

RéussirSonCV

MANUEL TECHNIQUE

Réalisé par
Mohamed Maghzaoui

16 août
2025





Table des matières

- 1. Introduction.....1
- 2. Architecture du Projet.....2
 - 2.1. Frontend.....2
 - 2.2. Architecture Backend.....5
 - 2.3. Base de données.....10
- 3. Déploiement.....12
- 4. Sécurité de l'application.....15
- 5. Conclusion.....17

1. Introduction

ReussirSonCv est une application web qui aide les utilisateurs à créer, personnaliser et analyser leur CV en ligne.

Elle propose une interface moderne et intuitive permettant de saisir ses informations (profil, expériences, compétences), de personnaliser le design du CV (couleurs, polices, mise en page) et de visualiser le rendu en temps réel.

Le projet repose sur une architecture moderne :

- **Frontend** : React.js, Tailwind CSS, DaisyUI (déployé sur Netlify)
- **Backend** : Django REST Framework (déployé sur Render avec Docker)
- **Base de données** : MySQL hébergé sur Aiven
- **Services additionnels** : GitHub Actions (CI/CD), Brevo (validation email), Figma (maquettes UI/UX).



Vue générale de l'application et navigation principale.

2. Architecture du Projet

2.1. Frontend

Outils & Technologies

- **Vite** : outil de build rapide et léger pour le développement et la compilation du projet React.
- **React.js** : bibliothèque principale pour construire l'interface utilisateur.
- **React Router** : gestionnaire de routes pour la navigation entre les pages.
- **Tailwind CSS + DaisyUI** : framework CSS utilitaire + bibliothèque de composants réutilisables pour une interface moderne et responsive.
- **React Query** : gestion des appels API (caching, synchronisation des données).
- **Axios** : client HTTP configuré avec `withCredentials = true` pour gérer CSRF.
- **react-to-print** : export et téléchargement du CV en PDF.

Structure des pages

- `/` → **Home** : page d'accueil
- `/dashboard` → **Dashboard** : liste des CV de l'utilisateur
- `/dashboard/edit/:id` → **ResumeEditor** : éditeur de CV (création / modification)
- `/dashboard/analyse/:id` → **ResumeAnalyser** : analyse automatique du CV via IA
- `/profile` → **Profile** : gestion du profil utilisateur
- `/mentions-legales`, `/politique-confidentialite`, `/cgu`, `/FAQ` → pages légales et aide
- `*` → **NotFound** : page 404

Composants réutilisables

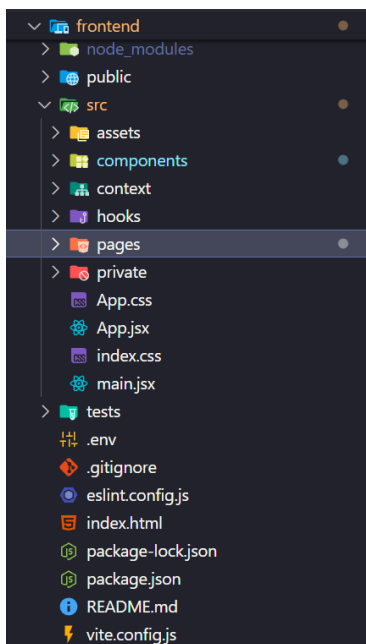
- **Navbar** : navigation principale
- **Footer** : pied de page
- **PrivateRoute** : protection des routes (authentification obligatoire)

Hooks personnalisés

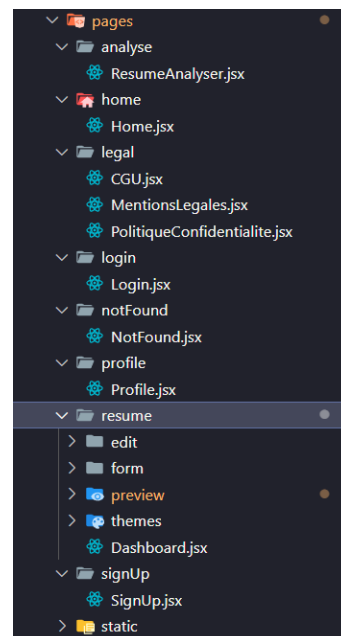
- **useCSRF()** : récupération automatique du token CSRF pour sécuriser les requêtes Axios

Contexte et état global

- **UserContext** : gestion de l'utilisateur (authentification, profil, logout) avec React Query
- **ResumeContext** : gestion des CV de l'utilisateur
- **React Query** : pour la gestion des appels API (caching, synchronisation des données)



Structure principale du frontend



Structure des pages

```
return (  
  <UserContext.Provider value={{ user, loading, error, logout, refetch }}>  
    {children}  
  </UserContext.Provider>  
);  
};  
  
// Custom hook to use the UserContext  
export const useUser = () => useContext(UserContext);
```

UserContext.jsx – Gestion de l'utilisateur Extrait de code clé

```
try {
  const response = await axios.post(`${apiUrl}/register/`, data);
```

Exemple d'appel API avec Axios

```
return (
  <ResumeContext.Provider
    value={{
      resumes,
      loading: isLoading,
      error,
      addResume,
      deleteResume,
      refetch: () => queryClient.invalidateQueries(["resumes"]),
    }}
  >
    {children}
  </ResumeContext.Provider>
);
};

export const useResumes = () => useContext(ResumeContext);
```

ResumeContext – Extrait de code clé

```
function App() {
  useCSRF();
  return (
    <div className="flex flex-col min-h-screen">
      <Navbar />
      <main className="flex-grow">
        <Routes>
          <Route path="/" element={<Home />} />
          <Route
            path="/dashboard"
            element={<PrivateRoute element={<Dashboard />} />}
          />
          <Route
            path="/dashboard/edit/:id"
            element={<PrivateRoute element={<ResumeEditor />} />}
          />
          <Route
            path="/dashboard/analyse/:id"
            element={<PrivateRoute element={<ResumeAnalyser />} />}
          />
          <Route
            path="/profile"
            element={<PrivateRoute element={<Profile />} />}
          />
          <Route path="/mentions-legales" element={<MentionsLegales />} />
          <Route
            path="/politique-confidentialite"
            element={<PolitiqueConfidentialite />}
          />
          <Route path="/cgu" element={<CGU />} />
          <Route path="/FAQ" element={<FAQ />} />
          <Route path="*" element={<NotFound />} />
        </Routes>
      </main>
      <Footer />
    </div>
  );
}
```

App.jsx – Structure des routes

2.2. Architecture Backend

Technologies & Outils

- **Framework** : Django + Django REST Framework
- **Base de données** : MySQL (hébergé sur Aiven)
- **Authentification** : Django **Session Authentication** (gestion sécurisée via sessions et CSRF)
- **ORM** : Django ORM pour la gestion relationnelle des utilisateurs, CV, expériences, compétences, etc.
- **Email** : Envoi de mails via Brevo pour la vérification des comptes
- **IA** : Gemini (Google) intégré pour l'analyse automatique des CV

Structure et logique backend

- **Apps principales** :
 - user/ → gestion des utilisateurs et de l'authentification
 - resume/ → gestion des CV et de leurs sections (expériences, formations, projets, compétences, langues, informations personnelles)
 - ats/ → fonctionnalités liées à l'analyse des CV par IA
- **Autres fichiers et dossiers** :
 - reussirsoncv/ → configuration globale du projet (settings, urls, wsgi)
 - manage.py → script de gestion Django
 - Dockerfile → containerisation du backend
 - requirements.txt → liste des dépendances Python
 - media/ → stockage des fichiers média (images, PDF générés)
 - resumelimages/ → images liées aux CV
 - venv/ → environnement virtuel Python
- **Models** : définition des structures de données via le **Django ORM** pour gérer les relations et les opérations sur la base de données.
- **Serializers** : conversion des objets Django en JSON et vice-versa pour l'API REST, avec validation des données.

- **Views** : séparation entre **authentification** (register_view, login_view, logout_view, user_view, change_password_view, delete_user_view) et **gestion des CV** (CVViewSet et CVSubModelViewSet).
- **Permissions** : toutes les vues CV et sous-modèles utilisent IsAuthenticated pour sécuriser l'accès.
- **Vérifications** : chaque action sur un CV ou ses sous-éléments vérifie que l'utilisateur possède les droits nécessaires (ex. modification/suppression uniquement de ses propres CV).
- **Admin** : configuration des modèles dans l'interface Django Admin pour gérer facilement les utilisateurs et leurs CV depuis le back-office.

Endpoints principaux

Authentification

Endpoint	Méthode	Description
/api/register/	POST	Création d'un nouvel utilisateur
/api/login/	POST	Connexion
/api/logout/	POST	Déconnexion
/api/user/	GET	Récupération des informations utilisateur
/api/user/password/	POST	Changement de mot de passe
/api/user/delete/	DELETE	Suppression du compte
/api/activate/<uid>/<token>/	GET	Activation du compte via email

Gestion des CV / Résumé

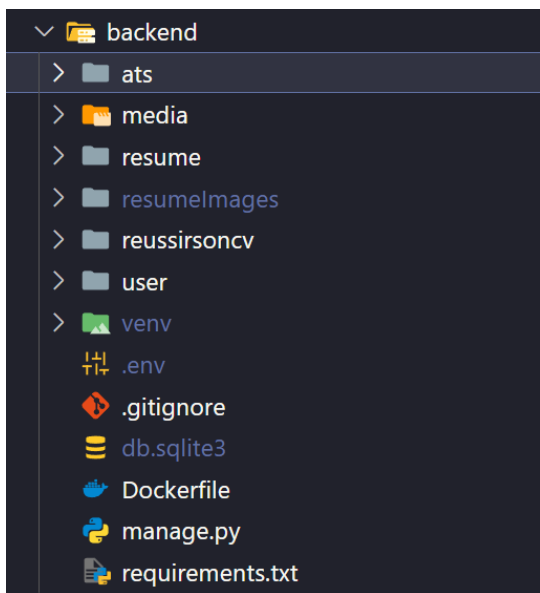
Endpoint	Méthode	Description
/api/cvs/	GET / POST	Liste et création de CV
/api/cvs/:id/	GET / PUT / DELETE	Détails, modification et suppression d'un CV
/api/experiences/	GET / PUT / DELETE	Gestion des expériences
/api/educations/	GET / PUT / DELETE	Gestion des formations
/api/projects/	GET / PUT / DELETE	Gestion des projets
/api/skills/	GET / PUT / DELETE	Gestion des compétences
/api/languages/	GET / PUT / DELETE	Gestion des langues
/api/personal-infos/	GET / PUT / DELETE	Gestion des informations personnelles
/api/analyse-cv/	POST	Analyser le CV via API Gemini

Fonctions supplémentaires

- test_api → vérification du bon fonctionnement de l'API
- csrf_api → récupération du token CSRF

Configuration principale (settings.py)

- Gestion des variables sensibles avec .env
- Middleware de sécurité et CSRF activé
- Configuration des sessions sécurisées
- Email backend configuré pour Brevo
- Throttling des requêtes via Django REST Framework
- Base de données MySQL avec STRICT_TRANS_TABLES pour la fiabilité des transactions



Structure principale du backend

```
from django.contrib.auth.models import AbstractUser, BaseUserManager
from django.db import models

class UserManager(BaseUserManager):
    def create_user(self, email, password=None, **extra_fields):
        if not email:
            raise ValueError("Email required")
        email = self.normalize_email(email)
        extra_fields.setdefault('is_active', False)
        user = self.model(email=email, **extra_fields)
        user.set_password(password)
        user.save()
        return user

    def create_superuser(self, email, password=None, **extra_fields):
        extra_fields.setdefault('is_staff', True)
        extra_fields.setdefault('is_superuser', True)

        return self.create_user(email, password, **extra_fields)

class User(AbstractUser):
    username = None
    email = models.EmailField(unique=True)
    profile_picture = models.ImageField(upload_to='profiles/', blank=True, null=True)
    birthdate = models.DateField(null=True, blank=True)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []

    objects = UserManager()
```

Exemple de model Django – User

```
from rest_framework import serializers
from .models import User
from django.contrib.auth.password_validation import validate_password

class UserSerializer(serializers.ModelSerializer):
    password = serializers.CharField(write_only=True)

    class Meta:
        model = User
        fields = ['id', 'email', 'first_name', 'last_name', 'profile_picture', 'birthdate', 'password']

    def validate_password(self, value):
        # Validate password using Django validators
        validate_password(value, self.instance)
        return value

    def create(self, validated_data):
        user = User(
            email=validated_data['email'],
            first_name=validated_data['first_name'],
            last_name=validated_data['last_name'],
            profile_picture=validated_data.get('profile_picture'),
            birthdate=validated_data.get('birthdate'),
            is_active=False
        )
        user.set_password(validated_data['password'])
        user.save()
        return user
```

Exemple de serializer – User

```
@api_view(['POST'])
@permission_classes([AllowAny])
def register_view(request):
    serializer = UserSerializer(data=request.data)
    if serializer.is_valid():
        user = serializer.save(is_active=False) # in case problem with serializer set user to not acti
        send_verification_email(user, request)

        return Response({'message': 'Utilisateur créé. Vérifie ton email pour l'activer.'}, status=201)
    return Response(serializer.errors, status=400)

#login
@api_view(['POST'])
@permission_classes([AllowAny])
def login_view(request):
    email = request.data.get('email')
    password = request.data.get('password')

    try:
        user = User.objects.get(email=email)
    except User.DoesNotExist:
        return Response({'error': 'invalid'}, status=401)

    if not user.check_password(password):
        return Response({'error': 'invalid'}, status=401)

    if not user.is_active:
        return Response({'error': 'unverified'}, status=401)

    login(request, user)
    return Response({'message': 'Logged in successfully'})
```

Exemple de View Utilisateur – Login et Register

2.3. Base de données

Généralités

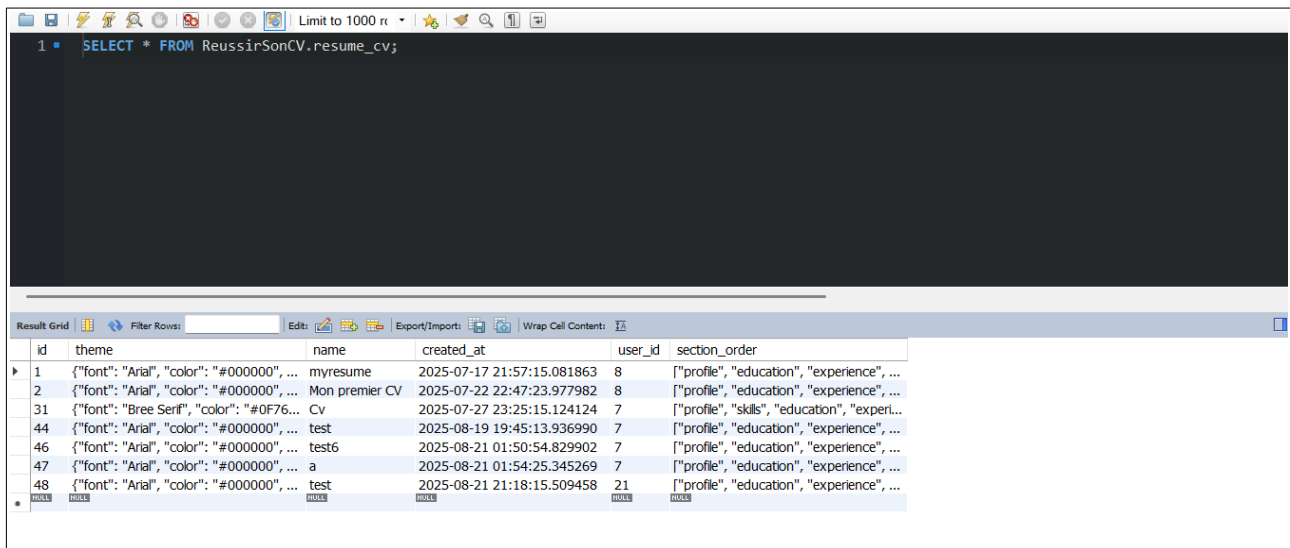
- **Système de gestion** : MySQL
- **Hébergement** : Aiven (cloud, haute disponibilité, scalabilité)
- **Modèle relationnel** : toutes les entités sont reliées par des clés étrangères pour assurer l'intégrité des données.
- **ORM utilisé** : Django ORM pour la gestion des modèles et des relations.

Tables principales

Table	Description	Champs principaux
user	Utilisateurs de l'application	id, email, password, first_name, last_name, profile_picture, birthdate
cv	CV créés par les utilisateurs	id, name, theme, section_order, created_at, user_id (FK)
profile	Section profil d'un CV	id, description, cv_id (FK)
experience	Expériences professionnelles	id, title, company, start_date, end_date, description, address, cv_id (FK)
education	Formation académique	id, degree, institution, start_date, end_date, description, address, cv_id (FK)
project	Projets réalisés	id, title, description, start_date, end_date, cv_id (FK)
skill	Compétences	id, name, cv_id (FK)
language	Langues	id, name, level, cv_id (FK)
personal_info	Informations personnelles détaillées	id, first_name, last_name, title, email, phone_number, address, age, user_picture, linkedin, website, portfolio, cv_id (FK)

Relations et intégrité

- **User** → **CV** : relation **1 à N** (user_id dans cv)
- **CV** → **Profile** : relation **1 à 1** (chaque CV a exactement un profil)
- **CV** → **PersonalInfo** : relation **1 à 1** (chaque CV a exactement une section d'informations personnelles)
- **CV** → **Experience / Education / Project / Skill / Language** : relation **1 à N** (chaque CV peut avoir plusieurs expériences, formations, projets, compétences et langues)
- **Clés étrangères** : chaque sous-table (experience, education, project, skill, language, profile, personal_info) possède une clé étrangère ou une relation 1 à 1 vers la table cv pour garantir que chaque élément appartient à un CV existant.



The screenshot shows a database management interface. At the top, a SQL query is entered: `1 • SELECT * FROM ReussirSonCV.resume_cv;`. Below the query, a table grid displays the results. The table has columns: `id`, `theme`, `name`, `created_at`, `user_id`, and `section_order`. The data rows show various resume entries with their respective themes, names, creation timestamps, user IDs, and section orders.

id	theme	name	created_at	user_id	section_order
1	{ "font": "Arial", "color": "#000000", ...	myresume	2025-07-17 21:57:15.081863	8	["profile", "education", "experience", ...
2	{ "font": "Arial", "color": "#000000", ...	Mon premier CV	2025-07-22 22:47:23.977982	8	["profile", "education", "experience", ...
31	{ "font": "Bree Serif", "color": "#0F76...	Cv	2025-07-27 23:25:15.124124	7	["profile", "skills", "education", "experi...
44	{ "font": "Arial", "color": "#000000", ...	test	2025-08-19 19:45:13.936990	7	["profile", "education", "experience", ...
46	{ "font": "Arial", "color": "#000000", ...	test6	2025-08-21 01:50:54.829902	7	["profile", "education", "experience", ...
47	{ "font": "Arial", "color": "#000000", ...	a	2025-08-21 01:54:25.345269	7	["profile", "education", "experience", ...
48	{ "font": "Arial", "color": "#000000", ...	test	2025-08-21 21:18:15.509458	21	["profile", "education", "experience", ...
...

Exemple de table cv

3. Déploiement

Frontend → Netlify

- **Branch principale** : main
- **Build command** : npm run build
- **Publish directory** : build

Backend → Render (Dockerisé)

- **Branch principale** : main
- **Build command** : pip install -r requirements.txt && python manage.py migrate
- **Start command** : gunicorn reussirsoncv.wsgi:application
- **Docker** : le backend est containerisé pour simplifier le déploiement et garantir un environnement stable.
- **Dockerfile** :

```
FROM python:3.11-slim

# Install system dependencies
RUN apt-get update && apt-get install -y \
    default-libmysqlclient-dev \
    build-essential \
    gcc \
    libssl-dev \
    pkg-config \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /app

# Copy requirements and install
COPY requirements.txt /app/
RUN pip install --upgrade pip \
    && pip install --no-cache-dir -r requirements.txt

# Copy the rest of the code
COPY . /app

# Default command for production (can be overridden in docker-compose for dev)
CMD ["sh", "-c", "python manage.py migrate && gunicorn reussirsoncv.wsgi:application --bind 0.0.0.0:8000 --workers 3"]
```

Tests d'intégration

Backend

- **Tests utilisateur** : création (register), connexion (login), déconnexion (logout), modification du profil et changement de mot de passe.
- **Tests CRUD CV et sous-éléments** : création, lecture, mise à jour et suppression des CV, expériences, formations, projets, compétences, langues et sections personnelles.
- **Vérification des permissions** : s'assure que chaque utilisateur ne peut modifier que ses propres CV et sous-éléments.

Frontend

- **Tests des pages protégées** : vérification que PrivateRoute empêche l'accès aux utilisateurs non authentifiés.
- **Interactions principales** : navigation entre Dashboard, éditeur de CV et les pages statiques,

CI/CD → GitHub Actions

- **Pipeline** pour backend et frontend.
- **Backend** :
 - Installation des dépendances Python
 - Création du fichier .env avec secrets
 - Migration de la base de données
 - Exécution des tests (python manage.py test)
- **Frontend** :
 - Installation des dépendances Node.js
 - Exécution des tests (npm run test)
 - Build du projet (npm run build)

Docker Compose (local/dev)

- Service web : backend Django exposé sur le port 8000.
- Montage du code en volume pour le développement local.
- Commande par défaut pour migration + démarrage via Unicorn.

```

version: '3.9'

services:
  web:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: reussirsoncv_django
    volumes:
      - ./backend:/app
    ports:
      - "8000:8000"
    env_file:
      - ./backend/.env
    # Override command for local dev
    command: sh -c "python manage.py migrate && gunicorn reussirsoncv.wsgi:application --bind 0.0.0.0:8000"

```

Docker Compose – Backend Django

```

frontend:
  name: Frontend React
  runs-on: ubuntu-latest
  defaults:
    run:
      working-directory: frontend
  steps:
    - uses: actions/checkout@v3
    - uses: actions/setup-node@v3
      with:
        node-version: 20

    - name: Install dependencies
      run: npm ci
      working-directory: ./frontend

    - name: Run Tests
      run: npm run test
      working-directory: ./frontend

    - name: Build
      run: npm run build
      working-directory: ./frontend

```

GitHub Actions – Job Frontend React

4. Sécurité de l'application

Authentification et permissions

- Backend Django utilise SessionAuthentication pour sécuriser les endpoints.
- Toutes les routes sensibles nécessitent que l'utilisateur soit authentifié (IsAuthenticated).
- Gestion des sessions sécurisée avec cookies Secure et SameSite=None pour le frontend hébergé séparément.

Protection CSRF

- Middleware Django CsrfViewMiddleware activé.
- Les cookies CSRF sont configurés avec CSRF_COOKIE_SECURE=True et CSRF_COOKIE_SAMESITE=None.
- Frontend récupère automatiquement le token CSRF via le hook useCSRF() pour toutes les requêtes Axios.

CORS (Cross-Origin Resource Sharing)

- Origines autorisées définies avec CORS_ALLOWED_ORIGINS.
- Autorisation des credentials pour les requêtes cross-origin (CORS_ALLOW_CREDENTIALS=True).

Limitation des requêtes

- Limites appliquées avec UserRateThrottle et AnonRateThrottle de Django REST Framework.
- Exemple : 5000 requêtes par jour pour un utilisateur authentifié, 100 par heure pour un utilisateur anonyme.


```
# REST FRAMEWORK
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.SessionAuthentication',
    ],
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ],
    "DEFAULT_THROTTLE_CLASSES": [
        "rest_framework.throttling.UserRateThrottle",
        "rest_framework.throttling.AnonRateThrottle",
    ],
    "DEFAULT_THROTTLE_RATES": {
        "user": "5000/day",
        "anon": "100/hour",
    }
}
```

Limitation des requêtes API

```
# CORS / CSRF
CORS_ALLOWED_ORIGINS = os.getenv('CORS_ALLOWED_ORIGINS', '').split(',')

CSRF_TRUSTED_ORIGINS = os.getenv('CSRF_TRUSTED_ORIGINS', '').split(',')
CORS_ALLOW_CREDENTIALS = True

SESSION_COOKIE_SAMESITE = "None"
CSRF_COOKIE_SAMESITE = "None"
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
```

Configuration CORS et CSRF

```
MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

Middleware de sécurité activé

5. Conclusion

Le projet **ReussirSonCv** offre une solution complète pour la création, la gestion et l'analyse de CV. L'architecture repose sur un frontend moderne en **React.js** avec Tailwind CSS et DaisyUI, et un backend robuste en **Django REST Framework** avec une base de données **MySQL hébergée sur Aiven**.

L'application assure la sécurité des utilisateurs grâce à l'authentification par sessions, la gestion des tokens **CSRF**, et des limites de requêtes pour prévenir les abus. Les fonctionnalités incluent la création et l'édition de CV, l'organisation des sections, l'analyse intelligent, et l'export en PDF.

Grâce à un déploiement automatisé via **Docker**, **Render**, **Netlify**, et **GitHub Actions**, l'application est facilement maintenable et scalable. **ReussirSonCv** illustre ainsi une approche moderne et sécurisée pour la gestion de CV en ligne, combinant performance, sécurité et expérience utilisateur optimale.