***Partie conceptionnel de CitizenService***

**Objectifs principaux :**

1. **Centraliser les services pour les citoyens :**
   * Créer une plateforme où les citoyens peuvent accéder à tous les services municipaux (démarches administratives, demandes, suivi, etc.) en un seul endroit.
2. **Gérer facilement les dossiers administratifs :**
   * Permettre aux citoyens de gérer leurs dossiers (naissance, mariage, permis, etc.) en ligne, avec la possibilité de suivre leur état en temps réel.
3. **Offrir des informations personnalisées :**
   * Fournir des informations spécifiques à chaque citoyen, selon son arrondissement ou sa localisation, pour mieux répondre à ses besoins.
4. **Permettre la soumission de réclamations :**
   * Offrir la possibilité aux citoyens de signaler des problèmes ou des retards dans leurs démarches administratives, et suivre l’avancement de leurs réclamations.
5. **Gérer les rendez-vous administratifs :**
   * Permettre aux citoyens de prendre des rendez-vous pour leurs démarches (retrouver un document, rencontrer un agent, etc.), et recevoir des rappels de ces rendez-vous.
6. **Assurer la sécurité des données :**
   * Garantir la protection des informations personnelles des citoyens avec un système d’authentification sécurisé.
7. **Utiliser une architecture flexible (Microservices) :**
   * Concevoir la plateforme avec des microservices, c’est-à-dire des services indépendants pour chaque fonctionnalité, afin de pouvoir améliorer ou ajouter des services facilement.
8. **Mettre en place un suivi en temps réel de la plateforme :**
   * Utiliser des outils pour surveiller la plateforme et réagir rapidement en cas de problème, pour garantir une bonne performance.
9. **Automatiser les tests et les mises à jour :**
   * Mettre en place des outils pour tester automatiquement le système et déployer les mises à jour de manière régulière sans interruption.

**Objectifs complémentaires :**

1. **Offrir une interface simple et agréable :**
   * Créer une interface facile à utiliser pour les citoyens, que ce soit sur le site web ou sur mobile.
2. **Assurer une bonne performance et une évolution facile :**
   * Utiliser des technologies qui permettent à la plateforme de bien fonctionner même en cas de forte utilisation, et de se développer facilement.
3. **Rendre la plateforme accessible à tous :**
   * S’assurer que tous les citoyens, y compris ceux ayant des handicaps, puissent utiliser la plateforme sans difficulté.

**UML :**

**Diagramme de cas d’utilisation**

**Une image contenant texte, diagramme, ligne, dessin

Le contenu généré par l’IA peut être incorrect.**

**Diagramme de classe :**

**Une image contenant texte, diagramme, Plan, Parallèle

Le contenu généré par l’IA peut être incorrect.**

**Microservices Design :**

**1. Service des Citoyens**

**Responsabilités :**

* Gérer l'authentification des citoyens via un système sécurisé.
* Gérer les informations personnelles des citoyens (nom, prénom, adresse, etc.).
* Permettre l'inscription, la mise à jour et la suppression des informations des citoyens.
* Fournir des API pour l'accès aux informations personnelles (en fonction des rôles ou permissions).

**Technologies utilisées :**

* **ASP.NET Core :** Pour le développement de l'API REST. Utiliser les middleware d'authentification et d'autorisation intégrés (JWT, OAuth2 via IdentityServer).
* **IdentityServer :** Pour la gestion de l'authentification via OAuth2 et OpenID Connect, afin de sécuriser l'API.
* **PostgreSQL / MySQL :** Base de données pour stocker les informations des citoyens.
* **Swagger/OpenAPI :** Pour la documentation des API REST, facilitant l’interopérabilité avec les autres services.

**Service des Dossiers Administratifs**

**Responsabilités :**

* Gérer la création, la modification, et la suppression des dossiers administratifs (naissance, mariage, permis, etc.).
* Gérer le statut des dossiers (en cours, validé, rejeté, etc.).
* Gérer les documents associés à chaque dossier (upload, suppression, etc.).
* Permettre aux citoyens d'obtenir l'état de leur dossier via une API.

**Technologies utilisées :**

* **ASP.NET Core :** Pour la création des API REST pour interagir avec les dossiers.
* **Entity Framework Core** : Pour la gestion des entités et des migrations de base de données.
* **PostgreSQL / SQL Server :** Base de données pour stocker les informations des dossiers administratifs et les documents associés.
* **RabbitMQ / Kafka** : Pour gérer les notifications ou événements asynchrones lorsque l'état des dossiers change (par exemple, notifier le citoyen lorsque son dossier est validé**).**
* **Swagger/OpenAPI :** Pour la documentation et l'exposition des API.

**3. Service de Suivi des Demandes**

**Responsabilités :**

* Permettre aux citoyens de suivre en temps réel l’avancement de leurs demandes (état de traitement, validation, etc.).
* Fournir une API qui renvoie des informations sur le statut de la demande en temps réel.
* Gérer les événements de mise à jour du statut des demandes (par exemple, lorsque le dossier passe de "en cours" à "validé").

**Technologies utilisées :**

* **ASP.NET Core :** Pour le développement des API REST permettant aux citoyens de suivre l'état de leurs demandes.
* **SignalR (optionnel) :** Pour les mises à jour en temps réel sur l’état des demandes via WebSockets, permettant d’envoyer des notifications en temps réel.
* **RabbitMQ / Kafka** : Pour la gestion des événements asynchrones entre le service des dossiers et le service de suivi des demandes.
* **PostgreSQL / SQL Server :** Pour stocker l’historique des demandes et leur statut.
* **Prometheus et Grafana :** Pour le monitoring et le suivi des performances du service.

**4. Service des Rendez-vous**

**Responsabilités :**

* **Gérer la planification des rendez-vous pour les citoyens** concernant leurs démarches administratives.
* Permettre aux citoyens de prendre, annuler ou modifier des rendez-vous.
* Gérer le statut des rendez-vous (programmé, réalisé, annulé).
* Intégrer les calendriers des services de la municipalité pour éviter les conflits de rendez-vous.

**Technologies utilisées :**

* **ASP.NET Core** : Pour la création des API REST permettant de gérer les rendez-vous.
* **Entity Framework Core** : Pour la gestion des entités liées aux rendez-vous et aux relations avec d’autres services (par exemple, les dossiers administratifs).
* **PostgreSQL / SQL Server :** Base de données pour stocker les informations des rendez-vous.
* **RabbitMQ / Kafka** : Pour gérer les événements asynchrones comme les notifications de rendez-vous programmés ou annulés.
* **Swagger/OpenAPI** : Pour la documentation de l'API.
* **Prometheus et Grafana** : Pour surveiller les performances et la santé des services de rendez-vous.

**Technologies communes à tous les services :**

* **Docker :** Pour la containerisation de chaque microservice, facilitant ainsi le déploiement et la gestion des versions.
* **Kubernetes** : Pour l'orchestration des microservices en production.
* **CI/CD avec GitHub Actions, GitLab CI/CD ou Azure DevOps : Pour l’automatisation des déploiements et des tests.**

**Besoins Fonctionnels**

Les besoins fonctionnels décrivent les fonctionnalités spécifiques que le système doit fournir.

1. Gestion des Citoyens

* B1.1 : Le système doit permettre l'inscription, la mise à jour, et la suppression des informations personnelles des citoyens (nom, prénom, adresse, téléphone, etc.).
* B1.2 : Le système doit garantir l’authentification sécurisée des citoyens via IdentityServer avec des tokens JWT.
* B1.3 : Le système doit permettre aux citoyens de consulter et de modifier leurs informations personnelles via une interface sécurisée.
* B1.4 : Le système doit permettre aux citoyens de récupérer leur mot de passe en cas d'oubli.

2. Gestion des Dossiers Administratifs

* B2.1 : Le système doit permettre la création, modification, et suppression des dossiers administratifs (naissance, mariage, permis, etc.).
* B2.2 : Le système doit permettre de suivre l’état des dossiers (en cours, validé, rejeté).
* B2.3 : Le système doit permettre aux citoyens de soumettre des documents associés à chaque dossier (par exemple, carte d'identité pour une demande de permis).
* B2.4 : Le système doit envoyer des notifications à chaque mise à jour du dossier (par exemple, dossier validé ou rejeté).
* B2.5 : Le système doit permettre de visualiser l'historique de toutes les actions effectuées sur un dossier.

3. Suivi des Demandes

* B3.1 : Le système doit permettre aux citoyens de suivre l’avancement de leurs demandes (ex: "en cours", "validé", "rejeté").
* B3.2 : Le système doit fournir une interface claire pour afficher le statut des demandes en temps réel.
* B3.3 : Le système doit informer les citoyens de toute modification de statut de leurs demandes via notifications push, email ou SMS.

4. Gestion des Rendez-vous

* B5.1 : Le système doit permettre aux citoyens de prendre des rendez-vous pour diverses démarches administratives.
* B5.2 : Le système doit permettre de modifier, annuler, et consulter les rendez-vous.
* B5.3 : Le système doit envoyer des rappels de rendez-vous par email, SMS ou notifications push.
* B5.4 : Le système doit permettre de synchroniser les rendez-vous avec les agendas des agents administratifs.

Besoins Non Fonctionnels

Les besoins non fonctionnels décrivent les contraintes ou critères qui définissent la qualité du système, mais qui ne sont pas directement liés à des fonctionnalités spécifiques.

1. Performance

* NF1.1 : Le système doit garantir un temps de réponse inférieur à 500 ms pour les actions critiques (comme la soumission d’un dossier ou la prise de rendez-vous).
* NF1.2 : Les services doivent pouvoir gérer un grand nombre d'utilisateurs simultanés sans dégradation de la performance, en utilisant des mécanismes de scalabilité horizontale (via Kubernetes).

2. Scalabilité

* NF2.1 : Le système doit être capable de s’adapter à une augmentation du nombre d’utilisateurs, avec la possibilité de scaler les services horizontalement.
* NF2.2 : Chaque microservice doit pouvoir être scalé indépendamment en fonction des besoins.

3. Sécurité

* NF3.1 : Le système doit garantir la sécurité des données personnelles des citoyens via le chiffrement des informations sensibles, à la fois en transit (HTTPS) et au repos (cryptage des bases de données).
* NF3.2 : Le système doit utiliser IdentityServer pour gérer l’authentification et l’autorisation des utilisateurs en utilisant OAuth2 et OpenID Connect.
* NF3.3 : Le système doit implémenter des mécanismes de gestion des rôles et des permissions pour limiter l’accès aux données sensibles selon le rôle de l’utilisateur.

4. Disponibilité

* NF4.1 : Le système doit être disponible 99.9% du temps en production, avec des mécanismes de redondance pour garantir une haute disponibilité.
* NF4.2 : Les microservices doivent être conçus pour tolérer les pannes et garantir une continuité de service même en cas de défaillance d’un ou plusieurs services.

5. Résilience

* NF5.1 : Le système doit être résilient face aux pannes, avec des mécanismes comme les retries, les circuits breakers et les file d'attente asynchrones (RabbitMQ/Kafka) pour garantir la continuité des opérations.
* NF5.2 : Le système doit gérer de manière autonome les erreurs et les pannes de services sans impacter l'expérience utilisateur.

6. Maintenance et Extensibilité

* NF6.1 : Le système doit être facile à maintenir avec des logs détaillés pour le suivi des erreurs et des événements via Serilog.
* NF6.2 : Le système doit être conçu de manière modulaire pour faciliter l’ajout de nouvelles fonctionnalités ou services dans le futur (par exemple, ajout de nouveaux types de dossiers administratifs ou services supplémentaires).

7. Interopérabilité

* NF7.1 : Le système doit être capable de s’intégrer avec d’autres systèmes externes via des API ouvertes et bien documentées.
* NF7.2 : Le système doit fournir des mécanismes d’importation et d’exportation des données dans des formats standardisés (par exemple, CSV, JSON).

8. Observabilité et Monitoring

* NF8.1 : Le système doit intégrer des outils de surveillance tels que Prometheus et Grafana pour suivre les métriques de performance, les erreurs et les alertes.
* NF8.2 : Le système doit générer des logs structurés via Serilog pour faciliter le débogage et le suivi des événements.

9. Conformité Légale

* NF9.1 : Le système doit être conforme aux lois et réglementations locales concernant la gestion des données personnelles (par exemple, la RGPD en Europe).
* NF9.2 : Les citoyens doivent être informés de la collecte et de l’utilisation de leurs données personnelles, et ils doivent pouvoir gérer leurs préférences en matière de confidentialité.

**Règles de Gestion Fonctionnelles**

**1. Gestion des Citoyens**

* **RG1.1** : Un citoyen ne peut pas s’inscrire avec une adresse e-mail déjà enregistrée dans le système.
* **RG1.2** : Lors de l'inscription, un citoyen doit fournir une adresse e-mail et un mot de passe uniques pour valider son compte.
* **RG1.3** : Le mot de passe doit être sécurisé avec un algorithme de hachage (par exemple, **bcrypt**) avant d'être stocké dans la base de données.
* **RG1.4** : L'authentification du citoyen doit être effectuée via **OAuth2** et **OpenID Connect** via **IdentityServer**.
* **RG1.5** : Si un citoyen oublie son mot de passe, il peut réinitialiser son mot de passe via un lien sécurisé envoyé à son e-mail.
* **RG1.6** : Les citoyens doivent pouvoir modifier ou supprimer leurs informations personnelles (adresse, téléphone, etc.), mais ces changements doivent être validés et audités par le système.

**2. Gestion des Dossiers Administratifs**

* **RG2.1** : Chaque dossier administratif doit être lié à un citoyen unique.
* **RG2.2** : Un citoyen ne peut avoir qu'un seul dossier en cours pour chaque type de dossier (ex : une seule demande de permis de conduire en même temps).
* **RG2.3** : Le système doit valider tous les documents soumis avant de les accepter dans le dossier (vérification de la validité des formats de fichier, taille, et type de document).
* **RG2.4** : Un dossier administratif doit être validé ou rejeté dans un délai spécifique (par exemple, 30 jours après la soumission).
* **RG2.5** : Lors de l'approbation ou du rejet d'un dossier, une notification doit être envoyée au citoyen concerné.
* **RG2.6** : Un dossier rejeté ne peut pas être modifié, mais le citoyen peut en soumettre un nouveau avec des informations corrigées.

**3. Suivi des Demandes**

* **RG3.1** : Un citoyen doit pouvoir consulter l'état actuel de chaque demande en temps réel via l'interface.
* **RG3.2** : Le suivi des demandes doit être mis à jour automatiquement dès qu'une action significative est réalisée (par exemple, validation d’un dossier).
* **RG3.3** : Chaque mise à jour de l’état d’une demande doit générer une notification (par email ou via l’interface) pour informer le citoyen.
* **RG3.4** : Le statut des demandes doit être enregistré avec un historique complet (qui a effectué chaque action, à quel moment, etc.).

**4. Gestion des Rendez-vous**

* **RG5.1** : Un citoyen peut réserver un rendez-vous uniquement pour une démarche administrative spécifique.
* **RG5.2** : Un rendez-vous peut être annulé ou modifié jusqu'à 24 heures avant la date prévue.
* **RG5.3** : Les citoyens doivent recevoir un rappel de leur rendez-vous 24 heures avant celui-ci par notification ou email.
* **RG5.4** : Un rendez-vous annulé doit être marqué comme "annulé" et le citoyen doit être informé de cette annulation.

**Règles de Gestion Non Fonctionnelles**

**1. Sécurité**

* **RGNF1.1** : Tous les échanges de données entre le client et le serveur doivent être sécurisés avec **HTTPS**.
* **RGNF1.2** : Les mots de passe doivent être cryptés avant d’être stockés (utilisation d’un algorithme de hachage comme **bcrypt** ou **PBKDF2**).
* **RGNF1.3** : Le système doit utiliser des tokens JWT pour sécuriser l’accès aux API et éviter les attaques de type **Cross-Site Request Forgery (CSRF)**.
* **RGNF1.4** : Le système doit limiter le nombre de tentatives de connexion échouées pour prévenir les attaques par force brute.
* **RGNF1.5** : Les données sensibles (comme les informations personnelles des citoyens) doivent être chiffrées à la fois au repos (dans la base de données) et en transit (lors des communications avec l'API).

**2. Performance et Scalabilité**

* **RGNF2.1** : Le système doit répondre aux demandes utilisateur dans un délai inférieur à **500ms** pour toutes les actions critiques.
* **RGNF2.2** : Le système doit être capable de supporter un volume de **10 000 utilisateurs simultanés** sans dégradation de la performance.
* **RGNF2.3** : Les services doivent être capables de s’adapter automatiquement à une augmentation du trafic en utilisant **Kubernetes** pour gérer la montée en charge.
* **RGNF2.4** : Le système doit avoir une architecture évolutive qui permet d'ajouter de nouveaux services sans impacter les services existants.

**3. Disponibilité**

* **RGNF3.1** : Le système doit garantir une disponibilité de **99,9%** sur une période de 30 jours, avec des mécanismes de redondance en cas de panne.
* **RGNF3.2** : En cas de panne d'un service, le système doit être capable de rediriger le trafic vers un service de secours, assurant ainsi une continuité de service.

**4. Maintenance**

* **RGNF4.1** : Le système doit être facile à maintenir avec des outils de surveillance et de journalisation des erreurs (par exemple, **Prometheus**, **Grafana**, **Serilog**).
* **RGNF4.2** : Les mises à jour logicielles doivent être déployées de manière transparente, sans affecter les utilisateurs en production, grâce à des mécanismes de **CI/CD**.

**5. Conformité Légale**

* **RGNF5.1** : Le système doit être conforme aux réglementations en matière de protection des données personnelles (par exemple, **RGPD** en Europe).
* **RGNF5.2** : Les citoyens doivent être informés de la collecte et de l’utilisation de leurs données personnelles lors de l'inscription, et doivent avoir un accès facile à une politique de confidentialité.

**Schéma de Base de Données**

**1. Table : Citoyen**

Cette table contient les informations personnelles des citoyens inscrits.

| Colonne | Type | Description |
| --- | --- | --- |
| Id | INT | Identifiant unique du citoyen (PK) |
| FirstName | VARCHAR(100) | Prénom du citoyen |
| LastName | VARCHAR(100) | Nom de famille |
| Email | VARCHAR(100) | Adresse email (doit être unique) |
| PhoneNumber | VARCHAR(15) | Numéro de téléphone |
| Address | VARCHAR(255) | Adresse du citoyen |
| DateOfBirth | DATE | Date de naissance |
| DateRegistered | DATE | Date d'inscription dans la plateforme |
| CIN | VARCHAR(20) | Numéro de la carte d'identité nationale (CIN) du citoyen (facultatif). |

2**Table : DossierAdministratif**

Cette table représente les dossiers administratifs soumis par les citoyens.

| **Colonne** | **Type** | **Description** |
| --- | --- | --- |
| Id | INT | Identifiant unique du dossier (PK). |
| CitizenId | INT | Référence à l'ID du citoyen (FK vers Citoyen). |
| TypeDossierId | INT | Référence au type de dossier (FK vers TypeDossier). |
| Status | VARCHAR(50) | Statut du dossier (ex : en cours, validé, rejeté). |
| SubmissionDate | DATE | Date de soumission du dossier. |
| ValidationDate | DATE | Date de validation du dossier (si applicable). |
| IsCompleted | BOOLEAN | Indicateur qui détermine si le dossier est complet (true/false). |

3. Table : TypeDossier

Les différents types de dossiers administratifs.

| Colonne | Type | Description |
| --- | --- | --- |
| Id | INT | Identifiant unique du type de dossier (PK) |
| Name | VARCHAR(100) | Nom du type de dossier (ex : Permis, Naissance) |
| Description | TEXT | Description du type de dossier |

4. Table : Document

Les documents associés à un dossier administratif.

| Colonne | Type | Description |
| --- | --- | --- |
| Id | INT | Identifiant unique du document (PK) |
| DossierId | INT | Référence au dossier (FK vers DossierAdministratif) |
| Type | VARCHAR(50) | Type de document (ex : Identité, Justificatif) |
| FilePath | VARCHAR(255) | Chemin d'accès au fichier du document |
| UploadDate | DATE | Date d'upload du document |

5. Table : Rendezvous

Les rendez-vous pris par les citoyens pour des démarches administratives.

| Colonne | Type | Description |
| --- | --- | --- |
| Id | INT | Identifiant unique du rendez-vous (PK) |
| CitizenId | INT | Référence au citoyen (FK vers Citoyen) |
| AppointmentDate | DATETIME | Date et heure du rendez-vous |
| TypeDossierId | INT | Référence au type de dossier pour le rendez-vous (FK vers TypeDossier) |
| Status | VARCHAR(50) | Statut du rendez-vous (programmé, réalisé, annulé) |

**6.Table : Admin**

Les informations des administrateurs qui gèrent le système.

| **Colonne** | **Type** | **Description** |
| --- | --- | --- |
| Id | INT | Identifiant unique de l'administrateur (PK) |
| FirstName | VARCHAR(100) | Prénom de l'administrateur |
| LastName | VARCHAR(100) | Nom de famille de l'administrateur |
| Email | VARCHAR(100) | Email de l'administrateur |
| PhoneNumber | VARCHAR(15) | Numéro de téléphone de l'administrateur |

**7 . Table : Category**

Les catégories de documents associées aux dossiers administratifs.

| **Colonne** | **Type** | **Description** |
| --- | --- | --- |
| Id | INT | Identifiant unique de la catégorie (PK) |
| Name | VARCHAR(100) | Nom de la catégorie (ex : Permis, Passeport) |
| Description | TEXT | Description de la catégorie |

**8 . Table : DocumentType**

Les types de documents associés à un dossier administratif.

| Colonne | Type | Description |
| --- | --- | --- |
| Id | INT | Identifiant unique du type de document (PK) |
| Name | VARCHAR(100) | Nom du type de document (ex : Permis de conduire) |
| IsImportable | BOOLEAN | Indique si le citoyen doit importer ce document (True/False) |
| Category | VARCHAR(100) | Catégorie du document (ex : Permis, Passeport) |

Table : Notification

Cette table contient les informations relatives aux notifications envoyées aux citoyens (par exemple : via email, SMS, etc.).

| Colonne | Type | Description |
| --- | --- | --- |
| Id | INT | Identifiant unique de la notification (PK). |
| CitizenId | INT | Référence à l'ID du citoyen (FK vers Citoyen). |
| Type | VARCHAR(50) | Type de notification (ex : "Dossier validé", "Rendez-vous", etc.). |
| Message | TEXT | Message de la notification envoyée. |
| NotificationDate | DATETIME | Date et heure d'envoi de la notification. |
| Status | VARCHAR(20) | Statut de la notification (ex : "Envoyé", "En attente", "Échoué"). |
| Channel | VARCHAR(50) | Canal de communication (ex : "Email", "SMS", "Push"). |
| IsRead | BOOLEAN | Indique si la notification a été lue par le citoyen (true/false). |
| RelatedEntityId | INT | Référence à l'entité liée (par exemple un RendezvousId ou un DossierId si applicable). |
| RelatedEntityType | VARCHAR(50) | Type d'entité liée (par exemple "Rendezvous", "Dossier"). |

Table : Réclamation

Cette table représente les réclamations soumises par les citoyens concernant des problèmes ou des demandes particulières.

| Colonne | Type | Description |
| --- | --- | --- |
| Id | INT | Identifiant unique de la réclamation (PK). |
| CitizenId | INT | Référence à l'ID du citoyen (FK vers Citoyen). |
| Subject | VARCHAR(255) | Sujet de la réclamation (ex : "Problème avec un document", "Demande de renseignement", etc.). |
| Description | TEXT | Description détaillée du problème ou de la demande. |
| Status | VARCHAR(50) | Statut de la réclamation (ex : "En cours", "Résolu", "Clôturé"). |
| SubmissionDate | DATETIME | Date et heure de soumission de la réclamation. |
| ResolutionDate | DATETIME | Date de résolution de la réclamation (si applicable). |
| AdminId | INT | Référence à l'ID de l'administrateur qui a traité la réclamation (FK vers Admin). |
| ResolutionDetails | TEXT | Détails concernant la résolution de la réclamation (si applicable). |

**Diagramme Entité-Relation (ERD)**

**Citizen** ↔ **DossierAdministratif**

* Un citoyen peut avoir plusieurs dossiers (relation **1-N**).
* Un dossier est lié à un seul citoyen, via la clé étrangère **CitizenId** dans **DossierAdministratif**.

**DossierAdministratif** ↔ **TypeDossier**

* Un dossier est associé à un type spécifique, via la clé étrangère **TypeDossierId** dans **DossierAdministratif**.

**DossierAdministratif** ↔ **Document**

* Un dossier peut avoir plusieurs documents associés, via la clé étrangère **DossierId** dans **Document**.

**Citizen** ↔ **RéclamationAdministratif**

* Un citoyen peut soumettre plusieurs réclamations, via la clé étrangère **CitizenId** dans **RéclamationAdministratif**.
* Une réclamation est liée à un dossier spécifique, via **DossierId**.

**DossierAdministratif** ↔ **Rendezvous**

* Un dossier peut avoir plusieurs rendez-vous associés à lui.
* Un rendez-vous est lié à un citoyen et à un dossier spécifique via **CitizenId** et **TypeDossierId**.

**Citizen** ↔ **Notification**

* Un citoyen peut recevoir plusieurs notifications, via la clé étrangère **CitizenId** dans **Notification**.

**Hiérarchie :**

CitizenService.Domain

│

├── Entities

│ ├── Citizen.cs <-- Entité Citizen (représente un citoyen)

│ ├── DossierAdministratif.cs <-- Entité DossierAdministratif (représente un dossier)

│ ├── Document.cs <-- Entité Document (liée à un dossier administratif)

│ ├── Rendezvous.cs <-- Entité Rendezvous (réservation d'un rendez-vous)

│ ├── Reclamation.cs <-- Entité Reclamation (réclamation d'un citoyen)

│ ├── Admin.cs <-- Entité Admin (administrateur)

│ ├── Notification.cs <-- Entité Notification (représente une notification envoyée à un citoyen)

│ └── Category.cs <-- Entité Category (catégories des documents)

│

├── Aggregates

│ ├── CitizenAggregate.cs <-- Agrégat pour Citizen (gère les règles métiers autour du citoyen)

│ ├── DossierAggregate.cs <-- Agrégat pour DossierAdministratif

│ └── RendezvousAggregate.cs <-- Agrégat pour Rendezvous

│

├── Repositories

│ ├── ICitizenRepository.cs <-- Interface de repository pour Citizen

│ ├── IDossierRepository.cs <-- Interface de repository pour Dossier

│ ├── IRendezvousRepository.cs <-- Interface de repository pour Rendezvous

│ ├── IReclamationRepository.cs <-- Interface de repository pour Reclamation

│ ├── INotificationRepository.cs <-- Interface de repository pour Notification

│ └── IAdminRepository.cs <-- Interface de repository pour Admin

│

└── Services

├── CitizenService.cs <-- Service de domaine pour gérer la logique des citoyens

├── DossierService.cs <-- Service de domaine pour gérer la logique des dossiers

├── RendezvousService.cs <-- Service de domaine pour gérer les rendez-vous

├── NotificationService.cs <-- Service de domaine pour gérer les notifications

├── ReclamationService.cs <-- Service de domaine pour gérer les réclamations

└── AdminService.cs <-- Service de domaine pour gérer les administrateurs

CitizenService.Application

│

├── Interfaces

│ ├── ICitizenService.cs <-- Interface de service pour Citizen

│ ├── IDossierService.cs <-- Interface de service pour Dossier

│ ├── IRendezvousService.cs <-- Interface de service pour Rendezvous

│ ├── INotificationService.cs <-- Interface de service pour Notification

│ ├── IReclamationService.cs <-- Interface de service pour Reclamation

│ └── IAdminService.cs <-- Interface de service pour Admin

│

├── Services

│ ├── CitizenService.cs <-- Service pour gérer la logique des citoyens

│ ├── DossierService.cs <-- Service pour gérer la logique des dossiers

│ ├── RendezvousService.cs <-- Service pour gérer la logique des rendez-vous

│ ├── NotificationService.cs <-- Service pour gérer la logique des notifications

│ ├── ReclamationService.cs <-- Service pour gérer la logique des réclamations

│ └── AdminService.cs <-- Service pour gérer les administrateurs

│

├── DTOs

│ ├── CitizenDTO.cs <-- DTO pour Citizen (transfert des données entre API et Application)

│ ├── DossierDTO.cs <-- DTO pour Dossier

│ ├── RendezvousDTO.cs <-- DTO pour Rendezvous

│ ├── NotificationDTO.cs <-- DTO pour Notification

│ ├── ReclamationDTO.cs <-- DTO pour Reclamation

│ └── ErrorDTO.cs <-- DTO pour gérer les erreurs

│

└── Validation

├── CitizenValidator.cs <-- Validation des données des citoyens

├── DossierValidator.cs <-- Validation des dossiers administratifs

└── ReclamationValidator.cs <-- Validation des réclamations

CitizenService.Infrastructure

│

├── Data

│ ├── CitizenServiceDbContext.cs <-- Contexte de base de données avec Entity Framework Core

│ └── Migrations/

│ └── [Migration Files] <-- Fichiers de migration pour la base de données

│

├── Repositories

│ ├── CitizenRepository.cs <-- Implémentation du repository pour Citizen

│ ├── DossierRepository.cs <-- Implémentation du repository pour Dossier

│ ├── RendezvousRepository.cs <-- Implémentation du repository pour Rendezvous

│ ├── ReclamationRepository.cs <-- Implémentation du repository pour Reclamation

│ ├── NotificationRepository.cs <-- Implémentation du repository pour Notification

│ └── AdminRepository.cs <-- Implémentation du repository pour Admin

│

├── Services

│ ├── RabbitMQNotificationService.cs <-- Service pour envoyer des notifications via RabbitMQ

│ ├── EmailNotificationService.cs <-- Service pour envoyer des notifications par email

│ └── ExternalApiService.cs <-- Service pour les appels aux APIs externes (par exemple, SMS)

│

└── Configuration

├── RabbitMQConfig.cs <-- Configuration de RabbitMQ

├── EmailConfig.cs <-- Configuration pour l'email (SMTP)

└── AppSettings.json <-- Fichier de configuration global

CitizenService.API

│

├── Controllers

│ ├── CitizenController.cs <-- Contrôleur pour gérer les citoyens

│ ├── DossierController.cs <-- Contrôleur pour gérer les dossiers administratifs

│ ├── RendezvousController.cs <-- Contrôleur pour gérer les rendez-vous

│ ├── NotificationController.cs <-- Contrôleur pour gérer les notifications

│ ├── ReclamationController.cs <-- Contrôleur pour gérer les réclamations

│ └── AdminController.cs <-- Contrôleur pour gérer les administrateurs

│

├── Models (DTOs)

│ ├── CitizenDTO.cs <-- DTO pour Citizen

│ ├── DossierDTO.cs <-- DTO pour Dossier

│ ├── RendezvousDTO.cs <-- DTO pour Rendezvous

│ ├── NotificationDTO.cs <-- DTO pour Notification

│ ├── ReclamationDTO.cs <-- DTO pour Reclamation

│ └── ErrorDTO.cs <-- DTO pour gérer les erreurs

│

└── Swagger

└── SwaggerConfig.cs <-- Configuration de Swagger pour l'API

CitizenService.Tests

│

├── UnitTests

│ ├── CitizenServiceTests.cs <-- Tests unitaires pour CitizenService

│ ├── DossierServiceTests.cs <-- Tests unitaires pour DossierService

│ ├── RendezvousServiceTests.cs <-- Tests unitaires pour RendezvousService

│ ├── NotificationServiceTests.cs <-- Tests unitaires pour NotificationService

│ ├── ReclamationServiceTests.cs <-- Tests unitaires pour ReclamationService

│ └── AdminServiceTests.cs <-- Tests unitaires pour AdminService

│

└── IntegrationTests

├── ApiIntegrationTests.cs <-- Tests d'intégration pour l'API

├── DatabaseIntegrationTests.cs <-- Tests d'intégration avec la base de données

└── RabbitMQIntegrationTests.cs <-- Tests d'intégration pour RabbitMQ

**Une image contenant texte, capture d’écran, nombre, Police

Le contenu généré par l’IA peut être incorrect.Maquettes :**

**Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.**

Une image contenant texte, capture d’écran, Police, conception

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, Police, conception

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Police, conception

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.