**Documentation API pour l'équipe frontend**

**Table des matières**

1. [Authentification](https://claude.ai/chat/d125cd2c-8e25-4051-bccb-9b411afbcfdf#authentification)
2. [Gestion du fichier électoral](https://claude.ai/chat/d125cd2c-8e25-4051-bccb-9b411afbcfdf#gestion-du-fichier-%C3%A9lectoral)
3. [Gestion des candidats](https://claude.ai/chat/d125cd2c-8e25-4051-bccb-9b411afbcfdf#gestion-des-candidats)
4. [Gestion des parrains](https://claude.ai/chat/d125cd2c-8e25-4051-bccb-9b411afbcfdf#gestion-des-parrains)
5. [Gestion des parrainages](https://claude.ai/chat/d125cd2c-8e25-4051-bccb-9b411afbcfdf#gestion-des-parrainages)
6. [Gestion des membres DGE](https://claude.ai/chat/d125cd2c-8e25-4051-bccb-9b411afbcfdf#gestion-des-membres-dge)

**Authentification**

**1. Connexion - Génération de token JWT**

* **Endpoint**: POST /login
* **Description**: Génère un jeton JWT après validation de l'email et du mot de passe
* **Paramètres**:
  + username: Email de l'utilisateur
  + password: Mot de passe de l'utilisateur
* **Format de la requête**:
* { "username": "admin@example.com", "password": "motdepasse123"}
* **Réponse**:
* { "access\_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...", "token\_type": "bearer"}
* **Codes d'erreur**:
  + 401 Unauthorized: Email ou mot de passe incorrect

**2. Récupérer les informations de l'utilisateur connecté**

* **Endpoint**: GET /me
* **Description**: Récupère les informations de l'utilisateur connecté
* **Authentification**: Bearer Token requis
* **Réponse**:
* { "id": 1, "email": "admin@example.com", "role": "admin"}
* **Utilisation**: Inclure le token dans l'en-tête HTTP Authorization: Bearer <token>

**3. Récupérer tous les membres (admin uniquement)**

* **Endpoint**: GET /membres
* **Description**: Récupère la liste de tous les membres
* **Authentification**: Bearer Token requis (rôle admin uniquement)
* **Réponse**: Liste des membres

**4. Créer un membre (admin uniquement)**

* **Endpoint**: POST /membres
* **Description**: Crée un nouveau membre
* **Authentification**: Bearer Token requis (rôle admin uniquement)
* **Format de la requête**:
* { "email": "nouveau@example.com", "password": "motdepasse123", "role": "user"}
* **Réponse**: Détails du membre créé

**5. Déconnexion**

* **Endpoint**: POST /logout
* **Description**: Permet d'indiquer à l'application frontend de supprimer le token
* **Réponse**:
* { "message": "Déconnexion réussie. Supprimez le token côté client."}
* **Note**: Pour JWT, la déconnexion est gérée côté client en supprimant le token

**6. Obtenir un token d'accès (OAuth2)**

* **Endpoint**: POST /token
* **Description**: Alternative pour obtenir un token JWT via OAuth2
* **Paramètres**:
  + username: Email de l'utilisateur
  + password: Mot de passe de l'utilisateur
* **Réponse**: Identique à l'endpoint /login

**Gestion du fichier électoral**

**1. Vérifier si un upload est autorisé**

* **Endpoint**: GET /statut-upload
* **Description**: Vérifie si un nouvel upload de fichier électoral est autorisé
* **Authentification**: Bearer Token requis
* **Réponse**:
* { "upload\_autorise": true}
* **Utilisation**: À vérifier avant de proposer l'interface d'upload à l'utilisateur

**2. Upload du fichier électoral**

* **Endpoint**: POST /upload-fichier-electoral
* **Description**: Upload et contrôle initial du fichier électoral
* **Authentification**: Bearer Token requis
* **Format**: Multipart Form Data
* **Paramètres**:
  + file: Fichier à uploader
  + checksum: Empreinte de validation du fichier
* **Réponse**:
* { "message": "Fichier téléchargé et validé avec succès", "tentative\_id": 123, "success": true}
* **Codes d'erreur**:
  + 400 Bad Request: Format de fichier invalide ou problème avec le fichier
  + 403 Forbidden: Upload non autorisé (fichier déjà en traitement)
* **Utilisation**:
* // Exemple avec FormDataconst formData = new FormData();formData.append('file', fichierElectoral);formData.append('checksum', calculChecksumFichier(fichierElectoral));fetch('/upload-fichier-electoral', { method: 'POST', headers: { 'Authorization': `Bearer ${token}` }, body: formData});

**3. Contrôler les électeurs**

* **Endpoint**: POST /controler-electeurs/{tentative\_id}
* **Description**: Contrôle la complétude et le format des données des électeurs
* **Authentification**: Bearer Token requis
* **Paramètres**:
  + tentative\_id: ID de la tentative d'importation (retourné par l'endpoint d'upload)
* **Réponse**:
* { "success": true, "message": "Tous les électeurs sont valides.", "statistiques": { "nbElecteursTotal": 1000, "nbElecteursValides": 980, "nbElecteursInvalides": 20, "typesErreurs": { "formatNumElecteur": 5, "donneeManquante": 15 } }, "peut\_valider": true}
* **Utilisation**: À appeler après l'upload réussi du fichier électoral pour vérifier les données

**4. Valider l'importation**

* **Endpoint**: POST /valider-importation/{tentative\_id}
* **Description**: Valide l'importation et transfère les données dans la table permanente
* **Authentification**: Bearer Token requis
* **Paramètres**:
  + tentative\_id: ID de la tentative d'importation
* **Réponse**:
* { "success": true, "message": "Importation validée avec succès. 980 électeurs importés.", "nb\_electeurs\_importes": 980}
* **Codes d'erreur**:
  + 400 Bad Request: Des électeurs invalides existent ou aucun électeur valide
  + 500 Internal Server Error: Erreur lors de la validation
* **Utilisation**: À appeler uniquement si peut\_valider est à true dans la réponse de l'endpoint de contrôle

**5. Obtenir les statistiques d'importation**

* **Endpoint**: GET /statistiques-importation/{tentative\_id}
* **Description**: Récupère les statistiques d'une tentative d'importation
* **Authentification**: Bearer Token requis
* **Paramètres**:
  + tentative\_id: ID de la tentative d'importation
* **Réponse**: Statistiques détaillées (identiques à celles retournées par l'endpoint de contrôle)
* **Utilisation**: Pour afficher un tableau de bord ou des graphiques sur l'importation

**6. Obtenir les électeurs problématiques**

* **Endpoint**: GET /electeurs-problematiques/{tentative\_id}
* **Description**: Récupère la liste des électeurs présentant des problèmes
* **Authentification**: Bearer Token requis
* **Paramètres**:
  + tentative\_id: ID de la tentative d'importation
  + type\_erreur (optionnel): Filtrer par type d'erreur
* **Réponse**:
* { "electeurs": [ { "numElecteur": "123456789", "nom": "Dupont", "prenom": "Jean", "erreurs": ["formatNumElecteur"] } ], "count": 1}
* **Utilisation**: Pour afficher la liste des électeurs à corriger à l'utilisateur

**7. Réinitialiser l'état d'upload (admin uniquement)**

* **Endpoint**: POST /reinitialiser-etat-upload
* **Description**: Réinitialise l'état d'upload pour permettre un nouvel import
* **Authentification**: Bearer Token requis (rôle admin uniquement)
* **Réponse**:
* { "success": true, "message": "État d'upload réinitialisé avec succès."}
* **Codes d'erreur**:
  + 403 Forbidden: L'utilisateur n'est pas administrateur
* **Utilisation**: En cas de blocage du processus d'importation ou pour permettre une nouvelle importation

**Gestion des candidats**

**1. Créer un candidat**

* **Endpoint**: POST /
* **Description**: Crée un nouveau candidat et génère un code de sécurité
* **Format de la requête**:
* { "numElecteur": "123456789", "email": "candidat@example.com", "telephone": "+123456789", "parti": "Parti politique", "profession": "Profession"}
* **Réponse**: Détails du candidat créé

**2. Obtenir tous les candidats**

* **Endpoint**: GET /
* **Description**: Retourne la liste de tous les candidats
* **Réponse**: Liste des candidats

**3. Obtenir un candidat**

* **Endpoint**: GET /{numElecteur}
* **Description**: Récupère un candidat avec son dernier code de sécurité actif
* **Paramètres**:
  + numElecteur: Numéro d'électeur du candidat
* **Réponse**: Détails du candidat avec son code de sécurité

**4. Mettre à jour un candidat**

* **Endpoint**: PUT /{numElecteur}
* **Description**: Mise à jour des informations du candidat
* **Paramètres**:
  + numElecteur: Numéro d'électeur du candidat
* **Format de la requête**:
* { "email": "nouveau@example.com", "telephone": "+987654321"}
* **Réponse**: Détails du candidat mis à jour

**5. Supprimer un candidat**

* **Endpoint**: DELETE /{numElecteur}
* **Description**: Supprime un candidat
* **Paramètres**:
  + numElecteur: Numéro d'électeur du candidat
* **Réponse**:
* { "message": "Candidat supprimé avec succès."}

**6. Générer un nouveau code de sécurité**

* **Endpoint**: POST /{numElecteur}/generer\_mdp
* **Description**: Génère un nouveau mot de passe sécurisé pour un candidat
* **Paramètres**:
  + numElecteur: Numéro d'électeur du candidat
* **Réponse**: Détails du nouveau code de sécurité

**Gestion des parrains**

**1. Enregistrer un parrain**

* **Endpoint**: POST /parrain/register
* **Description**: Enregistre un nouvel électeur en tant que parrain
* **Format de la requête**:
* { "numElecteur": "123456789", "email": "parrain@example.com", "telephone": "+123456789"}
* **Réponse**: Détails du parrain avec ses informations d'électeur

**2. Demander un code d'authentification**

* **Endpoint**: POST /parrain/request-code
* **Description**: Demande un nouveau code d'authentification
* **Paramètres**:
  + num\_electeur: Numéro d'électeur du parrain
* **Réponse**:
* { "message": "Nouveau code envoyé"}

**3. Vérifier un code d'authentification**

* **Endpoint**: POST /parrain/verify-code
* **Description**: Vérifie un code d'authentification
* **Paramètres**:
  + num\_electeur: Numéro d'électeur du parrain
  + code: Code d'authentification à vérifier
* **Réponse**:
* { "valid": true}

**4. Obtenir un parrain**

* **Endpoint**: GET /parrain/{num\_electeur}
* **Description**: Récupère un parrain par son numéro
* **Paramètres**:
  + num\_electeur: Numéro d'électeur du parrain
* **Réponse**: Détails du parrain avec ses informations d'électeur

**5. Obtenir tous les parrains**

* **Endpoint**: GET /parrain/
* **Description**: Liste tous les parrains
* **Paramètres**:
  + skip (optionnel): Nombre d'éléments à sauter (pagination)
  + limit (optionnel): Nombre maximum d'éléments à retourner
* **Réponse**: Liste des parrains avec leurs informations d'électeur

**6. Supprimer un parrain**

* **Endpoint**: DELETE /parrain/{num\_electeur}
* **Description**: Supprime un parrain
* **Paramètres**:
  + num\_electeur: Numéro d'électeur du parrain
* **Réponse**: 204 No Content en cas de succès

**Gestion des parrainages**

**1. Créer un parrainage**

* **Endpoint**: POST /
* **Description**: Un électeur parraine un candidat
* **Format de la requête**:
* { "numElecteur": "123456789", "numCandidat": "987654321", "codeValidation": "ABC123"}
* **Réponse**:
* { "message": "Parrainage enregistré avec succès."}

**2. Obtenir le nombre de parrainages d'un candidat**

* **Endpoint**: GET /candidat/{numCandidat}
* **Description**: Récupère uniquement le nombre total de parrainages d'un candidat
* **Paramètres**:
  + numCandidat: Numéro d'électeur du candidat
* **Réponse**:
* { "numCandidat": "987654321", "nbrParrainages": 42}

**Gestion des membres DGE**

**1. Créer un membre**

* **Endpoint**: POST /
* **Description**: Crée un nouveau membre DGE
* **Format de la requête**:
* { "email": "membre@example.com", "telephone": "+123456789", "password": "motdepasse123", "role": "user"}
* **Réponse**: Détails du membre créé

**2. Obtenir tous les membres**

* **Endpoint**: GET /
* **Description**: Retourne la liste de tous les membres DGE
* **Réponse**: Liste des membres DGE

**3. Vérifier les identifiants d'un membre**

* **Endpoint**: POST /login
* **Description**: Vérifie les identifiants d'un membre
* **Paramètres**:
  + email: Email du membre
  + password: Mot de passe du membre
* **Réponse**:
* { "message": "Connexion réussie"}
* **Codes d'erreur**:
  + 401 Unauthorized: Email ou mot de passe incorrect