

Documentation API - Application de Sondage Électoral 2025

Informations générales

URL de base: `https://api-electoralpoll.adjemincloud.com`

Version: 1.0

Format: JSON

Encodage: UTF-8

Date de mise à jour: 17 octobre 2025

Table des matières

1. Authentication
 2. Endpoints publics
 - GET /api/candidates
 - GET /api/candidates/{id}
 - POST /api/check-eligibility
 - POST /api/votes
 3. Rate Limiting
 4. Codes d'erreur
 5. Exemples de code
 6. Sécurité
-

Authentication

Les endpoints publics (consultation de candidats, soumission de vote) **ne nécessitent pas d'authentification**.

Les endpoints d'administration nécessitent un token Bearer JWT (non documentés ici, réservés au

backoffice).

Endpoints publics

GET /api/candidates

Récupère la liste complète des candidats actifs disponibles pour le vote.

URL

```
GET https://api-electoralpoll.adjemincloud.com/api/candidates
```

Headers

```
Accept: application/json
```

Paramètres

Aucun

Réponse succès (200 OK)

```
{
  "success": true,
  "data": [
    {
      "id": 1,
      "name": "Jean Dupont",
      "party": "Parti Démocratique",
      "photo_url": "https://api-electoralpoll.adjemincloud.com/storage/
candidates/jean-dupont.jpg",
      "order_number": 1
    },
    {
```

```
    "id": 2,
    "name": "Marie Martin",
    "party": "Parti Socialiste",
    "photo_url": "https://api-electoralpoll.adjeminccloud.com/storage/
candidates/marie-martin.jpg",
    "order_number": 2
  }
],
"generated_at": "2025-10-17T10:30:00.000000Z"
}
```

Champs de réponse

Champ	Type	Description
success	boolean	Indique si la requête a réussi
data	array	Liste des candidats
data[].id	integer	Identifiant unique du candidat
data[].name	string	Nom complet du candidat
data[].party	string	Parti politique du candidat
data[].photo_url	string	URL de la photo du candidat
data[].order_number	integer	Ordre d'affichage (tri)
generated_at	string (ISO 8601)	Date/heure de génération

Notes importantes

- Les résultats sont **mis en cache pendant 1 heure** (Redis)
- Seuls les candidats avec `is_active = true` sont retournés
- Les candidats sont triés par `order_number` (ordre croissant)

Exemple cURL

```
curl -X GET \
```

```
'https://api-electoralpoll.adjemincloud.com/api/candidates' \
-H 'Accept: application/json'
```

GET /api/candidates/{id}

Récupère les détails d'un candidat spécifique par son ID.

URL

```
GET https://api-electoralpoll.adjemincloud.com/api/candidates/{id}
```

Headers

```
Accept: application/json
```

Paramètres URL

Paramètre	Type	Requis	Description
id	integer	Oui	Identifiant unique du candidat

Réponse succès (200 OK)

```
{
  "success": true,
  "data": {
    "id": 1,
    "name": "Jean Dupont",
    "party": "Parti Démocratique",
    "description": "Candidat avec 20 ans d'expérience en politique...",
    "photo_url": "https://api-electoralpoll.adjemincloud.com/storage/"
```

```
candidates/jean-dupont.jpg",
  "order_number": 1
}
```

Réponse erreur - Candidat non trouvé (404 Not Found)

```
{
  "success": false,
  "message": "Candidat non trouvé"
}
```

Exemple cURL

```
curl -X GET \
  'https://api-electoralpoll.adjeminccloud.com/api/candidates/1' \
  -H 'Accept: application/json'
```

POST /api/check-eligibility

Vérifie si un numéro de carte d'électeur a déjà été utilisé pour voter.

 **Rate Limit: 10 requêtes par minute par IP**

URL

```
POST https://api-electoralpoll.adjeminccloud.com/api/check-eligibility
```

Headers

```
Content-Type: application/json
```

Accept: application/json

Corps de la requête

```
{
  "card_number": "ABC123456789"
}
```

Paramètres

Champ	Type	Requis	Description	Validation
card_number	string	Oui	Numéro de carte d'électeur	Alphanumérique, 6-20 caractères

Réponse succès - Éligible (200 OK)

```
{
  "success": true,
  "eligible": true,
  "message": "Ce numéro peut voter"
}
```

Réponse succès - Déjà voté (200 OK)

```
{
  "success": true,
  "eligible": false,
  "message": "Ce numéro de carte a déjà voté"
}
```

Réponse erreur - Validation (422 Unprocessable Entity)

```
{
  "success": false,
  "message": "Données invalides",
  "errors": {
    "card_number": [
      "Le champ card_number est obligatoire.",
      "Le champ card_number doit contenir entre 6 et 20 caractères."
    ]
  }
}
```

Exemple cURL

```
curl -X POST \
  'https://api-electoralpoll.adjeminccloud.com/api/check-eligibility' \
  -H 'Content-Type: application/json' \
  -H 'Accept: application/json' \
  -d '{
    "card_number": "ABC123456789"
  }'
```

Notes importantes

- Cette vérification ne garantit **pas** que le vote sera accepté (vérification temps réel lors de la soumission finale)
- Utilisez cet endpoint pour un feedback utilisateur immédiat
- Ne faites **pas** de polling répétitif sur cet endpoint (respectez le rate limit)

POST /api/votes

 **ENDPOINT PRINCIPAL** - Soumet un vote complet avec carte d'électeur et choix de candidat.

- ⚠ **Rate Limit: 10 requêtes par minute par IP**
- ⚠ **Disponible uniquement pendant la période de sondage active**

URL

POST <https://api-ectoralpoll.adjemincloud.com/api/votes>

Headers

Content-Type: multipart/form-data
Accept: application/json

Corps de la requête (multipart/form-data)

Champ	Type	Requis	Description	Validation
voter_card_number	string	Oui	Numéro de carte d'électeur	Alphanumérique, 6-20 caractères, unique
voter_card_image	file	Oui	Photo de la carte d'électeur	JPG/PNG/WEBP, max 5 MB, min 800x600px
candidate_id	integer	Oui	ID du candidat choisi	Doit exister et être actif

Réponse succès (201 Created)

```
{
  "success": true,
  "message": "Vote enregistré avec succès",
  "vote_id": "9c8e7d6f-5a4b-3c2d-1e0f-9a8b7c6d5e4f",
  "voted_at": "2025-10-17T10:35:22.000000Z"
}
```


Champs de réponse

Champ	Type	Description
success	boolean	Toujours <code>true</code> en cas de succès
message	string	Message de confirmation
vote_id	string (UUID)	Identifiant unique du vote enregistré
voted_at	string (ISO 8601)	Date/heure exacte du vote

Réponses d'erreur

Erreur validation (422 Unprocessable Entity)

```
{
  "success": false,
  "message": "Données invalides",
  "errors": {
    "voter_card_number": [
      "Le numéro de carte a déjà été utilisé pour voter."
    ],
    "voter_card_image": [
      "L'image doit être au format JPG, PNG ou WEBP.",
      "L'image ne doit pas dépasser 5 MB."
    ],
    "candidate_id": [
      "Le candidat sélectionné n'existe pas."
    ]
  }
}
```

Erreur - Carte déjà utilisée (422 Unprocessable Entity)

```
{
  "success": false,
  "message": "Données invalides",
  "errors": {
    "voter_card_number": [
      "Ce numéro de carte a déjà été utilisé pour voter."
    ]
  }
}
```

Erreur - Période de sondage inactive (403 Forbidden)

```
{
  "success": false,
  "message": "Le sondage n'est pas actuellement actif."
}
```

Erreur - IP bloquée pour fraude (403 Forbidden)

```
{
  "success": false,
  "message": "Accès temporairement bloqué"
}
```

Erreur - Rate limit dépassé (429 Too Many Requests)

```
{
  "success": false,
  "message": "Trop de requêtes. Veuillez réessayer dans quelques instants.",
  "retry_after": 60
}
```

Exemple cURL

```
curl -X POST \
  'https://api-electoralpoll.adjeminccloud.com/api/votes' \
  -H 'Accept: application/json' \
  -F 'voter_card_number=ABC123456789' \
  -F 'voter_card_image=@/path/to/voter_card.jpg' \
  -F 'candidate_id=1'
```

Exemple JavaScript (Fetch API)

```
const formData = new FormData();
formData.append('voter_card_number', 'ABC123456789');
formData.append('voter_card_image', fileInput.files[0]);
formData.append('candidate_id', 1);

fetch('https://api-electoralpoll.adjeminccloud.com/api/votes', {
  method: 'POST',
  headers: {
    'Accept': 'application/json'
  },
  body: formData
})
.then(response => response.json())
.then(data => {
  if (data.success) {
    console.log('Vote enregistré:', data.vote_id);
  } else {
    console.error('Erreur:', data.message, data.errors);
  }
})
.catch(error => console.error('Erreur réseau:', error));
```

Notes critiques de sécurité

1. **Unicité stricte:** Un numéro de carte ne peut voter qu'**une seule fois**
2. **Période active:** L'endpoint est bloqué en dehors de la période de sondage configurée
3. **Détection de fraude:**

- Détection des IP multiples
 - Détection de votes trop rapides (< 30 sec entre votes depuis la même IP)
 - Blocage automatique des IP suspectes
4. **Logs d'audit:** Chaque soumission est tracée avec IP, user-agent, et browser fingerprint
 5. **Cryptage:** Les numéros de carte et IPs sont cryptés en base de données

Workflow recommandé frontend

1. Utilisateur remplit le formulaire
↓
2. [Optionnel] Appeler POST /api/check-eligibility pour feedback immédiat
↓
3. Utilisateur confirme son choix
↓
4. Appeler POST /api/votes avec toutes les données
↓
5. Afficher confirmation ou erreurs

Rate Limiting

L'API applique des limites strictes pour protéger contre les abus.

Endpoint	Limite	Fenêtre	Clé
GET /api/candidates	60 requêtes	1 minute	IP
GET /api/candidates/{id}	60 requêtes	1 minute	IP
POST /api/check-eligibility	10 requêtes	1 minute	IP
POST /api/votes	10 requêtes	1 minute	IP

Headers de réponse rate limit

```
X-RateLimit-Limit: 10
X-RateLimit-Remaining: 7
X-RateLimit-Reset: 1697536200
```

Erreur 429 - Too Many Requests

```
{
  "success": false,
  "message": "Trop de requêtes. Veuillez réessayer dans quelques instants.",
  "retry_after": 60
}
```

Bonnes pratiques:

- Respectez toujours le header `Retry-After`
- Implémentez un backoff exponentiel côté client
- Évitez le polling agressif
- Mettez en cache les données candidates côté frontend

Codes d'erreur

Codes HTTP standards

Code	Signification	Description
200	OK	Requête réussie
201	Created	Ressource créée avec succès (vote enregistré)
400	Bad Request	Syntaxe de requête incorrecte
403	Forbidden	Accès refusé (IP bloquée, période inactive)
404	Not Found	Ressource non trouvée

422	Unprocessable Entity	Validation échouée
429	Too Many Requests	Rate limit dépassé
500	Internal Server Error	Erreur serveur interne
503	Service Unavailable	Service temporairement indisponible

Structure standard des erreurs

```
{
  "success": false,
  "message": "Description générale de l'erreur",
  "errors": {
    "field_name": [
      "Message d'erreur détaillé 1",
      "Message d'erreur détaillé 2"
    ]
  }
}
```

Exemples de code

JavaScript (Axios)

```
import axios from 'axios';

const API_BASE_URL = 'https://api-electoralpoll.adjemincloud.com';

// Récupérer les candidats
async function getCandidates() {
  try {
    const response = await axios.get(`${API_BASE_URL}/api/candidates`);
    return response.data.data;
  }
}
```

```

    } catch (error) {
      console.error('Erreur:', error.response?.data || error.message);
      throw error;
    }
  }
}

```

// Vérifier éligibilité

```

async function checkEligibility(cardNumber) {
  try {
    const response = await axios.post(
      `${API_BASE_URL}/api/check-eligibility`,
      { card_number: cardNumber }
    );
    return response.data.eligible;
  } catch (error) {
    console.error('Erreur:', error.response?.data || error.message);
    return false;
  }
}

```

// Soumettre un vote

```

async function submitVote(cardNumber, cardImage, candidateId) {
  const formData = new FormData();
  formData.append('voter_card_number', cardNumber);
  formData.append('voter_card_image', cardImage);
  formData.append('candidate_id', candidateId);

  try {
    const response = await axios.post(
      `${API_BASE_URL}/api/votes`,
      formData,
      {
        headers: {
          'Content-Type': 'multipart/form-data'
        }
      }
    );
    return response.data;
  } catch (error) {
    if (error.response?.status === 422) {
      // Erreurs de validation
      throw new Error(
        Object.values(error.response.data.errors).flat().join(', ')
      );
    }
  }
}

```

```

    );
  }
  throw error;
}
}

```

TypeScript (avec types)

```

interface Candidate {
  id: number;
  name: string;
  party: string;
  photo_url: string;
  order_number: number;
}

interface ApiResponse<T> {
  success: boolean;
  data?: T;
  message?: string;
  errors?: Record<string, string[]>;
}

interface VoteSubmitResponse {
  success: boolean;
  message: string;
  vote_id: string;
  voted_at: string;
}

class ElectoralPollAPI {
  private baseUrl = 'https://api-electoralpoll.adjemincloud.com';

  async getCandidates(): Promise<Candidate[]> {
    const response = await fetch(`${this.baseUrl}/api/candidates`, {
      headers: { 'Accept': 'application/json' }
    });

    if (!response.ok) {
      throw new Error(`HTTP ${response.status}`);
    }
  }
}

```



```

    }

    const data: ApiResponse<Candidate[]> = await response.json();
    return data.data || [];
}

async checkEligibility(cardNumber: string): Promise<boolean> {
    const response = await fetch(`${this.baseUrl}/api/check-eligibility`, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Accept': 'application/json'
        },
        body: JSON.stringify({ card_number: cardNumber })
    });

    const data: ApiResponse<{ eligible: boolean }> = await response.json();
    return data.data?.eligible || false;
}

async submitVote(
    cardNumber: string,
    cardImage: File,
    candidateId: number
): Promise<VoteSubmitResponse> {
    const formData = new FormData();
    formData.append('voter_card_number', cardNumber);
    formData.append('voter_card_image', cardImage);
    formData.append('candidate_id', candidateId.toString());

    const response = await fetch(`${this.baseUrl}/api/votes`, {
        method: 'POST',
        headers: { 'Accept': 'application/json' },
        body: formData
    });

    if (!response.ok) {
        const error = await response.json();
        throw new Error(error.message || 'Erreur lors du vote');
    }

    return response.json();
}

```

```

}

// Utilisation
const api = new ElectoralPollAPI();

async function handleVoteSubmission() {
  try {
    const candidates = await api.getCandidates();
    console.log('Candidats:', candidates);

    const isEligible = await api.checkEligibility('ABC123456789');
    if (!isEligible) {
      alert('Ce numéro a déjà voté');
      return;
    }

    const result = await api.submitVote(
      'ABC123456789',
      imageFile,
      selectedCandidateId
    );

    console.log('Vote enregistré:', result.vote_id);
    alert(result.message);
  } catch (error) {
    console.error('Erreur:', error);
    alert('Une erreur est survenue');
  }
}

```

Angular Service

```

import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError, map } from 'rxjs/operators';

interface Candidate {
  id: number;
  name: string;
}

```

```

    party: string;
    photo_url: string;
    order_number: number;
}

interface ApiResponse<T> {
    success: boolean;
    data?: T;
    message?: string;
    errors?: Record<string, string[]>;
}

@Injectables({
    providedIn: 'root'
})
export class ElectoralPollService {
    private apiUrl = 'https://api-electoralpoll.adjemincloud.com/api';

    constructor(private http: HttpClient) {}

    getCandidates(): Observable<Candidate[]> {
        return this.http.get<ApiResponse<Candidate[]>>(`${this.apiUrl}/
candidates`)
            .pipe(
                map(response => response.data || []),
                catchError(this.handleError)
            );
    }

    checkEligibility(cardNumber: string): Observable<boolean> {
        return this.http.post<ApiResponse<{ eligible: boolean }>>(
            `${this.apiUrl}/check-eligibility`,
            { card_number: cardNumber }
        ).pipe(
            map(response => response.data?.eligible || false),
            catchError(this.handleError)
        );
    }

    submitVote(
        cardNumber: string,
        cardImage: File,
        candidateId: number
    ) {

```

```

): Observable<any> {
  const formData = new FormData();
  formData.append('voter_card_number', cardNumber);
  formData.append('voter_card_image', cardImage);
  formData.append('candidate_id', candidateId.toString());

  return this.http.post(`${this.apiUrl}/votes`, formData)
    .pipe(catchError(this.handleError));
}

private handleError(error: any) {
  console.error('API Error:', error);
  return throwError(() => new Error(
    error.error?.message || 'Une erreur est survenue'
  ));
}
}

```

Sécurité

Protection des données

1. **Cryptage en transit:** Toutes les communications se font via **HTTPS uniquement**
2. **Cryptage au repos:**
 - Numéros de carte cryptés en base de données
 - Adresses IP cryptées
3. **CORS:** Configuré pour accepter uniquement les domaines autorisés
4. **CSRF Protection:** Tokens CSRF pour les requêtes sensibles

Détection de fraude










Le système détecte automatiquement:

- Votes multiples depuis la même IP
- Patterns de vote suspects (trop rapide, même fingerprint)
- Tentatives de contournement du rate limit

- Images de carte invalides ou de mauvaise qualité

Conséquences: Blocage temporaire ou permanent de l'IP

Bonnes pratiques pour les développeurs

1.  **Toujours valider côté client** avant d'envoyer au serveur
2.  **Afficher des messages d'erreur clairs** à l'utilisateur
3.  **Implémenter un système de retry** avec backoff pour les erreurs réseau
4.  **Ne jamais stocker les numéros de carte** côté frontend (privacy)
5.  **Utiliser HTTPS** pour toutes les requêtes
6.  **Gérer les timeouts** (timeout recommandé: 30 secondes)
7.  **Logger les erreurs** pour faciliter le debugging
8.  **Ne jamais** inclure de credentials dans le code source
9.  **Ne jamais** contourner les rate limits

Validation des images côté client

Avant d'envoyer au serveur, validez:

```
function validateImageFile(file) {
  const validTypes = ['image/jpeg', 'image/png', 'image/webp'];
  const maxSize = 5 * 1024 * 1024; // 5 MB

  if (!validTypes.includes(file.type)) {
    throw new Error('Format d\'image invalide. Utilisez JPG, PNG ou WEBP.');
```

```
  }

  if (file.size > maxSize) {
    throw new Error('L\'image ne doit pas dépasser 5 MB.');
```

```
  }

  return true;
}

// Vérifier la résolution
function checkImageResolution(file) {
  return new Promise((resolve, reject) => {
```

```
const img = new Image();
img.onload = () => {
  if (img.width < 800 || img.height < 600) {
    reject(new Error('Résolution minimale: 800x600 pixels'));
  } else {
    resolve(true);
  }
};
img.src = URL.createObjectURL(file);
});
}
```

Support et Contact

Pour toute question technique concernant l'API:

- **Email support:** angebagui@adjemin.com

Changelog

Version	Date	Changements
1.0	17/10/2025	Version initiale de l'API

© 2025 Electoral Poll Application - Tous droits réservés

Document généré le: 17 octobre 2025

Version: 1.0