fluide avec notre backend en Spring Boot.
2. **Spring Boot (Backend)**
   Spring Boot est un framework Java puissant permettant de développer des applications backend robustes et scalables. Nous avons opté pour cette technologie en raison de :
   — **Simplicité et rapidité de développement** : grâce à sa configuration minimale et ses nombreux modules intégrés.
   — **Sécurité avancée** : intégration facile avec Spring Security pour la gestion de l'authentification et de l'autorisation.
   — **Compatibilité avec les bases de données relationnelles** : facilitant l'interaction avec SyllABD.
   — **Architecture microservices** : permettant une scalabilité et une maintenabilité accrues.
3. **SyllABD (Base de Données)**
   SyllABD est un système de gestion de base de données relationnelle conçu pour offrir une gestion performante et une grande flexibilité. Nous l'avons choisi pour :
   — **Son adaptabilité à différents contextes** : particulièrement adaptée aux projets nécessitant une personnalisation avancée.
   — **Sa performance en traitement de requêtes** : optimisée pour gérer efficacement les interactions avec des ensembles de données importants.
   — **Sa compatibilité avec Spring Boot** : facilitant l'intégration pour la gestion des transactions complexes.
   — **Sa modularité et extensibilité** : permettant une configuration ajustée aux besoins spécifiques du projet.

L'association de **Next.js, Spring Boot et SyllABD** nous permet d'avoir une application web performante, évolutive et sécurisée. Ce choix technologique nous offre un excellent compromis entre rapidité de développement, flexibilité et performance, tout en assurant une expérience utilisateur optimale.

2.3 Détails de l'implémentation

2.3.1 Serveur d'application

L'architecture que nous avons employée est l'architecture standard de Spring Boot :

— **Les models :** Ils représentent les données et les règles de gestion associées dans une application. Ils définissent la structure des données, les relations entre les entités et les contraintes à respecter. En général, les modèles sont directement liés à la base de données et peuvent inclure des méthodes pour manipuler ces données.

— **Les repositories :** Ils agissent comme une couche intermédiaire entre les modèles et les services. Leur rôle principal est d'encapsuler la logique d'accès aux données, en fournissant des méthodes pour récupé rer, ajouter, modifier et supprimer des données dans la base de données. Ils permettent d'abstraire les opérations de persistance, ce qui facilite le test et la maintenance du code.

— **Les services** : Ils contiennent la logique métier de l'application. Ils orchestrent les opérations sur les modèles via les repositories, appliquent les règles de gestion, et fournissent des fonctionnalités complexes en combinant plusieurs opérations de données. Cette couche permet de séparer la logique métier des contrôleurs, ce qui améliore la réutilisabilité et la testabilité du code.

— **Les controllers :** Les contrôleurs sont responsables de gérer les requêtes entrantes, de traiter les données reçues, d'appeler les services appropriés, et de retourner les réponses aux clients (généralement via des vues ou des API). Ils servent de point d'entrée dans l'application et coordonnent les interactions entre l'utilisateur et le système.

— Les packages accesoires comme dto,config, util...

2.3.2 Déploiement

Voir dans l'annexe.

3 Contexte : Non Répudiation pour la Collecte via Taxis ou Covoiturage

La non-répudiation est un principe de sécurité permettant de garantir qu'une partie ne peut pas nier avoir effectué une action ou participé à une transaction. Dans le cadre de la collecte par taxis ou de covoiturage, il est crucial d'assurer que ni le conducteur ni le client ne puissent nier avoir validé leur rencontre et la transaction de service associée.

3.1 Objectif

L'objectif ici est de modéliser une solution de non-répudiation utilisant un handshake basé sur un code QR, c'est -à -dire, modéliser une solution qui assure que ni le conducteur ni le client ne puisse nier la rencontre. Ce système de validation peut intervenir lors de la rencontre entre le conducteur et le client qui a réservé et confirmé un trajet. Cela inclut l'authentification et la validation mutuelle de la présence des deux parties au moment du début du trajet.

3.2 Scénario

1. Client : Il réserve une course ou un trajet via une application ou une plateforme de collecte par taxi ou de covoiturage, reçoit la confirmation, et obtient un code QR spécifique à son trajet.
2. Conducteur : Le conducteur confirme la réservation et la course est planifiée. Il publie le trajet, reçoit une notification de réservation et la confirmation du client, ainsi qu'un code QR correspondant à cette transaction.
3. Rencontre sur le lieu de départ : Lorsque le conducteur et le client se rencontrent physiquement au point de départ, une méthode de non-répudiation doit être utilisée pour confirmer que cette rencontre a effectivement eu lieu. Lors de cette rencontre, un échange sécurisé basé sur un code QR est utilisé pour assurer la non-répudiation. Cela permet de garantir que les deux parties sont bien celles prévues par la plateforme.

3.3 Modélisation de la Solution

1. **Préparation de la rencontre :**
   — Le client et le conducteur ont chacun une application mobile dédiée (ou intégrée à un service de transport).
   — Lorsque le client confirme la réservation, un code QR spécifique à cette réservation est généré. Ce code contient un identifiant unique de la course, une signature numérique (pour authentification et intégrité), ainsi que d'autres informations comme l'heure de départ prévue et l'identifiant du véhicule.
2. **Génération et Transmission des QR Codes**
   — Client : Après avoir réservé, l'application du client génère un code QR qui contient des informations liées à la réservation :
      — Identifiant unique du trajet
      — Date et heure du trajet
      — Identité cryptée du client é publique du client
   — Conducteur : Il reçoit également un code QR avec des informations similaires :
      — Identifiant unique du trajet
      — Date et heure du trajet
      — Identité cryptée du conducteur

— Clé publique du conducteur

Les QR codes sont créés et partagés uniquement après que les deux parties aient confirmé la réservation, assurant ainsi qu'ils sont synchronisés.

3. **Validation mutuelle au Moment de la Rencontre** Lors de la rencontre, le client et le conducteur échangent leurs QR codes à travers l'application mobile.
   — **Phase d'échange :**
      — Le conducteur scanne le QR code du client, et l'application valide automatiquement si l'identifiant unique du trajet et les informations de la réservation correspondent.
      — Le client scanne également le QR code du conducteur pour vérifier que c'est bien le conducteur prévu par la plateforme.
      — Lors du scan, l'application vérifie les informations contenues dans le code QR (signature numérique, informations sur la course, etc.).
      — En parallèle, un code QR de validation généré par l'application du conducteur est montré au client pour un échange bilatéral. Le client scanne également ce code pour confirmer la présence du conducteur.
      — Les deux applications échangent et vérifient les informations via une
   — **Authentification et confirmation :**
      — Après l'échange des informations via les codes QR, un serveur central vérifie les signatures numériques des deux parties (conducteur et client).
      — Si tout est validé (les informations de la course correspondent et les signatures sont authentiques), la course est marquée comme commencée dans le système.
      — Une preuve cryptographique (timestamp, signature des deux parties) est stockée côté serveur, assurant la non-répudiation.
   — **Non-répudiation via signatures numériques :**
      — Une fois les QR codes validés, les deux parties signent numériquement la transaction via leurs clés privées respectives. Cette signature numérique est ajoutée à la blockchain (si une technologie de registre distribué est utilisée), assurant que la transaction est immuable.
      — Les informations validées (heure, date, emplacement GPS au moment de la validation) sont également enregistrées sur un serveur sécurisé.
   — **Données supplémentaires :**
      — Un timestamp (marque de temps) est ajouté lors de la validation mutuelle.
      — La géolocalisation pourrait également être utilisée pour confirmer que la rencontre a eu lieu au point de rendez-vous prévu.
4. **Enregistrement et Audits :**Les données collectées sont enregistrées pour une potentielle vérification ultérieure (audit). En cas de litige (par exemple, si l'une des parties nie avoir réalisé le trajet), la signature numérique, la correspondance des QR codes, ainsi que les données enregistrées (comme l'emplacement GPS au moment de la rencontre) servent de preuve irréfutable.
5. **Résolution des litige**
   En cas de litige post-trajet, l'historique des transactions et les signatures numériques peuvent être utilisées pour prouver que :
   — Le client et le conducteur se sont bien rencontrés à l'heure et au lieu prévus.
   — Le conducteur a bien validé la présence du client et vice versa.

Étapes clés de la modélisation technique :

1. **Génération et validation des codes QR :**
   — Chaque QR code doit inclure une signature numérique pour garantir que les informations sont inviolables.

— Utilisation de certificats publics/privés pour signer les informations côté serveur (les clients et conducteurs pourraient avoir des clés privées associées à leur compte).

2. **Échange sécurisé :**L'échange des QR codes pourrait être complété par une transmission des données chiffrées via TLS/SSL pour garantir la confidentialité des informations échangées.
3. **Stockage des preuves :**Stocker les informations de la course validée (y compris les signatures et les timestamps) sur un serveur sécurisé ou via une blockchain pour assurer l'intégrité et l'immutabilité des données.
4. **Conformité légale :**Il est important de respecter les réglementations en vigueur concernant la protection des données personnelles (comme le RGPD dans l'Union Européenne), car des informations sensibles comme la géolocalisation ou les identifiants des utilisateurs sont impliquées.

Schéma de communication avec serveur central Cette approche qui intègre un serveur central pour valider les données et stocker les preuves nécessaires garantit la non-répudiation. son rôle consiste :

— **à la validation des signatures numériques** : Le serveur s'assure que les signatures échangées entre les deux parties sont valides, garantissant que ni l'une ni l'autre ne peut renier sa participation.
— **au stockage des preuves :** Toutes les preuves de la rencontre (signatures, timestamp, géolocalisation si nécessaire) sont stockées de manière sécurisée pour garantir la non-répudiation.
— **à la synchronisation** : Le serveur centralise les informations pour que les deux applications (conducteur et client) soient synchronisées et puissent afficher la confirmation de la rencontre en temps réel.

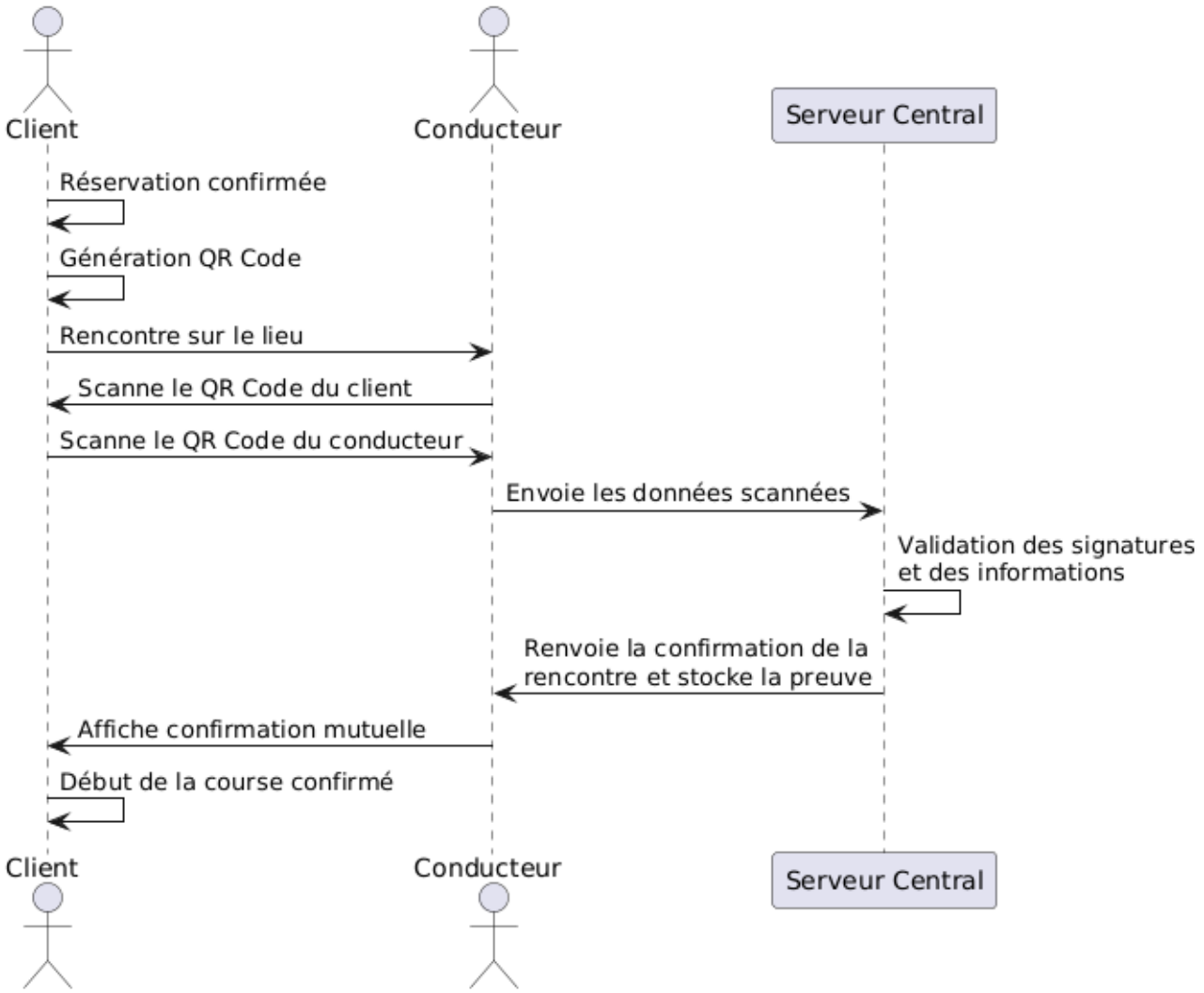

FIGURE 10 – Schéma de communication avec serveur central

Explication du flux de communication

1. Réservation confirmée : Le client fait une réservation via l'application, et une confirmation est envoyée au conducteur.
2. Génération de QR Code : L'application du client génère un QR Code spécifique à la réservation, incluant des informations comme l'identifiant de la course, la signature numérique, etc.
3. Rencontre sur le lieu : Le client et le conducteur se retrouvent physiquement au point de départ prévu.
4. Scan du QR Code du client : Le conducteur scanne le QR Code du client avec son application, obtenant les informations nécessaires (identifiant de la course, signature numérique).
5. Scan du QR Code du conducteur : Le client scanne également un QR Code généré par le conducteur. Cette étape permet de valider mutuellement la présence des deux parties.
6. Envoi des données scannées au serveur : Les deux applications (celle du conducteur et celle du client) envoient les données scannées au serveur central, y compris les informations cryptées et les signatures numériques.

7. Validation par le serveur central : Le serveur central vérifie que les signatures numériques sont valides et que les informations échangées correspondent à la réservation initiale.
8. Confirmation renvoyée par le serveur : Une fois les validations effectuées, le serveur envoie une confirmation à l'application du client et à celle du conducteur, confirmant que la rencontre a eu lieu. Il stocke également une preuve de cette interaction (timestamp, signatures) pour garantir la non-répudiation.
9. Confirmation sur les applications : Les deux parties voient une confirmation de la rencontre dans leurs applications respectives.
10. Début de la course confirmé : La course peut officiellement commencer et est enregistrée dans le système.

3.4 Avantages du système basé sur QR Code

1. Non-répudiation : Ni le conducteur ni le client ne peuvent nier avoir validé la rencontre grâce aux signatures numériques.
2. Simplicité d'utilisation : Les codes QR permettent un échange rapide et facile d'informations via les applications mobiles. Ils sont simples à utiliser et facilement scannables avec les smartphones.
3. Sécurité : L'utilisation de la signature numérique assure que les transactions sont immuables et vérifiables.
4. Traçabilité : Toutes les transactions sont enregistrées de manière à être auditées, garantissant que ni le client ni le conducteur ne peuvent nier leur participation.
5. Non-répudiation cryptographique : L'intégration des signatures numériques et des registres distribués (blockchain ou autre) assure une preuve solide des interactions.

3.5 Technologies potentielles

— API de génération et de validation de QR codes : Utilisation de bibliothèques comme zxing (Java) ou QRCode.js pour générer et scanner les QR codes.
— Cryptographie asymétrique : Utilisation d'algorithmes tels que RSA ou ECC pour générer les clés privées/publiques et signer numériquement les transactions.
— Blockchain : Si la transparence et l'immuabilité sont primordiales, des technologies comme Ethereum ou Hyperledger Fabric peuvent être utilisées pour stocker les transactions.

Cette solution assure une forte sécurité pour la collecte des taxis et covoiturage, tout en préservant la transparence et en renforçant la confiance entre les utilisateurs.

Conclusion

Dans un monde où les transactions numériques sont en constante augmentation, assurer la sécurité et la fiabilité des échanges est un enjeu majeur. Ce projet apporte une solution efficace à la problématique de la non-répudiation dans le cadre d'une application de commande de taxis, en intégrant un mécanisme de handshake basé sur la génération et la vérification de codes QR. Grâce à ce système, le passager et le chauffeur peuvent prouver leur engagement mutuel dans une course, garantissant ainsi l'authenticité et l'intégrité du processus. Cette approche permet non seulement de prévenir les tentatives de fraude ou de substitution, mais aussi de renforcer la confiance et la transparence dans le service. En intégrant des principes fondamentaux de sécurité informatique, notamment l'authentification et la non-répudiation, ce projet constitue une avancée significative vers des services de transport plus sécurisés et fiables. L'implémentation de cette solution ouvre également la voie à d'autres applications dans divers domaines nécessitant des garanties d'intégrité et de traçabilité des transactions numériques.

Annexes

A.1. Planning

Dans le soucis de ne pas perdre de temps et de constamment avoir une vue claire sur le projet, nous avons dressé un planning de travail s'étalant sur 16 semaines. Chacune de ces semaines s'est vue attribuée un onjectif bien précis ; de telle sorte que nous soyons constamment en évolution dans la réalisation.

Semaine	**Objectifs**
Semaine 1	Analyse du projet.
Semaine 2	Modélisation UML.
Semaine 3	Apprentissage de la technologie Next.js pour le frontend.
Semaine 4	Réalisation et merge du frontend, suivi de sa présentation.
Semaine 5-6	Récupération de l'application Let's Go sur laquelle nous allons désormais travailler, auprès des aînés académiques, et familiarisation avec celle-ci.
Semaine 7	Adaptation du frontend de Let's Go au nôtre.
Semaine 8	Apprentissage de la technologie Spring Boot pour le backend.
Semaine 9	Réalisation du backend, car l'application Let's Go était uniquement frontend.
Semaine 10	Mise en place de l'API de génération et de scan de code QR.
Semaine 11	Implémentation de l'API pour le code QR.
Semaine 12	Tests et réajustements.
Semaine 13	Déploiement de l'application avec scan de code QR pour l'authentification.
Semaine 15-16	Rédaction du rapport final.

TABLE 1 – Planning du projet

A.2. L'API de génération et de scan de QR codes pour un système d'authentification

Présentation Générale

La QRAPI est une API REST développée en Spring Boot qui permet de :
— Générer des QR codes contenant des informations relatives à une course (identifiants client, chauffeur, course, localisation, etc.)
— Scanner et valider ces QR codes pour confirmer une course ou enregistrer une action (par exemple, mettre à jour l'historique d'un scan)
— Sécuriser l'accès aux endpoints via l'authentification par token JWT

Note sur l'authentification

Bien que l'exemple présente ici les endpoints pour générer et scanner un QR code, l'obtention d'un token JWT (via l'endpoint `/api/reserve`) est nécessaire pour sécuriser l'accès. Cet endpoint d'authentification est attendu dans l'architecture globale mais n'est pas détaillé dans ce contrôleur.

Chaque utilisateur doit obtenir un token JWT (généralement via `/api/reserve`) et l'inclure dans l'en-tête HTTP :

```
Authorization: Bearer <TOKEN>
```

lors des appels aux endpoints suivants.

Endpoints Principaux

a) Authentification – Réservation du Token

Remarque : Cet endpoint n'est pas implémenté dans ce contrôleur. Il est supposé exister dans le cadre de l'API globale pour générer un token JWT.
— **URL** : `/api/reserve`
— **Méthode** : `POST`
— **Paramètres** :
— `fournisseur` (passé dans le corps de la requête ou en query string)
Exemple d'appel avec cURL :

```
curl -X POST "http://localhost:8080/api/reserve?fournisseur=Fra%20Yuuki" \
     -H "Content-Type: application/json"
```

Réponse attendue :

```
{
    "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJGcmEgWXV1a2kiLCJpYXQiOjE2MzMyNDk5OTl9.XXXX
}
```

b) Génération du QR Code

— **URL** : `/api/qr/generate`
— **Méthode** : `POST`
— **Paramètres** :
— **Query parameters** :

- `secret` : La clé utilisée pour signer le QR code (doit respecter la longueur minimale pour HS256, généralement 32 caractères ou plus)
- `expirationMillis` : La durée de validité (en millisecondes) du token signé qui sera intégré dans le QR code

— **Corps (JSON)** :

```
{
  "clientId": 123456,
  "chauffeurId": 654321,
  "courseId": 987654,
  "lieu": "Gare Centrale",
  "heure": "14:30",
  "date": "2025-02-07",
  "ville": "Paris",
  "pays": "France"
}
```

— **Authentification** :

— Le token JWT (obtenu via `/api/reserve`) doit être transmis dans l'en-tête :

```
Authorization: Bearer <TOKEN>
```

Fonctionnement :

1. Le contrôleur attribue à l'objet `QRData` un identifiant unique (UUID) et enregistre le nom du fournisseur (extrait du `Principal`).
2. Les données de `QRData` sont converties en chaîne (via `toString()`, par exemple) puis hachées avec l'algorithme SHA-256.
3. Le hash est ensuite signé en créant un token JWT, à l'aide de la clé `secret` et d'une durée de validité définie par `expirationMillis`.
4. Le token signé est converti en image QR (au format PNG) grâce à la bibliothèque ZXing.

Exemple d'appel cURL :

```
curl -X POST "http://localhost:8080/api/qr/generate?secret=MaSuperCleSecrete&expirationM
     -H "Content-Type: application/json" \
     -H "Authorization: Bearer <TOKEN>" \
     -d '{
           "clientId": 123456,
           "chauffeurId": 654321,
           "courseId": 987654,
           "lieu": "Gare Centrale",
           "heure": "14:30",
           "date": "2025-02-07",
           "ville": "Paris",
           "pays": "France"
         }'
```

Réponse :

Le serveur renvoie une image PNG (sous forme de bytes) contenant le QR code généré. Le header de la réponse sera défini sur `Content-Type: image/png`.

c) Scan du QR Code

URL :

`/api/qr/scan`

Méthode :

`POST`

Paramètres :

- **Query parameters :**
  - `qrCodeData` : Le token JWT (issu du QR code) contenant le hash signé
  - `secret` : La clé utilisée pour vérifier la signature du token
- **Corps (JSON) :**
  - Un objet `History` qui contient des informations supplémentaires à enregistrer lors du scan (par exemple, des champs personnalisés relatifs à l'opération)

Authentification :

Le header HTTP doit contenir :

```
Authorization: Bearer <TOKEN>
```

Fonctionnement :

- Le contrôleur décode et vérifie la signature du token contenu dans `qrCodeData` en utilisant la clé `secret`.
- Si la signature n'est pas valide, la réponse est un HTTP 401 avec le message « QR Code invalide ! ».
- En cas de signature valide, le service de scan (via `ScanService`) récupère l'objet `QRHash` associé.
- À partir du `QRHash`, le contrôleur recherche les données complètes dans la table `QRData`.
- L'objet `History` est complété avec des informations extraites de `QRData` (telles que `clientId`, `chauffeurId`, `courseId`, `fournisseur`) et sauvegardé dans la base.

Exemple d'appel cURL :

```
curl -X POST "http://localhost:8080/api/qr/scan?qrCodeData=<JWT_QR_CODE>&secret=MaSuperC
     -H "Content-Type: application/json" \
     -H "Authorization: Bearer <TOKEN>" \
     -d '{
           "someHistoryField": "valeur",
           "autreChamp": "valeur"
         }'
```

Réponse :

- En cas de succès, le serveur renvoie l'objet `QRData` correspondant (ou `null` s'il n'est pas trouvé) au format JSON, avec un status HTTP 200.
- En cas d'erreur de signature, une réponse HTTP 401 est renvoyée.
- En cas d'exception, une réponse HTTP 400 avec un message d'erreur est renvoyée.

A.3. Guide de Déploiement

3.1. Prérequis

— Java 11 ou supérieur (idéalement Java 17 ou 23)
— Maven ou Gradle pour la construction
— Cassandra (installé localement, en Docker ou sur un cluster distant)
— Docker (optionnel, pour containeriser l'application et/ou la base de données)

3.2. Configuration et Préparation

Configurer la base de données Cassandra

Assurez-vous que Cassandra est installé et accessible. Exemple avec Docker :

```
docker run --name cassandra -d -p 9042:9042 cassandra:latest
```

Puis, dans `cqlsh`, créez (si nécessaire) le keyspace :

```
CREATE KEYSPACE IF NOT EXISTS transportapp
WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
```

Configurer l'application

Dans le fichier `src/main/resources/application.properties` (ou `application.yml`), configurez Cassandra :

```
spring.application.name=QRAPI
spring.data.cassandra.contact-points=127.0.0.1
spring.data.cassandra.port=9042
spring.data.cassandra.keyspace-name=transportapp
spring.data.cassandra.local-datacenter=datacenter1
spring.data.cassandra.schema-action=create-if-not-exists
server.port=8080
```

Vérifier les dépendances Maven

Assurez-vous d'inclure dans votre `pom.xml` les dépendances suivantes (et toute autre nécessaire, par exemple pour JJWT et ZXing) :

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-cassandra</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<!-- Dépendances pour JJWT, ZXing, etc. -->
```

3.3. Construction et Lancement

Construire l'application

Avec Maven, exécutez :

```
mvn clean install
```

Lancer l'application

Exécutez :

```
java -jar target/QRAPI-0.0.1-SNAPSHOT.jar
```

L'application sera accessible sur `http://localhost:8080`.

3.4. Déploiement via Docker (Optionnel)

Créer un Dockerfile à la racine du projet :

```
FROM openjdk:17-jdk-alpine
COPY target/QRAPI-0.0.1-SNAPSHOT.jar /app/QRAPI.jar
WORKDIR /app
EXPOSE 8080
CMD ["java", "-jar", "QRAPI.jar"]
```

Construire l'image Docker :

```
docker build -t qrapi .
```

Lancer le conteneur Docker :

```
docker run -p 8080:8080 qrapi
```

A.4. Analyse du probleme

4.1. Identification formelle du probleme

Notre identifie plusieurs défis majeurs, notamment un manque de confiance et de fiabilité dans le processus d'authentification des utilisateurs, ce qui peut compromettre la vérification des identités. De plus, l'intégrité et la protection des données sensibles représentent des enjeux critiques, exposant le système à des risques de manipulation ou de fuite d'informations. Enfin, la gestion et le contrôle des accès aux informations demeurent une problématique essentielle, nécessitant des mécanismes robustes pour garantir une utilisation sécurisée et conforme aux droits des utilisateurs.

Fonctions

- **Génération du QR Code** Une fonction $f : R \to Q$ qui prend une course et génère un QR code unique, en calculant :
  - Un hash des données (via SHA-256)
  - Un token signé (JWT) avec une clé secrète et une durée d'expiration
- **Validation du QR Code** Une fonction $g : Q \to \{\text{valide}, \text{invalide}\}$ qui, en vérifiant la signature du token, détermine la validité du QR code.

4.3. Solution formelle

Pour répondre aux problématiques identifiées, nous proposons une solution basée sur la génération et la vérification de QR codes pour assurer l'authenticité et l'intégrité des données échangées entre les parties prenantes (chauffeurs et clients). Cette solution repose sur les mécanismes suivants :

- **Authentification sécurisée :** L'utilisation de QR codes uniques, générés avec des algorithmes cryptographiques (ex. SHA-256 et JWT), permet de garantir la fiabilité des interactions et d'éviter les falsifications.
- **Intégrité des données :** Chaque QR code inclut un hash des informations critiques de la course, assurant que toute tentative de modification sera détectée.
- **Contrôle des accès :** Une gestion rigoureuse des permissions garantit que seules les parties autorisées (chauffeurs et clients associés à une course) peuvent accéder ou valider les informations via des scans en temps réel.
- **Traçabilité :** Un historique détaillé des scans (H) est conservé, permettant un suivi transparent et immuable des interactions pour chaque course.
- **Expiration et renouvellement :** Chaque QR code est valide pour une durée limitée, réduisant les risques d'utilisation frauduleuse ou de réutilisation des informations.

Cette solution formelle assure non seulement la confiance entre les utilisateurs, mais également une conformité avec les meilleures pratiques en matière de sécurité et de gestion des données sensibles.

4.4. Cas d'Utilisation

Scénario 1 : Génération d'un QR Code

- **Acteur :** Fournisseur (ex. chauffeur)
- **Préconditions :**
  - La course est créée et enregistrée.
  - L'utilisateur est authentifié (token JWT valide obtenu via `/api/reserve`).
- **Flux Principal :**
  - L'utilisateur envoie une requête `POST /api/qr/generate` avec les données de la course, le secret et la durée d'expiration.
  - Le système attribue un UUID à la course, calcule un hash, signe ce hash et génère le QR code correspondant.
  - Le QR code (image PNG) est renvoyé au client.
- **Postconditions :**
  - Le QR code ainsi que les informations associées (dans `QRData` et `QRHash`) sont enregistrés dans la base.

Scénario 2 : Scan et Validation d'un QR Code

- **Acteur :** Fournisseur (ex. chauffeur ou agent de vérification)
- **Préconditions :**
  - Le QR code a été généré et est en possession de l'utilisateur.
  - L'utilisateur est authentifié (token JWT valide).
- **Flux Principal :**
  - L'utilisateur scanne le QR code et envoie une requête `POST /api/qr/scan` avec :
    - Le token JWT issu du QR code (`qrCodeData`)
    - Le même secret utilisé pour la signature
    - Un objet `History` contenant des informations supplémentaires

  - Le système vérifie la signature du token.
  - En cas de succès, il récupère les données associées (via `QRHash` et `QRData`) et met à jour l'historique.
  - Le système renvoie l'objet `QRData` correspondant (ou `null` s'il n'est pas trouvé).
- **Flux Alternatif :**
  - Si la signature est invalide ou le QR code n'est pas trouvé, une erreur (HTTP 401 ou 404) est renvoyée.
- **Postconditions :**
  - L'opération de scan est enregistrée dans la base (via `History`).

A.5. Structure du Code et Diagramme de Classes

5.1. Principales Classes

- **QRData**
  - Contient les informations de la course (client, chauffeur, course, lieu, heure, date, ville, pays, fournisseur).
- **QRHash**
  - Enregistre le hash signé associé à une instance de `QRData` et la référence à son identifiant.
- **History**
  - Stocke l'historique des scans avec les identifiants et, éventuellement, un timestamp.

5.2. Services et Contrôleurs

- **QRCodeController**
  - Gère les endpoints `/api/qr/generate` et `/api/qr/scan`.
- **ScanService**
  - Contient la logique métier de validation et de traitement des scans.
- **JwtUtil**
  - (Optionnel) Gère la signature et la vérification des tokens JWT.

5.3. Diagramme de Classes (UML Simplifié)

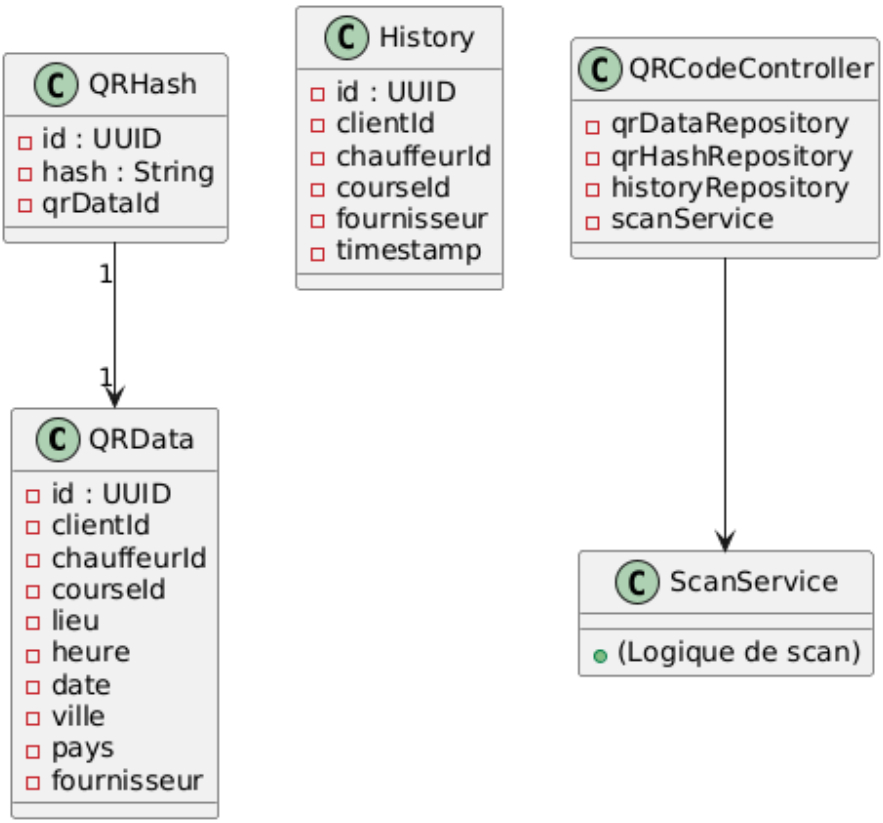

FIGURE 11 – Diagramme de classes simplifié

Documentation de l'API de QR Code

4 Identification formelle du probleme

Une API de génération et de scan de codes QR répond à plusieurs problématiques clés dans divers domaines, en offrant une solution rapide, universelle et sécurisée pour le partage d'informations. Elle simplifie les processus complexes, comme l'authentification, les paiements ou la traçabilité, en automatisant la transmission de données via des codes QR faciles à lire sur n'importe quel appareil compatible. En intégrant des mécanismes de chiffrement ou des signatures numériques, une telle API garantit également la sécurité et la non-répudiation, rendant les échanges de données sensibles plus fiables. Elle répond à la demande croissante des entreprises et développeurs d'intégrer rapidement des fonctionnalités de génération et de scan dans leurs systèmes sans devoir tout recréer de zéro. De plus, cette API standardise les outils de génération et de scan en une solution unique, adaptée à des cas d'utilisation variés comme la gestion de paiements numériques, la validation d'identités, la traçabilité logistique ou encore la gestion d'événements. En résumé, elle offre une plateforme flexible, performante et sécurisée pour répondre aux besoins modernes d'interopérabilité et d'accessibilité des données.

5 Modélisation mathématique

5.1 Analyse Mathématique du Scan et de la Lecture d'un QR Code

5.1.1 Détection et Correction de Perspective

Identification des motifs fixes du QR Code : Le scanner détecte les motifs caractéristiques :

- **Finder patterns** : 3 grands carrés aux coins.
- **Alignment patterns** : petits carrés permettant la correction des distorsions.

Détection des bords (Filtre de Sobel) : On applique un filtre de Sobel pour détecter les contours de l'image :

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{1}$$

Le gradient de l'image est donné par :

$$G = \sqrt{G_x^2 + G_y^2} \tag{2}$$

Détection des lignes (Transformée de Hough) : L'équation d'une droite est :

$$y = mx + b \tag{3}$$

La transformée de Hough paramétrique est définie par :

$$\rho = x \cos\theta + y \sin\theta \tag{4}$$

Correction de perspective (Homographie) : L'homographie est décrite par une matrice H :

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = H \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5}$$

avec :

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{6}$$

5.1.2 Binarisation et Segmentation des Données

Méthode d'Otsu : Le seuil optimal T est défini par :

$$\sigma_B^2(T) = w_0(T)w_1(T)(\mu_0(T) - \mu_1(T))^2 \tag{7}$$

Les pixels sont ensuite convertis en noir et blanc :

$$I(x,y) = \begin{cases} 1, & \text{si } I(x,y) \geq T \\ 0, & \text{sinon} \end{cases} \tag{8}$$

5.1.3 Décodage des Données

Les bits sont extraits en suivant un parcours en zig-zag et convertis en texte ASCII.

5.1.4 Correction des Erreurs avec Reed-Solomon

Calcul des syndromes : On évalue le polynôme reçu $c(x)$ aux puissances de α :

$$S_j = c(\alpha^j), \quad j = 0, ..., 2t-1 \tag{9}$$

Localisation des erreurs (Algorithme de Berlekamp-Massey) : On cherche un polynôme de localisation :

$$\Lambda(x) = 1 + \lambda_1 x + \lambda_2 x^2 + ... + \lambda_t x^t \tag{10}$$

Correction des erreurs (Algorithme de Forney) : L'erreur est corrigée par :

$$e_i = -\frac{\Omega(\alpha^{-i})}{\Lambda'(\alpha^{-i})} \tag{11}$$

5.1.5 Conversion Finale et Affichage

Une fois les erreurs corrigées, les octets sont convertis en texte lisible par l'utilisateur.

5.1.6 Conclusion

Le processus de scan d'un QR code repose sur plusieurs concepts mathématiques :
— **Vision par ordinateur** : détection de formes et correction de perspective.
— **Traitement du signal** : binarisation et segmentation.
— **Théorie des codes** : correction d'erreurs avec Reed-Solomon.

Ces étapes garantissent une lecture robuste et rapide, même en présence de bruit.

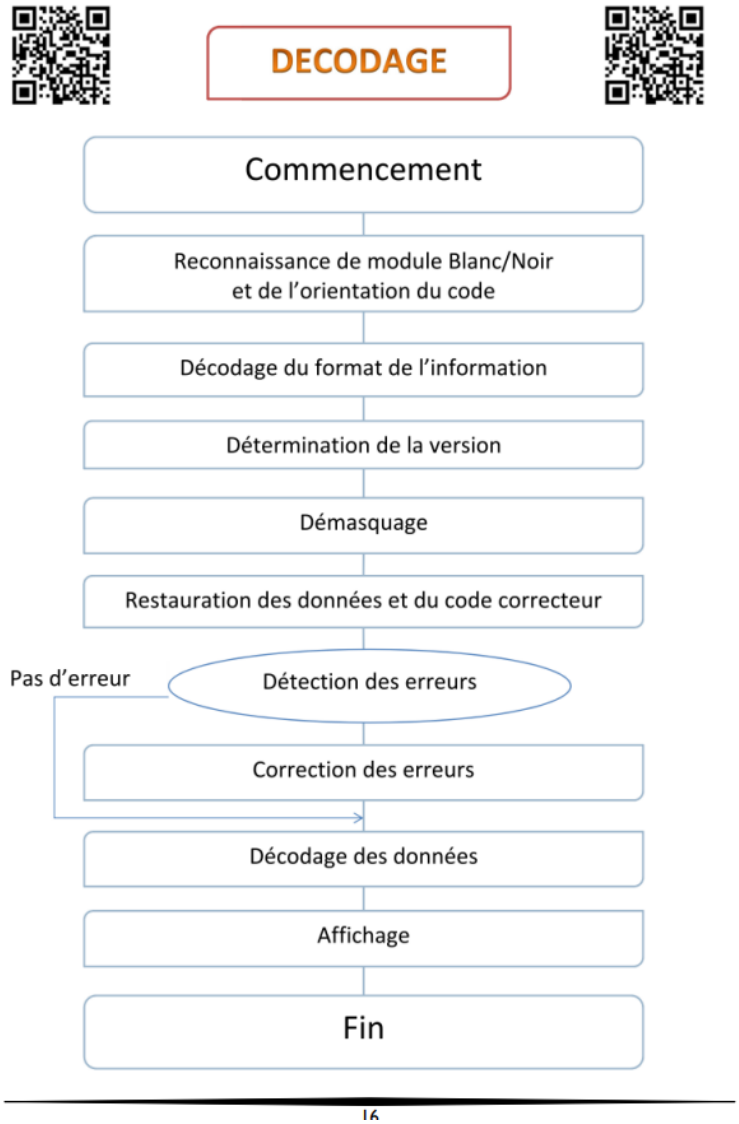

FIGURE 12 – Algorithme de décodage

5.2 Analyse Mathématique de la Génération de QR Code

La génération des QR codes repose sur des principes mathématiques avancés, notamment la théorie des champs finis, les codes correcteurs d'erreurs (codes de Reed-Solomon) et l'arithmétique modulaire. Voici une analyse détaillée :

5.2.1 Encodage des Données et Représentation en Champ Fini

Conversion des Caractères en Binaire : Le texte est converti en code ASCII, puis en binaire. Par exemple, pour le mot **"QR"** :
- 'Q' (ASCII 81) → 01010001
- 'R' (ASCII 82) → 01010010

Représentation Polynômiale des Octets : Chaque octet est représenté comme un polynôme de degré 7 dans un corps fini $GF(2^8)$. Un octet $b_7b_6b_5b_4b_3b_2b_1b_0$ est converti en :

$$P(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 \tag{12}$$

Exemple pour 'Q' (01010001) :

$$x^6 + x^4 + 1 \tag{13}$$

Cela signifie que l'octet est traité comme un élément du champ $GF(2^8)$.

Construction du Champ Fini $GF(2^8)$: Le champ $GF(2^8)$ contient 256 éléments, notés comme des puissances d'un élément primitif α :

$$\alpha^0, \alpha^1, \alpha^2, ..., \alpha^{255} \tag{14}$$

Un polynôme primitif est utilisé pour générer ce champ, souvent :

$$p(x) = x^8 + x^4 + x^3 + x^2 + 1 \tag{15}$$

Les puissances successives de α permettent de représenter tous les octets comme des polynômes modulo $p(x)$.

5.2.2 Organisation et Correction d'Erreurs avec les Codes de Reed-Solomon

Principe des Codes de Reed-Solomon : Un code de Reed-Solomon $RS(n, k)$ fonctionne dans un champ $GF(2^8)$.
- k : nombre de symboles de données
- n : longueur du code (incluant les symboles de correction)
- $2t = n - k$: nombre de symboles de correction, permettant de corriger jusqu'à t erreurs.

Chaque bloc de données est transformé en polynôme de message :

$$m(x) = m_0 + m_1x + m_2x^2 + ... + m_{k-1}x^{k-1} \tag{16}$$

Génération du Polynôme de Correction : Un polynôme générateur $g(x)$ est défini pour insérer la redondance :

$$g(x) = (x - \alpha^0)(x - \alpha^1)...(x - \alpha^{2t-1}) \tag{17}$$

Exemple pour $t = 2$:

$$g(x) = (x - \alpha^0)(x - \alpha^1) = x^2 + \alpha x + 1 \tag{18}$$

Puis, on multiplie $m(x)$ par x^{2t} et on divise par $g(x)$ pour obtenir le polynôme de correction $s(x)$:

$$x^{2t}m(x) = q(x)g(x) + s(x) \tag{19}$$

Détection et Correction des Erreurs : Si des erreurs sont détectées, on calcule les syndromes S_j en évaluant :

$$S_j = c(\alpha^j) \text{ pour } j = 0, ..., 2t - 1 \tag{20}$$

Si $S_j = 0$, aucune erreur. Sinon, un algorithme comme Berlekamp-Massey est utilisé pour localiser et corriger les erreurs.

5.2.3 Placement des Données et Masquage

Structure de la Matrice QR Code : Un QR code est une matrice $n \times n$, où n dépend de la version du QR code :

- Version 1 : 21×21
- Version 2 : 25×25
- ...
- Version 40 : 177×177

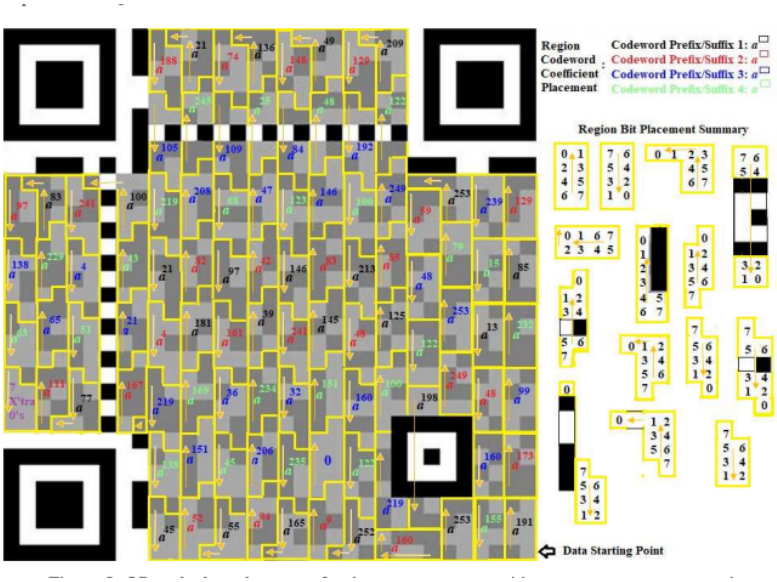

FIGURE 13 – Matrice QR Code

Masquage des Données : Le masquage évite les motifs répétitifs qui pourraient gêner la lecture. Huit motifs sont définis, comme :

$$(i + j) \mod 2 = 0 \tag{21}$$

Le motif optimal est choisi selon un score de pénalité.

5.2.4 Génération finale du QR code

Après placement et masquage, nous obtenons une matrice binaire de 21×21 prête à être convertie en QR code.

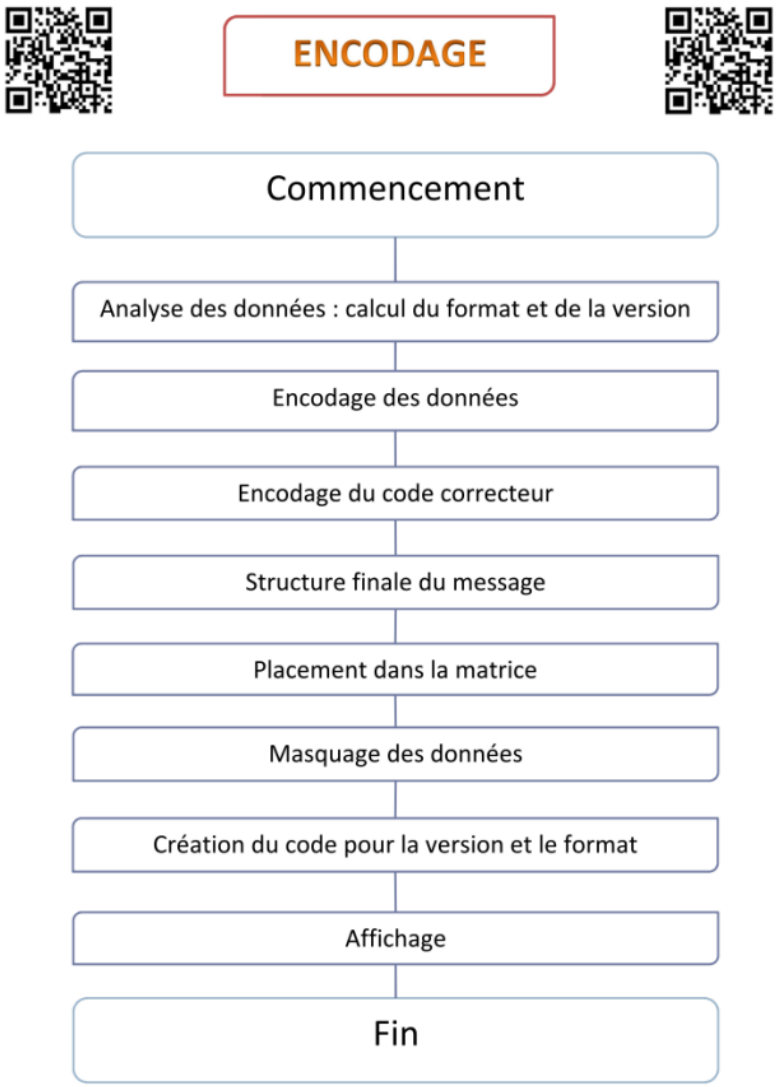

FIGURE 14 – Algotithme d'encodage

6 Solution formelle

La solution proposée repose sur la conception et le développement d'une API complète et polyvalente, capable de générer et de scanner des codes QR en réponse aux besoins identifiés. Cette API offre une interface centralisée accessible via des endpoints, permettant aux utilisateurs ou aux applications clientes d'intégrer facilement des fonctionnalités liées aux codes QR dans leurs systèmes. Pour la génération, l'API permet de personnaliser les codes QR selon plusieurs paramètres tels que le contenu, la taille, la couleur ou encore le niveau de correction d'erreur, tout en garantissant la conformité avec les standards universels. Pour le scan, l'API offre une capacité d'analyse rapide et précise des codes QR pour extraire les données encodées, avec des options supplémentaires pour vérifier l'intégrité ou la validité des informations, notamment via des mécanismes de chiffrement ou de signature numérique.

La solution est conçue pour être modulaire et évolutive, permettant d'ajouter des fonctionnalités supplémentaires en fonction des besoins futurs, comme la gestion de QR codes dynamiques ou l'intégration avec des systèmes de traçabilité ou d'authentification. En outre, l'API met l'accent sur la sécurité, en intégrant des techniques de chiffrement pour protéger les données sensibles, ainsi que sur la performance, avec un traitement optimisé pour des volumes élevés de requêtes. Cette solution formelle répond ainsi aux défis liés à l'interopérabilité, la flexibilité et la fiabilité, tout en offrant une expérience utilisateur fluide et efficace.

Références

[1] Langlet Marie,Castel Vincent, Legenvre Francois-Xavier, *Les QR codes*, Maths En Jeans, 2012-2013.