CESIZen : Cube 3 groupe

CESI

Romain MURIER

Fadi BOUCHNAK

Table des matières

[Architecture technique 3](#_Toc202184078)

[Déploiement de l’application 3](#_Toc202184079)

[Méthode de déploiement 3](#_Toc202184080)

[Déploiement mobile 3](#_Toc202184081)

[Environnements 3](#_Toc202184082)

[Sécurité de l’infrastructure 3](#_Toc202184083)

[Accès réseau et ports 3](#_Toc202184084)

[Authentification 3](#_Toc202184085)

[Données personnelles 3](#_Toc202184086)

[Vulnérabilités couvertes 3](#_Toc202184087)

[Matrice de risques 3](#_Toc202184088)

[Conclusion 3](#_Toc202184089)

Introduction générale

Le projet (Re)sources relationnelles est une application mobile initié par le ministère. Cette plateforme vise à offrir aux citoyens un accès sécurisé et centralisé sur différentes ressources.

Le projet repose sur une architecture complète, divisée en trois composants principaux :

* Une **API REST** construite avec  et **Prisma**,
* Une **interface d’administration** en React,
* Une **application mobile** développée avec **React Native (Expo)**.

L’ensemble s’appuie sur une base de données PostgreSQL conteneurisée via Docker, et l’authentification est gérée par la solution Clerk, enrichie par une gestion des rôles stockée en base.

Ce document a pour objectif de détailler la **stratégie de déploiement** de l’ensemble de la plateforme ainsi que les **mesures de sécurité** mises en place pour garantir la protection des données, la fiabilité du système, et la conformité aux bonnes pratiques actuelles. Il s’inscrit dans une démarche de professionnalisation et de préparation à la livraison d’un projet technique complet.

# Architecture technique

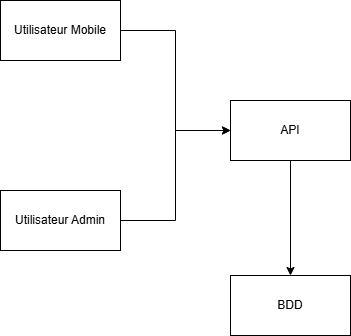
### Vue d’ensemble

L’architecture se compose de quatre blocs principaux :

* **Base de données PostgreSQL** : stocke toutes les données utilisateurs (hors mot de passe), les ressources, les accès aux différentes ressources, etc)
* **API Express** : gère la logique métier et l'accès aux données via Prisma (ORM), en mode App Router.
* **Frontend Admin (React)** : interface dédiée aux administrateurs.
* **Application mobile (Expo/React Native)** : destinée aux utilisateurs finaux, elle permet l’interaction avec toutes les fonctionnalités proposées.

Tous les services sont orchestrés à l’aide de **Docker Compose**, ce qui garantit un environnement cohérent, isolé et reproductible.

### Schéma d’architecture



Chaque front (mobile ou web) communique exclusivement avec l’API. L’API est le seul point d’accès à la base de données.

### Choix des technologies

| **Composant** | **Technologie choisie** | **Raisons du choix** |
| --- | --- | --- |
| **API** | Next.js + Prisma | App Router moderne, ORM fiable, facilité avec TypeScript |
| **Admin** | React + Vite | Rapide, léger, adapté au SPA administratives |
| **Mobile** | React Native (Expo) | Déploiement rapide, cross-platform natif |
| **Base de données** | PostgreSQL | Fiable, relationnel, largement supporté, bon support Prisma |
| **Conteneurisation** | Docker + Compose | Standard, cohérence environnementale, facilité de déploiement |
| **Auth** | Clerk + rôles BDD | Authentification SaaS simple à intégrer, surcouche personnalisée |

# Déploiement de l’application

### Méthode de déploiement

Le déploiement se fait via la CI/CD intégré à Github.  
  
La CI/CD se met en marche uniquement si un push est fait sur develop, afin de déployer l’environnement de dev, et sur master pour l’environnement de production.

Elle est composée de 4 blocks :

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

Le contenu généré par l’IA peut être incorrect.

* Build & Push docker image : Push l’image docker
* Deploy DEV to Server : Si push sur develop, se connecte au serveur pour déployer en environnement de dev
* Deploy PROD to Server : Si push sur master, se connecte au serveur pour déployer en environnement de prod
* SonarCloud Scan : effectue un scan SonarCloud.

### Déploiement mobile

Le déploiement mobile se fait via EAS de Expo.  
Il permet de générer un APK pour Android, ou IPA pour IOS.

Une fois fait, le fichier est à fournir aux utilisateur, comme par exemple à travers le Play Store ou l’Apple Store.

### Environnements

Comme dit précédemment, nous avons 2 environnement :

* Développement
* Production

Les ports accessibles sont donc différents pour permettre l’accessibilité aux différents environnements simultanément.

* API
  + Prod : 4000
  + Dev : 3000
* Nginx
  + Prod : 80 / 443
  + Dev : 8080
* Front
  + Prod : 5174
  + Dev : 5173
* Postgres :
  + Prod : 5431
  + Dev : 5432

# Sécurité de l’infrastructure

### Infrastructure

L’infrastructure est géré par AWS.

Nous avons donc un EC2 (Elastic Compute Cloud), qui est un serveur virtuel.

Afin de garantir une IP fixe, nous lions une IP Elastic à notre serveur.

### Accès réseau et ports

Afin de garantir une sécurité accrue, l’accès au serveur est restreint.

Nous pouvons limiter l’accès à certains ports uniquement, ainsi qu’à certaines IP.

Pour se faire, nous accordons l’accès uniquement aux ports qui concernent notre application (4000,3000,80,443,8080,5174,5173,5431,5432, ainsi que le port 22 pour la connexion SSH).

### Authentification

L’authentification est gérée par Clerk, ainsi que par une surcouche géré en base de données afin de gérer le rôle de l’utilisateur (Citoyen / Modérateur / Administrateur / Super Administrateur).

L’inscription Clerk gère l’authentification et génère un id unique pour chaque utilisateur. Cet ID est récupéré par l’application, puis inséré en base de données.

Avec ces identifiants, un JWT est mis en place, afin de sécuriser les requêtes.  
L’access token est un token à durée limitée et permet avec une clé secrète de vérifier que l’utilisateur qui fait la requête est bien l’utilisateur à l’origine, et non une usurpation.

Ce token est valide pendant 9 minutes. Pour récupérer un autre token, nous avons accès à un autre token, qui s’appelle un Refresh Token, qui a une validité de 24 heures et qui permets de regénérer un access token. Si le refresh token n’est plus valide, l’utilisateur doit alors se reconnecter à l’application.

### Données personnelles

Toute donnée personnelle est chiffré.

La partie d’authentification est donc géré par Clerk, et contient donc une sécurité accrue, comme avec un hashage du mot de passe avec sel et poivre, permettant ainsi de pouvoir vérifier le mot de passe sans permettre à une personne malveillante de le récupérer, même s’il a accès à la base de données, avec un sel et poivre qui permet d’éviter l’usage d’une raimbow table.

Les données personnelles autre que les mots de passes, comme les emails sont chiffrés avec une clé secrete, qui permet de récupérer cet information (comme un envoi de mail), sans pour autant mettre en danger ces informations en cas de fuite de données.



Ce chiffrement est géré par BCrypt, qui permet de chiffrer avec plusieurs itérations afin de rendre ce chiffrement encore plus fort.

### Vulnérabilités couvertes

Les vulnérabilités sont couvertes par le développeur, ainsi que par un un scan SonarCloud, qui est une version en ligne et gratuite (pour les projets open-sources) et permet d’avoir un tableau de bord avec les vulnérabilités, la maintenabilité, la couverture (tests unitaires) ou encore la duplication de code.

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

Le contenu généré par l’IA peut être incorrect.

Avec ce scan régulié (à chaque push sur master), nous pouvons par exemple bloquer le déploiement si la note n’est pas de A ou de B, afin de ne pas mettre en production une version qui ne fonctionne pas, ou mal.

# Matrice de risques

| **Identifiant** | | **Risque** | | **Criticité** | | **Mesure préventive** | | **Mesure corrective** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R1 | | Perte de données (base de données) | | Élevée | | Utilisation d’un volume Docker dédié (postgres\_data) et backups réguliers avec pg\_dump | | Restauration via pg\_restore depuis un dump local ou distant |
| R2 | | Faille d’authentification ou usurpation | | Élevée | | Authentification via Clerk + vérification du rôle en BDD | | Révocation de session, désactivation immédiate du compte |
| R3 | | Attaque par injection (SQL, XSS) | | Moyenne | | Utilisation de Prisma (protection injections), validation côté front + backend | | Nettoyage des entrées invalides, correctif immédiat |
| R4 | | Saturation ou déni de service (DoS / DDoS) | | Moyenne | | Rate limit sur l’API + utilisation de | | Redémarrage des conteneurs, limitation des connexions en amont |
| R5 | | Vol ou fuite de données personnelles | | Élevée | | Chiffrement des données sensibles (AES, SHA-512), accès restreint, protocole HTTPS | | Notification RGPD, rotation des secrets, traçabilité |
| R6 | | Erreur humaine lors d’un déploiement | | Faible | | Docker Compose versionné, documentation technique claire, scans sonar | | Rebuild des conteneurs / env kubernetes |
| R7 | Problème réseau entre mobile et backend | | Moyenne | | Affichage d’un message d’erreur côté Expo l’API n’est pas atteignable | | Redémarrage du tunnel Expo, vérification | |
| R8 | Corruption du fichier .env | | Faible | | Fichiers .env versionnés partiellement, stockés localement | | Restauration à partir de sauvegarde ou du readme | |

# Conclusion

Le projet s’inscrit dans une démarche concrète de développement et de mise en œuvre d’une plateforme web et mobile au service des citoyens. À travers ce travail, nous avons pu mettre en place une architecture modulaire, sécurisée et conteneurisée, facilitant à la fois le déploiement local, les tests, ainsi que la maintenabilité du projet.

L’utilisation de technologies modernes telles que Docker, Express, Prisma, Expo, et Clerk nous ont permis de professionnaliser la gestion des environnements, des rôles utilisateurs et de la base de données. Les problématiques de sécurité ont été abordées dès la conception, avec la mise en place de bonnes pratiques de chiffrement, la centralisation des accès, ainsi que l’identification des risques critiques.