Civiscore rapport

1. Explication du sujet choisi et du problème résolu :

Le projet **Civiscore** vise à créer une plateforme d'évaluation participative des services publics par les citoyens.

Problème ciblé :

- Manque de visibilité sur la qualité des services publics
- Absence d'outils simples pour recueillir l'opinion des usagers
- Données de satisfaction souvent centralisées et non accessibles

Objectif :

- Donner la parole aux utilisateurs
- Centraliser les évaluations de manière structurée et visuelle
- Offrir aux gouvernements, ONG et citoyens une vision claire et interactive de la qualité des services dans différents pays

2. Description des fonctionnalités implémentées

uthentification et gestion utilisateur

- Inscription avec stockage sécurisé (hash + JWT)
- Connexion/déconnexion avec refresh token
- Gestion des rôles (utilisateur, administrateur)
- Accès restreint aux routes selon le rôle
- Modification de profil

Services publics

- CRUD complet sur les services (nom, catégorie, coordonnées, pays…)
- Association d'un service à un pays
- Calcul et affichage de la note moyenne

Évaluations

- Création/modification/suppression d’un avis
- Ajout de commentaires et notation par critères pondérés
- Affichage de tous les avis d’un service
- Moyenne agrégée par service et par pays

Critères d’évaluation

- Gestion des critères (nom, catégorie, poids)
- Association dynamique des critères aux avis

Interactions sociales

- Votes utiles sur les avis (positifs/négatifs)
- Signalement des avis abusifs (raison, description)

Visualisation interactive

- Globe 3D via Three.js (`globe.gl`)
- Affichage dynamique des services par pays
- Filtres actifs (pays, catégorie)

Interface

- Design responsive avec React + TailwindCSS
- Formulaires validés, messages d’erreur clairs
- Dashboard utilisateur (reviews, informations)

3. Analyse des défis rencontrés et solutions apportées

Gestion fine des autorisations pour les opérations critique

Défi

Certaines routes modifient les données sensibles (création, modification, suppression des critères d'évaluation).
Il faut :

- limiter ces actions aux administrateurs
- protéger l'API contre des accès non autorisés
- gérer les erreurs liées aux permissions de façon claire

Solution

- Utilisation de dépendances FastAPI personnalisées (`Depends(get_current_admin_user)`) sur les routes sensibles.
- Ces dépendances vérifient le rôle de l'utilisateur avant d'autoriser l'accès.
- En cas d'échec, FastAPI renvoie automatiquement une erreur 401 ou 403.
- Centralisation de la logique d'autorisation dans `get_current_admin_user`, évitant la duplication.

```python
@router.post("/", response_model=EvaluationCriteriaOut, status_code=status.HTTP_201_CREATED)
def create_criteria(
    criteria_in: EvaluationCriteriaCreate,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_admin_user)  # autorisation admin requise
):
    criteria = create_evaluation_criteria(db, criteria_in)
    return criteria
```

Gestion dynamique et robuste des données partielles dans l'interface :

Défi

Les données de services pour chaque pays peuvent être incomplètes ou absentes (ex. `healthcare` manquant).

Solution

Utiliser des types TypeScript avec propriétés optionnelles (`?`) pour refléter l'optionnalité des données.

Dans les fonctions qui génèrent les données (ex. `generateServiceData`, `generateComparisonData`), fournir une valeur par défaut à zéro (`|| 0`) pour éviter les erreurs.

```
const generateServiceData = () => {
  if (!country) return []

  const serviceTypes = ["Healthcare", "Education", "Transportation", "Utilities",
"Government"]
  return serviceTypes.map((type) => {
    const serviceKey = type.toLowerCase() as keyof CountryService
    return {
      name: `${type} System`,
      category: type,
      rating: country.services[serviceKey] || 0,
      reviews: Math.floor(Math.random() * 1000) + 200,
    }
  })
}
```