Documentation de l'API de Vérification QR Code et Photo

Équipe de Développement

15 février 2025

1 Introduction

Cette documentation explique comment le serveur backend en **Python (Flask)** doit gérer les requêtes de vérification envoyées par une application **Flutter**. L'application envoie des **QR codes** et/ou des **photos** pour authentification.

2 Endpoint attendu

L'application Flutter envoie une requête HTTP **POST** vers :

```
POST http://<IP>:<PORT>/validate_access
```

L'adresse IP et le port sont configurables dans l'application.

3 Format de la requête

L'application supporte trois modes de vérification :
— **QR Code seulement** : `scan_mode="qr"`
— **Photo seulement** : `scan_mode="photo"`
— **Les deux (par défaut)** : `scan_mode="both"`
Les données envoyées peuvent inclure :
— `scan_mode` : `"qr"`, `"photo"` ou `"both"`.
— `encrypted_data` : chaîne encodée (nécessaire pour QR Code).
— `photo` : fichier image (nécessaire pour la vérification photo).

4 Exemple de requête HTTP

Sans photo :

```
POST /validate_access HTTP/1.1
Host: <IP>:<PORT>
Content-Type: multipart/form-data; boundary=----Boundary

------Boundary
Content-Disposition: form-data; name="scan_mode"

qr
------Boundary
Content-Disposition: form-data; name="encrypted_data"

EXEMPLE_QR_CODE
------Boundary--
```

Avec photo :

```
POST /validate_access HTTP/1.1
Host: <IP>:<PORT>
Content-Type: multipart/form-data; boundary=----Boundary

------Boundary
Content-Disposition: form-data; name="scan_mode"

photo
------Boundary
Content-Disposition: form-data; name="photo"; filename="photo
    .jpg"
Content-Type: image/jpeg

[BLOB DE L'IMAGE]
------Boundary--
```

5 Traitement côté Backend (Flask)

Installation des dépendances :

```
pip install flask flask-cors opencv-python numpy
```

Code du serveur Flask :

```
from flask import Flask, request, jsonify
import cv2
import numpy as np

app = Flask(__name__)

```

```python
@app.route('/validate_access', methods=['POST'])
def validate_access():
    try:
        scan_mode = request.form.get('scan_mode', 'both')

        if scan_mode in ['qr', 'both']:
            qr_data = request.form.get('encrypted_data')
            if not qr_data:
                return jsonify({"error": "QR Code manquant"}), 400
            if qr_data != "EXEMPLE_QR_CODE_VALID":
                return jsonify({"error": "QR Code invalide"}), 403

        if scan_mode in ['photo', 'both']:
            if 'photo' not in request.files:
                return jsonify({"error": "Photo manquante"}), 400
            file = request.files['photo']
            np_image = np.frombuffer(file.read(), np.uint8)
            img = cv2.imdecode(np_image, cv2.IMREAD_COLOR)

            if not is_valid_face(img):
                return jsonify({"error": "Photo non reconnue"}), 403

        return jsonify({"status": "Access granted"}), 200

    except Exception as e:
        return jsonify({"error": str(e)}), 500

def is_valid_face(image):
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)
    return len(faces) > 0

if __name__ == '__main__':
    app.run(host="0.0.0.0", port=5000, debug=True)
```

6 Réponses attendues du serveur

Le backend doit répondre clairement à l'application Flutter :

Code HTTP	Description	Réponse JSON
200	Accès autorisé	{"status": "Access granted"}
400	Requête invalide	{"error": "QR Code manquant"}
403	Accès refusé	{"error": "QR Code invalide"}
500	Erreur interne	{"error": "Message d'erreur"}

7 Interaction entre Flutter et le Backend

1. L'utilisateur **scanne un QR Code** et/ou **prend une photo**.
2. L'application Flutter envoie une **requête HTTP POST** au serveur.
3. Le backend **vérifie les données** :
   — QR Code → comparaison avec une base de données.
   — Photo → reconnaissance faciale (via OpenCV).
4. Le backend **retourne une réponse appropriée** :
   — 200 : Accès autorisé.
   — 403 : Accès refusé.
   — 400 : Requête incorrecte.
   — 500 : Erreur interne.

8 Améliorations possibles

— Vérification avancée des QR Codes (base de données).
— Utilisation de l'IA pour la reconnaissance faciale.
— Ajout de logs et monitoring des accès.

9 Conclusion

Cette documentation décrit comment l'API de vérification d'accès doit être implémentée en Python/Flask pour communiquer avec l'application Flutter. En suivant ces instructions, le serveur pourra gérer efficacement les authentifications basées sur QR Code et reconnaissance faciale.