Diénaba Kande

API et de la logique métier, ce qui signifie qu'on doit développer les endpoints pour l'importation des fichiers CSV, l'enregistrement des candidats et la gestion des parrainages.

Mise en place du projet backend

1. Créer un projet Node.js avec Express

mkdir backend

cd backend

npm init -y

npm install express multer mariadb dotenv cors nodemon

Configurer le serveur Express (server.js)

```
//Configurer le serveur Express
require('dotenv').config();
const express = require('express');
const cors = require('cors');

const app = express();
app.use(cors());
app.use(express.json());

app.get('/', (req, res) => {
    res.send('API Gestion Parrainages');
});

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Serveur lancé sur le port ${PORT}`));
```

2. Créer les APIs principales

(1) Endpoint pour l'upload des fichiers CSV

Utilisation de multer pour gérer le fichier en entrée.

Vérification du format et stockage temporaire.

```
//Créer les APIs principales
//(1) Endpoint pour l'upload des fichiers CSV
const multer = require('multer');
const upload = multer({ dest: 'uploads/' });

app.post('/api/upload', upload.single('file'), (req, res) => {
    if (!req.file) return res.status(400).json({ message: 'Aucun fichier envoyé' });

    res.json({ message: 'Fichier reçu', fileName: req.file.filename });
});

const mariadb = require('mariadb');
const pool = mariadb.createPool({
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'gestionparrainage',
    connectionLimit: 5
});
```

(2) Enregistrement des candidats

Stocker les candidats dans la base de données via un modèle candidats.

```
//(2) Enregistrement des candidats
//Stocker les candidats dans la base de données via un modèle candidats.

app.post('/api/candidats', async (req, res) => {
    const { cin, nom, prenom, date_naissance, email, telephone, parti_politique } = r

    try {
        const conn = await pool.getConnection();
        await conn.query(
            'INSERT INTO candidats (cin, nom, prenom, date_naissance, email, telephon
            [cin, nom, prenom, date_naissance, email, telephone, parti_politique]
        );
        conn.release();
        res.json({ message: 'Candidat enregistré avec succès' });
    } catch (error) {
        res.status(500).json({ error: error.message });
    }
});
```

(3) Gestion des parrainages

Vérifier si l'électeur a déjà parrainé un candidat.

Enregistrer le parrainage dans la base.

```
app.post('/api/parrainages', async (req, res) => {
    const { numero_electeur, cin_candidat } = req.body;

    try {
        const conn = await pool.getConnection();

        // Vérifier si l'électeur a déjà parrainé
        const [exists] = await conn.query('SELECT * FROM parrainages WHERE numero_ele
        if (exists) {
            conn.release();
            return res.status(400).json({ message: 'L'électeur a déjà parrainé un can
        }

        // Insérer le parrainage
        await conn.query('INSERT INTO parrainages (numero_electeur, cin_candidat) VAL
        conn.release();
        res.json({ message: 'Parrainage enregistré avec succès' });
    } catch (error) {
        res.status(500).json({ error: error.message });
    }
});
```


3. Gestion de la sécurité

Mise en place des règles CORS

Ajoutons dans server.js :

app.use(cors({ origin: 'http://localhost:3000', credentials: true }));

Chiffrement des données sensibles (ex: codes de vérification)

Installation :

npm install crypto-js

Génération et chiffrement du code de vérification :

```
const CryptoJS = require('crypto-js');

function generateCode() {
    return Math.floor(100000 + Math.random() * 900000).toString(); // 6 chiffres
}

function encryptCode(code) {
    return CryptoJS.AES.encrypt(code, process.env.SECRET_KEY).toString();
}
```

Maintenant que nous avons mis en place les **endpoints principaux du backend**, nous allons :

1. **Gérer l'authentification** des électeurs et candidats (avec vérification par code).
2. **Finaliser la gestion des fichiers CSV** pour importer les électeurs.

3. **Configurer la communication avec le frontend React**.
4. **Faire des tests unitaires et optimiser la sécurité**.

1 Authentification des électeurs et candidats

L'idée est de permettre aux électeurs et candidats de s'authentifier via un **code de vérification envoyé par email/SMS**.

➢ Création de l'endpoint d'envoi de code

Nous allons générer un **code aléatoire**, l'envoyer par email/SMS et le stocker temporairement dans la base de données.

- **Installer le module nodemailer (pour envoyer des emails)**

npm install nodemailer

- **Créer un fichier auth.js pour gérer l'authentification**

Gestion avancée des fichiers CSV

Nous devons :

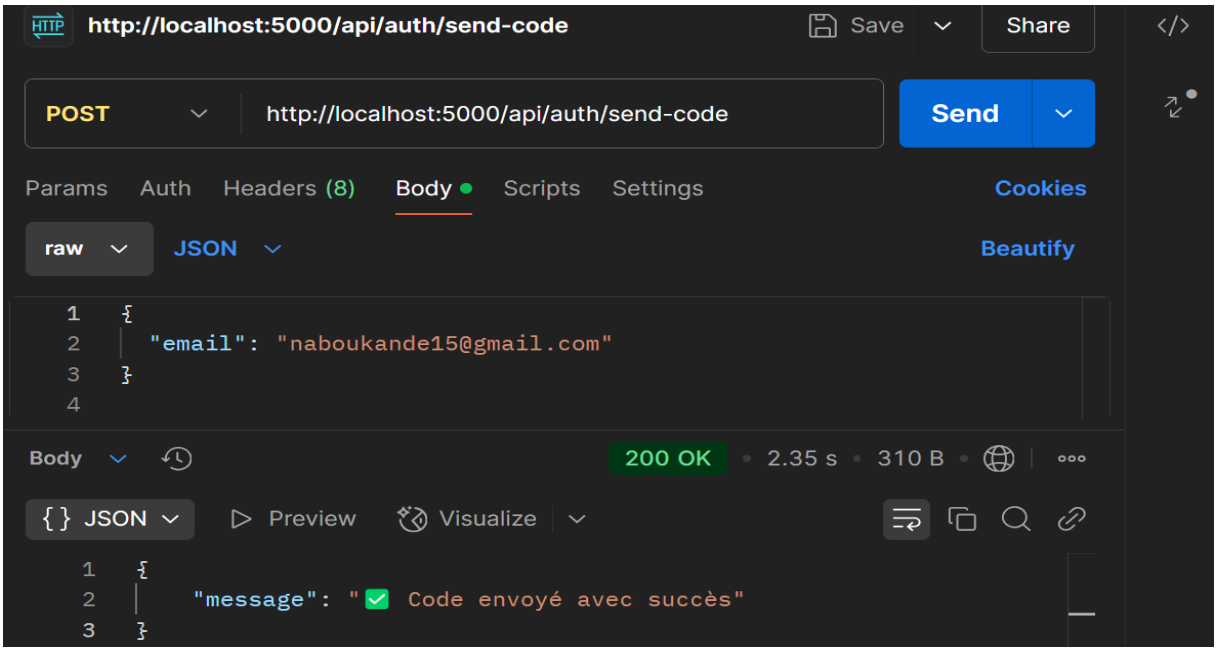
- **Vérifier le format** du fichier avant import.
- **Stocker les électeurs temporairement**, puis valider les données avant importation finale.

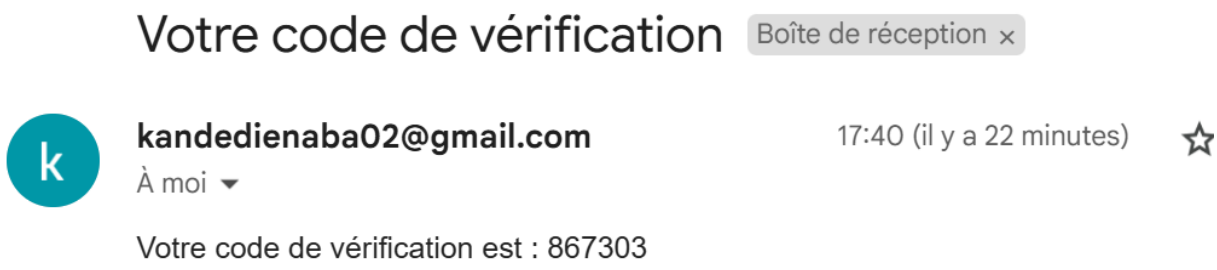
➢ Vérification et lecture du fichier CSV

1. **Installer csv-parser**

npm install csv-parser fs

Testons avec POSTMAN





3 Authentification complète avec JWT (JSON Web Token)

Maintenant qu'on peut envoyer et vérifier un code par email, on va gérer les sessions utilisateurs avec JWT.

📌 1. Explication du fonctionnement

🔹 Un utilisateur entre son email et reçoit un code.

🔹 Il vérifie ce code via /api/auth/verify-code.

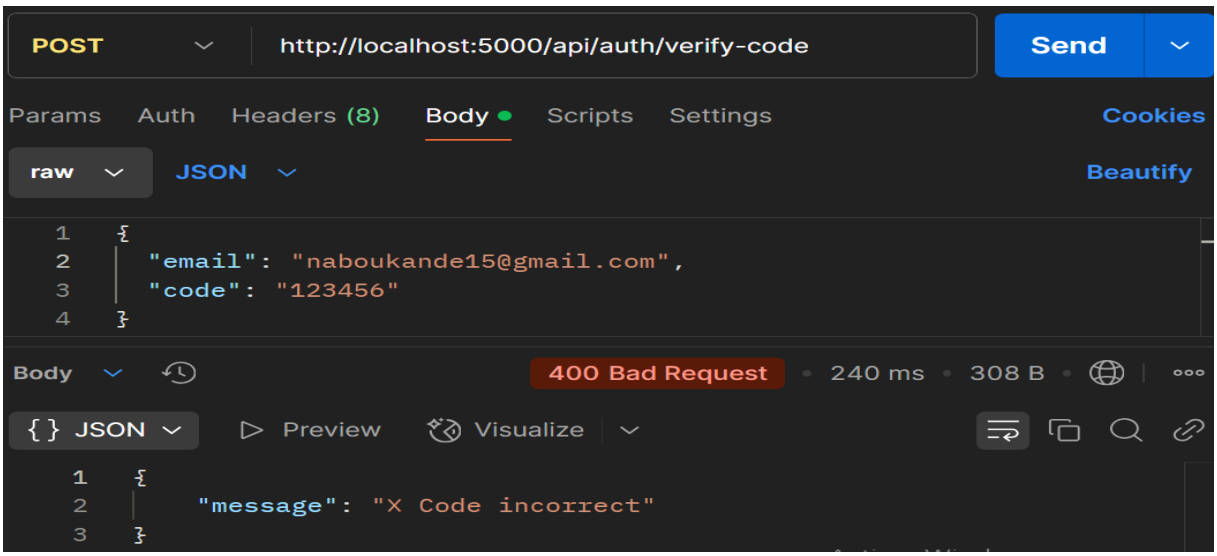
🔹 S'il est correct, on génère un Token JWT.

🔹 Ce token permet d'accéder aux routes sécurisées (ex: gérer les parrainages, voir ses infos, etc.).

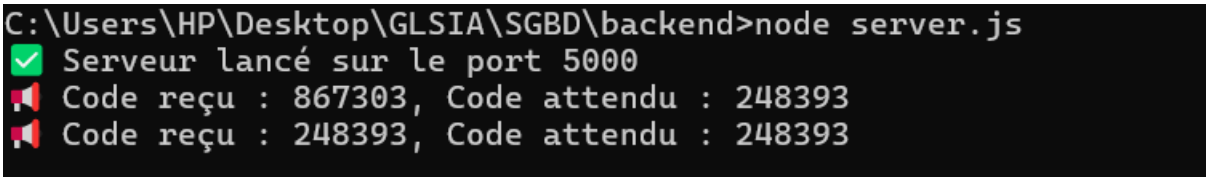
✅ 2. Installer JWT dans le backend

Dans le dossier backend/, installons jsonwebtoken :

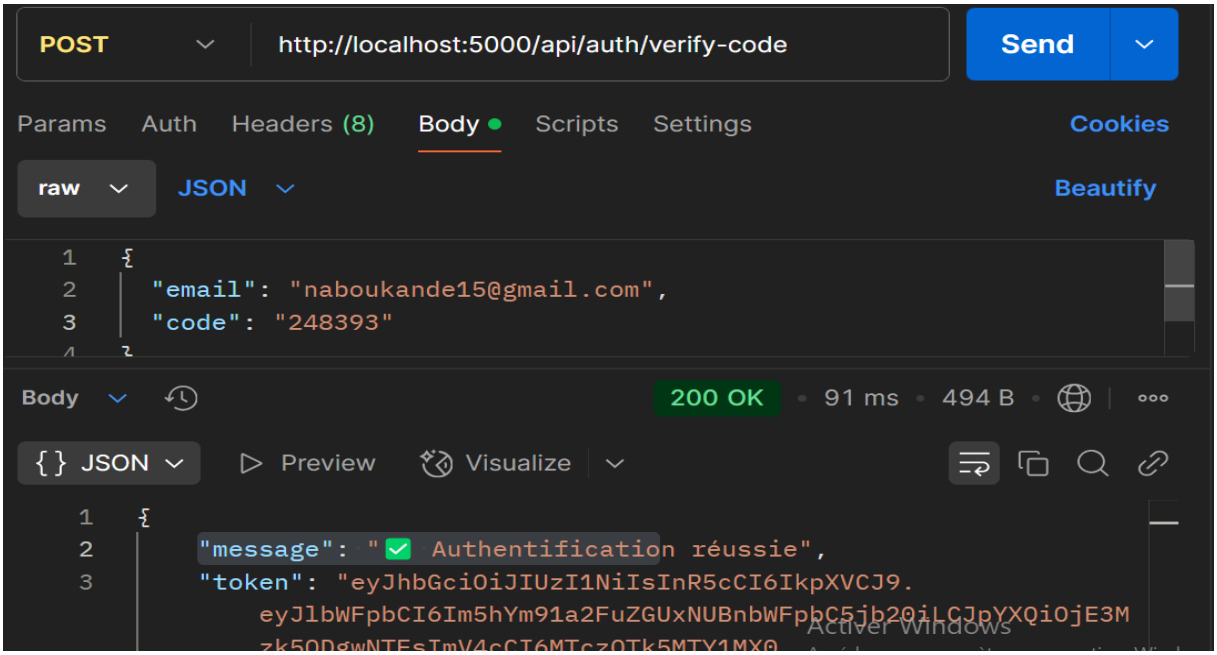
Code incorrect



Verifions le code :



Le code est 248393



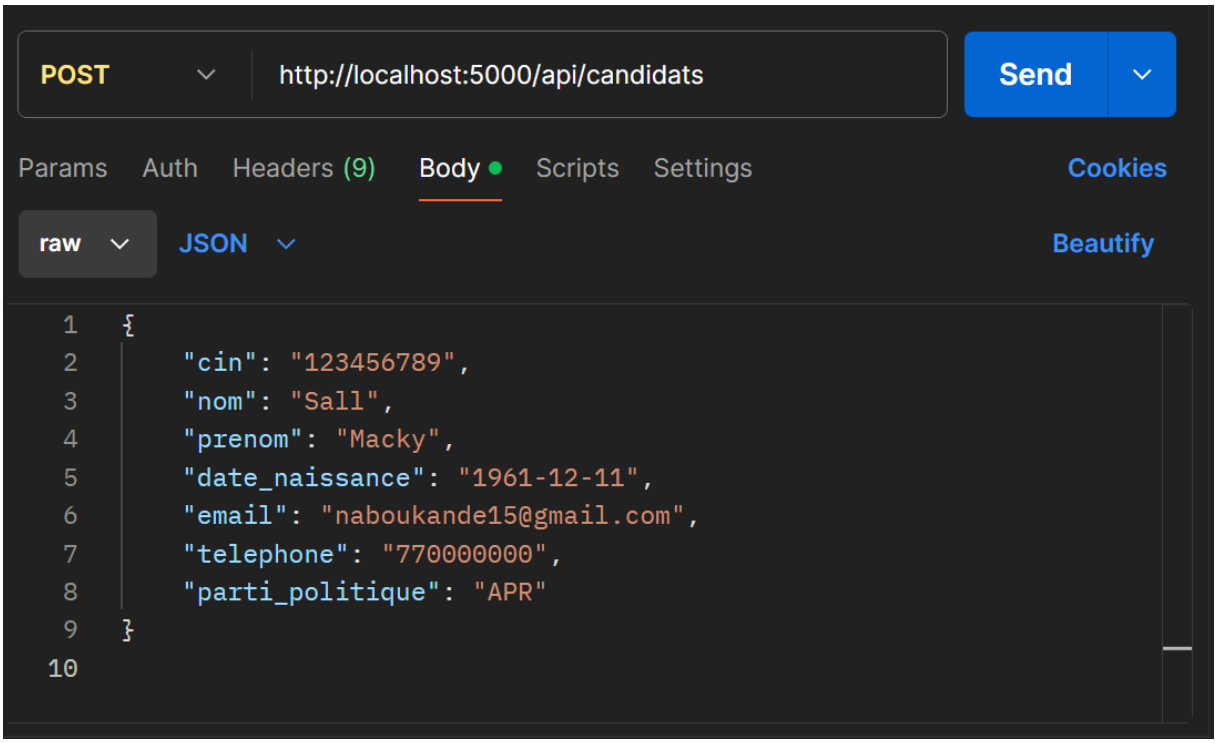
Le contenue de la table codes_verification

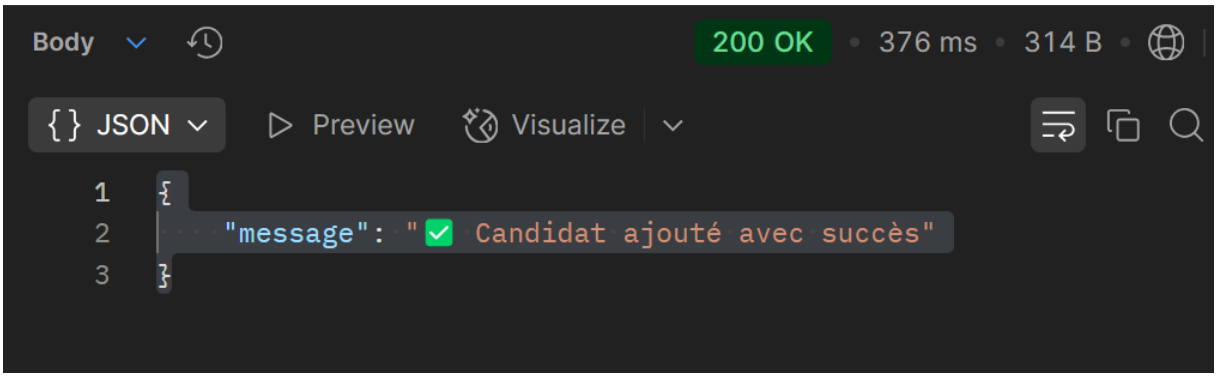
```
MariaDB [(none)]> USE gestionparrainage;
Database changed
MariaDB [gestionparrainage]> SELECT * FROM codes_verification;
+----+-----------------------+-----------------------------------------------+---------------------+
| id | email                 | code                                          | created_at          |
+----+-----------------------+-----------------------------------------------+---------------------+
|  1 | test@example.com      | U2FsdGVkX19TA5kZcDnpQz5lv0msMq1muTyzHnb9ah8= | 2025-02-18 22:51:37 |
|  2 | naboukande15@gmail.com | U2FsdGVkX1+ijSLcmX7rgdEUGYbeKjmlNqMV2gMnClk= | 2025-02-18 23:01:10 |
|  3 | naboukande15@gmail.com | U2FsdGVkX1+PlPw3agMNoOF/iTFmzjVczTo6oM7QWPg= | 2025-02-18 23:01:42 |
|  4 | naboukande15@gmail.com | U2FsdGVkX19Byc0TuoNsrYJp+Yg7aprW6AS04N0ZfIU= | 2025-02-19 17:40:28 |
+----+-----------------------+-----------------------------------------------+---------------------+
4 rows in set (0.001 sec)
```

4 Prochaine étape : Gestion des candidats

Un candidat peut :

- S’inscrire (POST /api/candidats)
- Voir la liste des candidats (GET /api/candidats)
- Modifier ses infos (PUT /api/candidats/:cin)
- Supprimer un candidat (DELETE /api/candidats/:cin)





On voit que le candidat est correctement inserer

🚀 Gestion des électeurs et des parrainages

Maintenant que l'API des candidats fonctionne, on va gérer les électeurs et leurs parrainages.

Un électeur pourra :

✅ S'inscrire (POST /api/electeurs)

✅ Voir la liste des électeurs (GET /api/electeurs)

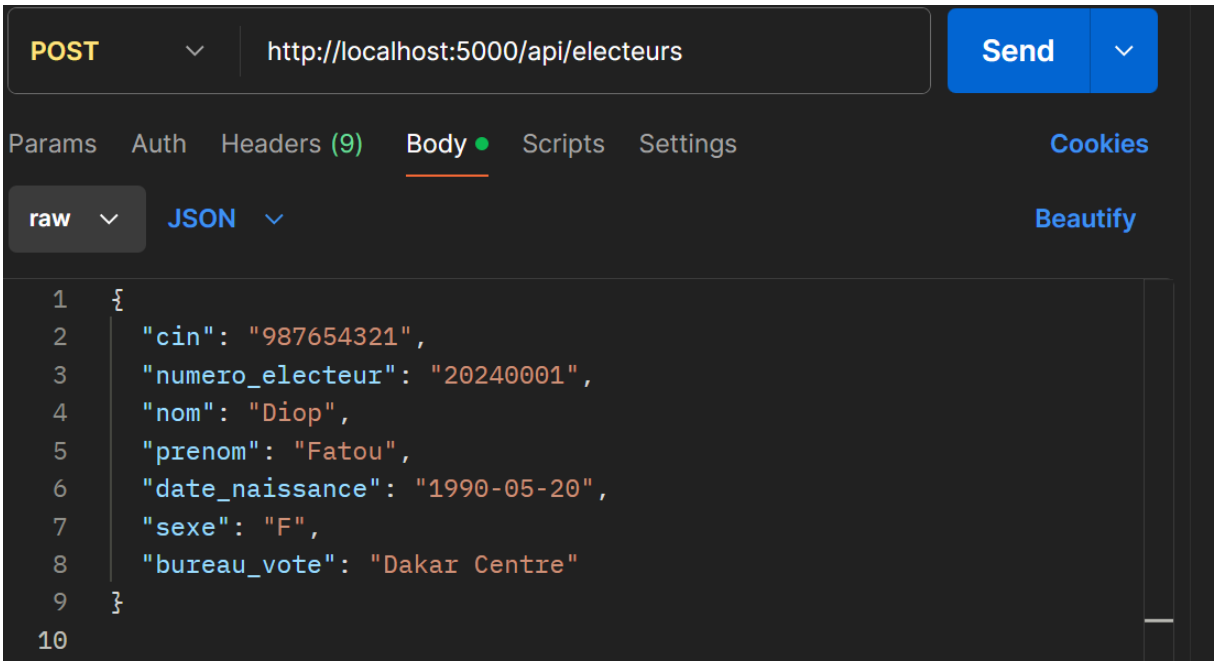
✅ Parrainer un candidat (POST /api/parrainages)

✅ Voir les parrainages (GET /api/parrainages)

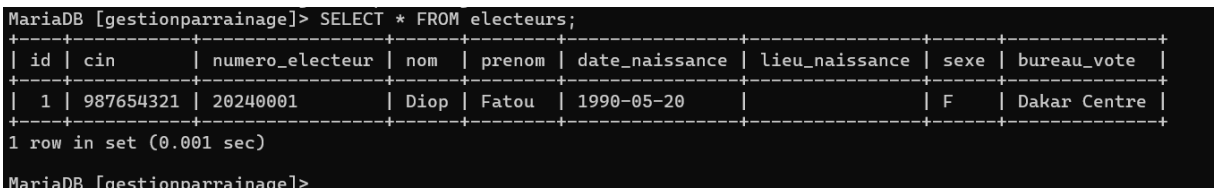
Ajouter un électeur

Méthode : POST

URL : http://localhost:5000/api/electeurs



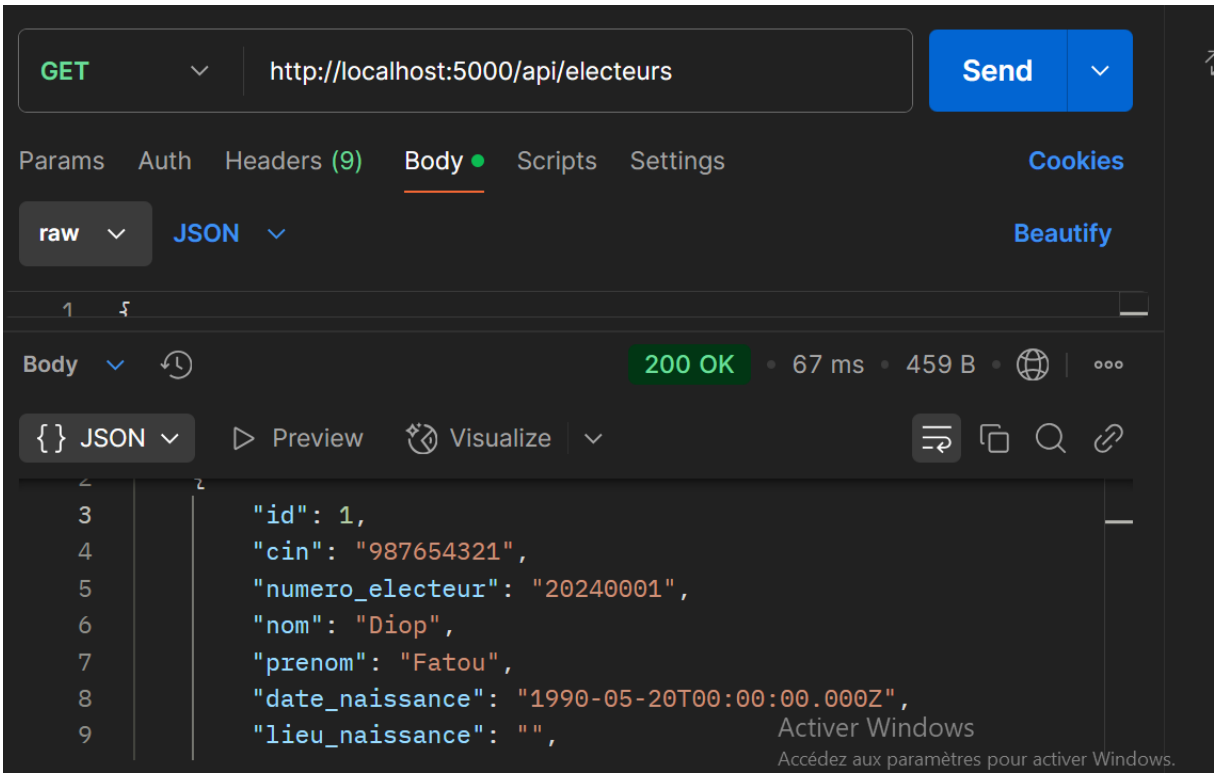
On voit qu'il est bien enrigistrer dans la base



Voir tous les électeurs

Méthode : GET

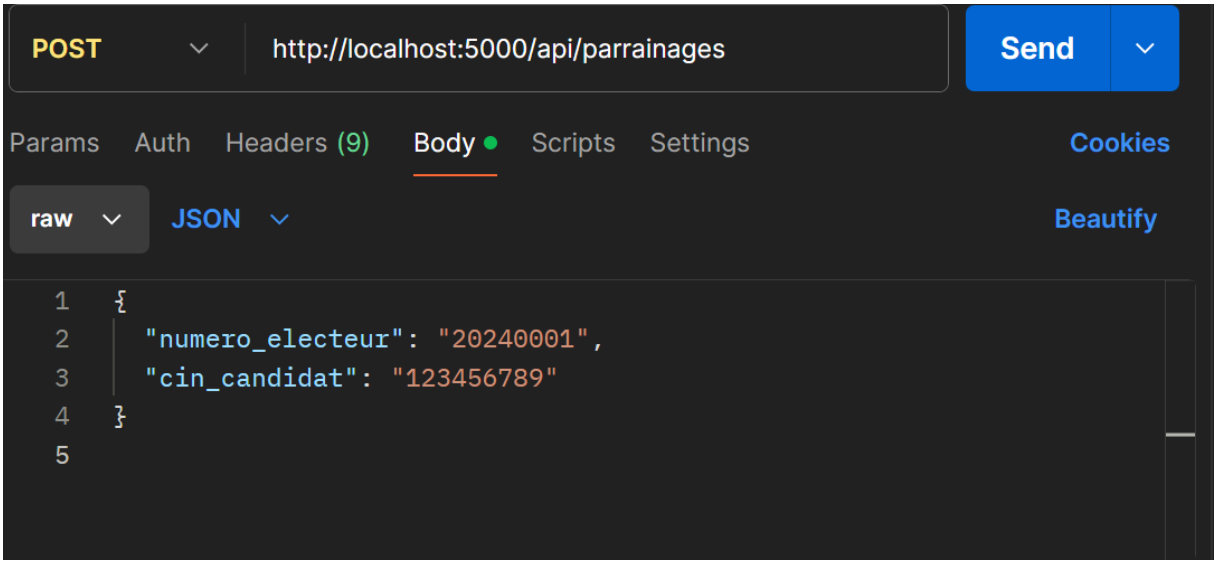
URL : http://localhost:5000/api/electeurs

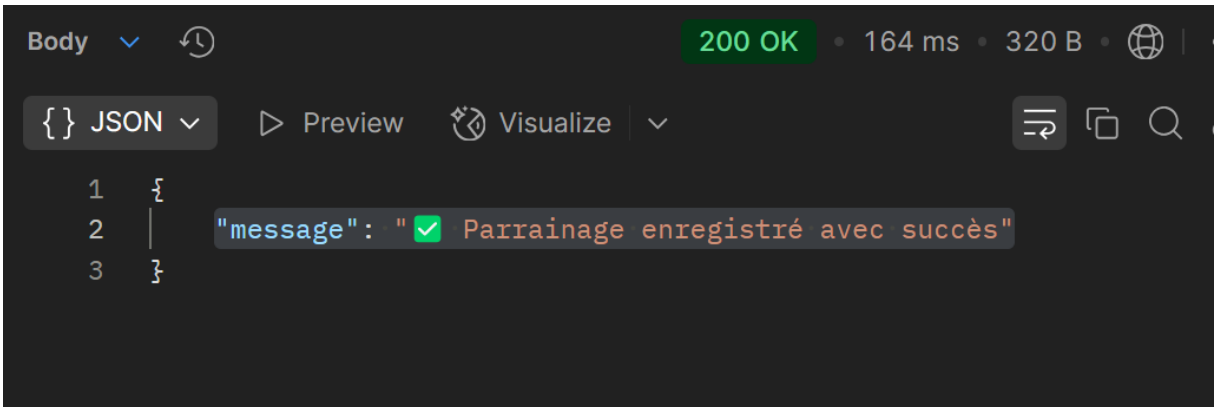


Enregistrer un parrainage

Méthode : POST

URL : http://localhost:5000/api/parrainages






```
MariaDB [gestionparrainage]> SELECT * FROM parrainages;
+----+-----------------+--------------+---------------------+
| id | numero_electeur | cin_candidat | date_parrainage     |
+----+-----------------+--------------+---------------------+
|  1 | 20240001        | 123456789    | 2025-02-19 22:01:08 |
+----+-----------------+--------------+---------------------+
1 row in set (0.001 sec)
```

✅ Résumé de ce qu'on a fait

1. Mise en place du backend avec Node.js + Express + MariaDB

🔹 Création du projet et installation des dépendances :

🔹 Mise en place du serveur Express (server.js).

🔹 Configuration de la base de données avec MariaDB (config/db.js).

2. Authentification des utilisateurs avec JWT

🔹 Envoi d'un code de vérification par email (POST /api/auth/send-code).

🔹 Vérification du code et génération d’un token JWT (POST /api/auth/verify-code).

🔹 Sécurisation des routes avec un middleware JWT (middleware/authMiddleware.js).

🔹 Route protégée pour vérifier l’authentification (GET /api/auth/me).

3. API des candidats

🔹 Ajout d’un candidat (POST /api/candidats).

🔹 Affichage de tous les candidats (GET /api/candidats).

🔹 Modification d’un candidat (PUT /api/candidats/:cin).

🔹 Suppression d’un candidat (DELETE /api/candidats/:cin).

4. API des électeurs

🔹 Ajout d’un électeur (POST /api/electeurs).

🔹 Affichage de tous les électeurs (GET /api/electeurs).

5. API des parrainages

🔹 Création de la table parrainages dans MariaDB.

🔹 Ajout d’un parrainage (POST /api/parrainages).

🔹 Affichage des parrainages enregistrés (GET /api/parrainages).

🔹 Empêcher un électeur de parrainer plusieurs candidats.

Prochaine étape : Intégration avec React (Frontend)

Maintenant que le backend est terminé et fonctionne bien, on va créer le frontend avec React pour :

📌 Interface d’authentification (Connexion avec le code envoyé par email).

📌 Tableau de bord pour les candidats (suivi des parrainages).

📌 Interface publique pour parrainer un candidat.

📌 Interface DGE pour gérer électeurs, candidats et statistiques.