Cahier des Charges – Backend d’Analyse d’Émotions (FastAPI + Gradio + OpenRouter)

1. Objectif du projet

Le backend a pour fonction de servir de couche d’intelligence entre l’application mobile et deux services d’IA tiers.

Il réalise deux tâches :

Détection d’émotion sur image via un modèle hébergé sur Hugging Face.

Génération d’un retour textuel empathique et de conseils personnalisés via OpenRouter (ex. DeepSeek).

L’architecture doit rester simple, modulaire et extensible afin de permettre l’évolution future du modèle ou de l’API.

2. Architecture générale

Le backend est développé avec FastAPI et expose deux endpoints clés :

/analyze : analyse d’image

/generate : génération de texte à partir des résultats d’analyse

Structure des fichiers :

/backend

│── main.py

│── requirements.txt

│── .env.example

└── README_BACKEND.md

3. Spécifications Fonctionnelles

3.1. Endpoint /analyze (Analyse d’image)

Entrée

Image envoyée via un formulaire multipart (file).

Traitement interne

Validation du fichier : format image obligatoire.

Sauvegarde temporaire de l’image.

Envoi de l’image au modèle Hugging Face via Gradio Client.

Interprétation de la sortie (qui peut être texte, liste ou dictionnaire).

Normalisation du label émotionnel :

Uniformisation vers un label standard (ex. joy → happy, sadness → sad).

Extraction :

émotion principale

pourcentage de confiance

émotions alternatives

sortie brute (transparence)

Sortie

JSON contenant tous les éléments nécessaires pour la suite du traitement.

3.2. Endpoint /generate (Génération des conseils)

Entrée

JSON produit par /analyze.

Aucune image n’est envoyée à ce stade.

Traitement interne

Construction d’un prompt détaillé destiné au modèle OpenRouter.

Transmission du prompt au modèle via API.

Obligation d’un retour strict au format JSON contenant :

description : explication courte de l’émotion détectée

tips : liste de trois conseils

message : message empathique personnalisé

Vérification de la validité JSON et présence des trois champs obligatoires.

Nettoyage final des valeurs (espaces, retours ligne).

Sortie

JSON prêt à être affiché dans l'application mobile.

4. Spécifications Techniques

4.1. Technologies Utilisées

FastAPI : création de l'API REST

Uvicorn : serveur ASGI

Gradio Client : communication avec Hugging Face

Requests : appels HTTP vers OpenRouter

python-multipart : gestion du multipart/form-data

dotenv : gestion des variables d'environnement

4.2. Fichiers de Configuration

.env.example

Doit contenir :

HF_TOKEN : optionnel (si le modèle Hugging Face est privé)

OPENROUTER_API_KEY : obligatoire

HOST et PORT : paramètres de lancement

L'utilisateur du backend doit créer un fichier .env à partir de ce modèle.

4.3. requirements.txt

Liste stricte des dépendances nécessaires au fonctionnement.

Installation via :

pip install -r requirements.txt

5. Contraintes Techniques et Qualité

Le backend doit fonctionner avec Python 3.10+.

Le service doit rester stateless (aucune persistance de données utilisateur).

L'image ne doit être conservée que temporairement et supprimée après traitement.

Les réponses OpenRouter doivent être strictement JSON valides ; la validation est obligatoire.

Le backend doit renvoyer des messages d'erreur clairs en cas de :

format d'image invalide

échec du modèle Hugging Face

JSON mal formé retourné par OpenRouter

6. Sécurité

Les clés API doivent être chargées exclusivement depuis .env.

Aucun secret ne doit être exposé dans le code.

Les images doivent être supprimées après analyse.

Le serveur doit rejeter les fichiers corrompus ou trop volumineux.

7. Documentation d'Installation (README_BACKEND.md)

Le README doit expliquer :

Création d'un environnement Python

Installation des dépendances

Configuration du fichier .env

Commande de lancement :

uvicorn main:app --reload

Exemple d'appel des endpoints avec curl

Workflow global

8. Schéma Global de Fonctionnement

L'utilisateur envoie une image → /analyze.

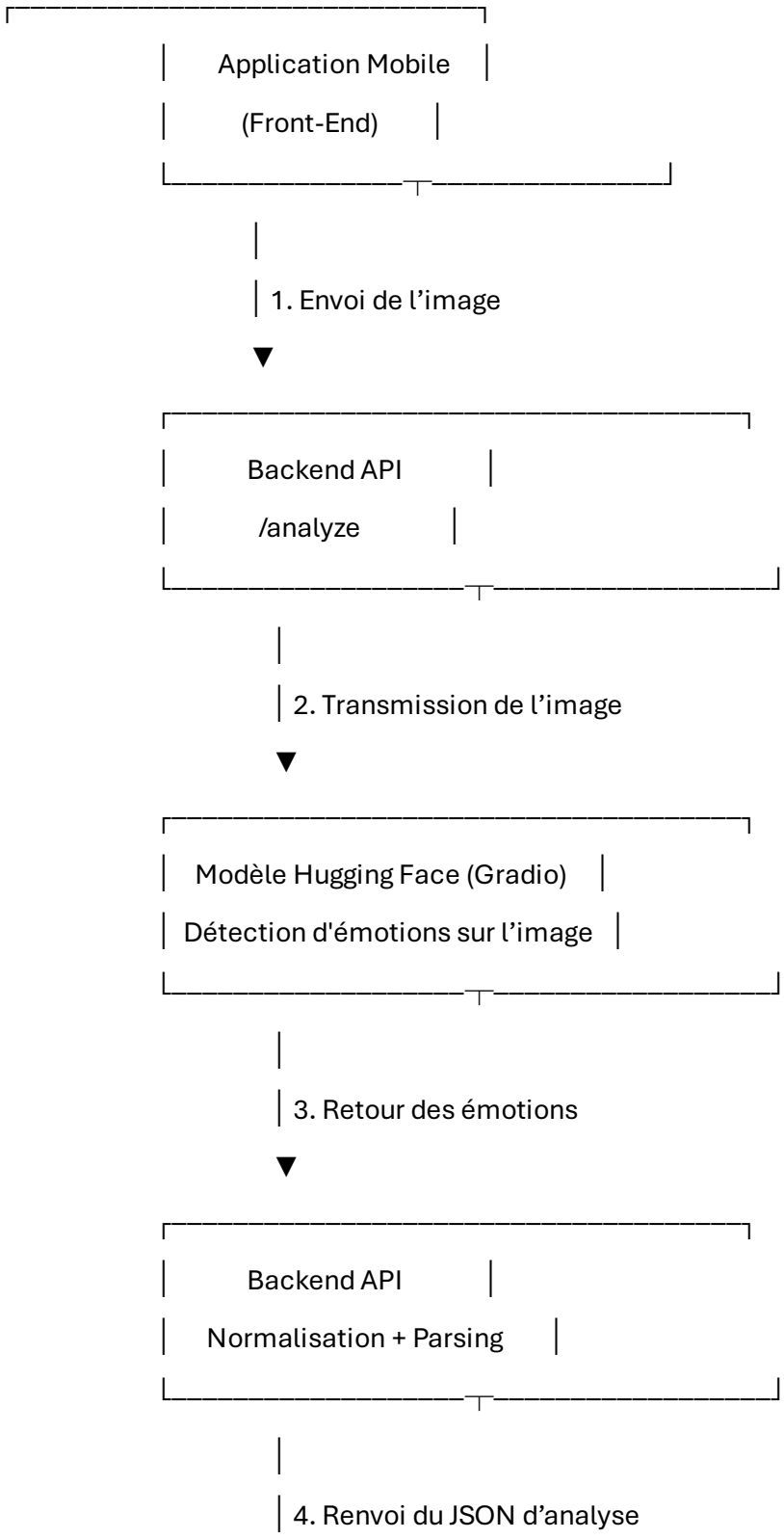
Le backend envoie l'image à Hugging Face → reçoit les émotions.

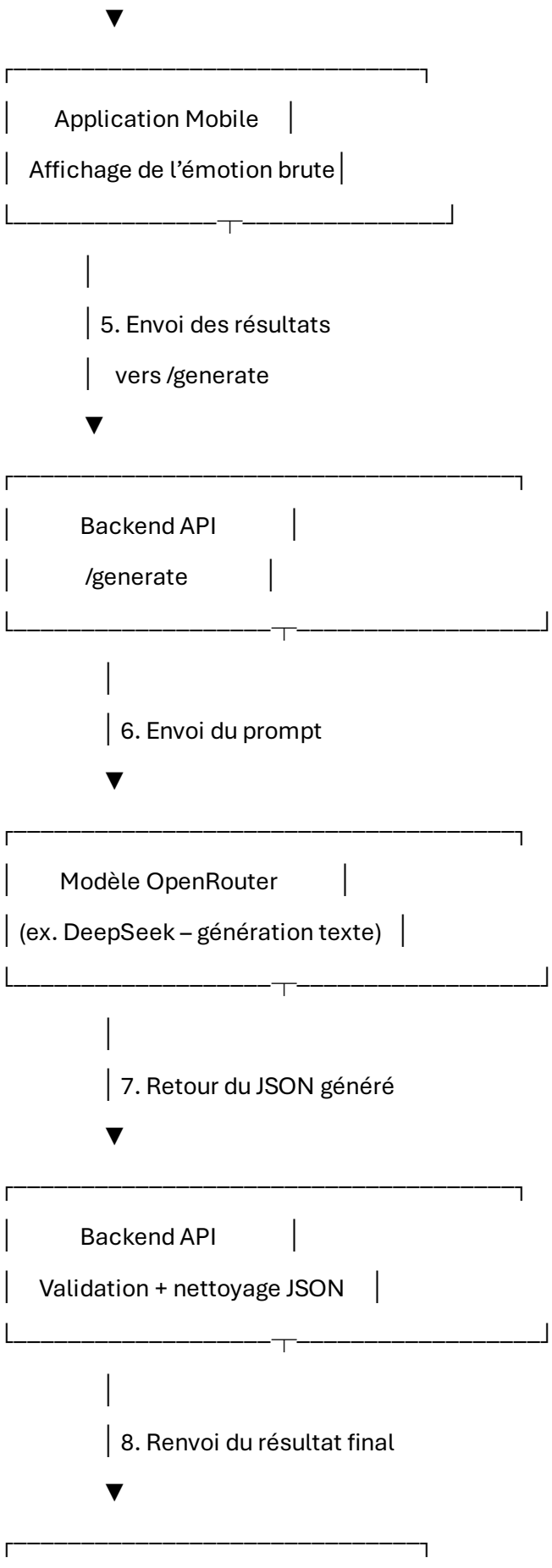
Les données analysées sont renvoyées au front.

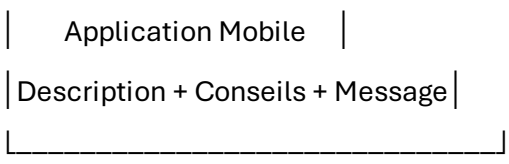
Le front appelle /generate avec ces données.

Le backend interroge OpenRouter → génère description + conseils + message.

Le front affiche directement le retour au format JSON.


Application Mobile
(Front-End)
1. Envoi de l’image
Backend API
/analyze
2. Transmission de l’image
Modèle Hugging Face (Gradio)
Détection d'émotions sur l’image
3. Retour des émotions
Backend API
Normalisation + Parsing
4. Renvoi du JSON d’analyse


Application Mobile
Affichage de l'émotion brute
5. Envoi des résultats
vers /generate
Backend API
/generate
6. Envoi du prompt
Modèle OpenRouter
(ex. DeepSeek – génération texte)
7. Retour du JSON généré
Backend API
Validation + nettoyage JSON
8. Renvoi du résultat final


Application Mobile
Description + Conseils + Message