Cahier des Charges Fonctionnel et Technique

Projet : Backend d’ Analyse d’ Émotions

Version : 1.0 **Date :** 2 décembre 2025

---

1. Introduction et Objectifs du Projet

Le présent document a pour objet de définir les spécifications fonctionnelles et techniques relatives au développement d’ un **Backend d’ Analyse d’ Émotions**. Ce service doit agir comme une couche d’ intelligence intermédiaire entre une application mobile front-end et des services d’ intelligence artificielle tiers.

1.1. Objectif Principal

L’ objectif principal du backend est de réaliser deux fonctions clés :

1. **Détection d’ émotion** sur une image fournie par l’ utilisateur via un modèle hébergé sur Hugging Face.
2. **Génération d’ un retour textuel** empathique et de conseils personnalisés basés sur l’ émotion détectée, en utilisant un modèle de langage via OpenRouter.

1.2. Principes d’ Architecture

L’ architecture doit être conçue pour être **simple, modulaire et extensible**, afin de faciliter l’ évolution future, notamment le changement de modèles d’ IA ou d’ APIs tierces.

---

2. Architecture Générale

Le backend sera développé en utilisant le framework **FastAPI** pour exposer une API REST.

2.1. Structure du Projet

Le projet sera organisé de la manière suivante :

Fichier/Dossier	Description
`/backend`	Dossier racine du projet.
`main.py`	Fichier principal contenant la définition de l'application FastAPI et les endpoints.
`requirements.txt`	Liste stricte des dépendances Python.
`.env.example`	Fichier modèle pour la configuration des variables d'environnement.
`README_BACKEND.md`	Documentation d'installation, de configuration et d'utilisation.

2.2. Endpoints de l'API

Le backend exposera deux endpoints principaux :

Endpoint	Méthode	Description
`/analyze`	`POST`	Analyse une image pour en extraire l'émotion principale.
`/generate`	`POST`	Génère un retour textuel personnalisé à partir des résultats de l'analyse.

---

3. Spécifications Fonctionnelles

3.1. Endpoint `/analyze` (Analyse d'image)

Élément	Spécification
Entrée	Image envoyée via un formulaire `multipart/form-data` (champ `file`).
Validation	Le format du fichier doit être une image valide.
Traitement Interne	1. Sauvegarde temporaire de l'image. 2. Envoi de l'image au modèle Hugging Face via `Gradio Client`. 3. Interprétation de la sortie du modèle (texte, liste ou dictionnaire). 4. **Normalisation du label émotionnel** (ex. `joy` → `happy`).
Extraction des Données	Émotion principale, pourcentage de confiance, émotions alternatives, et sortie brute du modèle (pour transparence).
Sortie	Réponse JSON contenant toutes les données extraites et normalisées, prêtes pour l'appel à `/generate`.

3.2. Endpoint `/generate` (Génération des conseils)

Élément	Spécification
Entrée	Réponse JSON complète produite par l'endpoint `/analyze`. **Aucune image n'est envoyée à ce stade.**
Traitement Interne	1. Construction d'un **prompt détaillé** pour le modèle OpenRouter. 2. Transmission du prompt au modèle via l'API OpenRouter. 3. **Contrainte de format :** Le modèle OpenRouter doit retourner une réponse **strictement au format JSON**.
Format JSON Requis	Le JSON retourné par OpenRouter doit contenir les trois champs obligatoires suivants : - `description` : explication courte de l'émotion détectée. - `tips` : liste de trois conseils personnalisés. - `message` : message empathique personnalisé.
Validation de Sortie	Vérification obligatoire de la validité du JSON et de la présence des trois champs. Nettoyage final des valeurs (suppression des espaces/retours ligne superflus).
Sortie	Réponse JSON finale, prête à être affichée directement par l'application mobile.

---

4. Spécifications Techniques

4.1. Technologies et Dépendances

Technologie	Rôle
FastAPI	Framework principal pour la création de l'API REST.
Uvicorn	Serveur ASGI pour l'exécution de l'application FastAPI.
Gradio Client	Communication avec le modèle d'analyse d'émotions hébergé sur Hugging Face.
Requests	Gestion des appels HTTP vers l'API OpenRouter.
python-multipart	Gestion des données `multipart/form-data` pour l'upload d'images.
dotenv	Gestion sécurisée des variables d'environnement.
Python	Version minimale requise : **3.10+**.

4.2. Configuration des Fichiers

Le fichier `.env.example` doit servir de modèle pour la configuration. L'utilisateur du backend devra créer un fichier `.env` à partir de ce modèle.

Variable d'Environnement	Obligatoire	Description
`OPENROUTER_API_KEY`	Oui	Clé d'API pour l'accès au service OpenRouter.
`HF_TOKEN`	Non	Jeton d'accès pour les modèles Hugging Face privés (optionnel).
`HOST`	Non	Adresse d'écoute du serveur (par défaut : `0.0.0.0`).
`PORT`	Non	Port d'écoute du serveur (par défaut : `8000`).

---

5. Contraintes Techniques et Qualité

5.1. Contraintes Générales

- **Stateless :** Le service doit rester **stateless** ; aucune persistance de données utilisateur (sessions, historique) n'est autorisée.
- **Gestion des Fichiers :** L'image envoyée doit être conservée uniquement de manière **temporaire** et **supprimée immédiatement** après le traitement par le modèle Hugging Face.
- **Validation JSON :** La validation de la réponse JSON retournée par OpenRouter est **obligatoire** pour garantir l'intégrité des données transmises au front-end.

5.2. Gestion des Erreurs

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

- Format d'image invalide ou fichier corrompu/trop volumineux.
- Échec de la communication ou de l'exécution du modèle Hugging Face.
- JSON mal formé ou incomplet retourné par l'API OpenRouter.

5.3. Sécurité

- **Secrets :** Les clés API (`OPENROUTER_API_KEY`, `HF_TOKEN`) doivent être chargées **exclusivement** depuis le fichier `.env`. **Aucun secret ne doit être exposé dans le code source.**
- **Nettoyage :** Suppression systématique des images après analyse pour des raisons de confidentialité et de sécurité.

---

6. Documentation et Workflow

6.1. Documentation d'Installation (`README_BACKEND.md`)

Le fichier `README_BACKEND.md` doit fournir une documentation complète incluant :

1. Procédure de création d’ un environnement Python (virtuel).
2. Instructions pour l’ installation des dépendances (`pip install -r requirements.txt`).
3. Guide de configuration du fichier `.env` .
4. Commande de lancement du serveur (`uvicorn main:app --reload`).
5. Exemples d’ appel des endpoints (`/analyze` et `/generate`) avec `curl` .

6.2. Schéma Global de Fonctionnement (Workflow)

Le workflow de l’ application est le suivant :

1. L’ application mobile envoie l’ image de l’ utilisateur à l’ endpoint `/analyze` .
2. Le backend envoie l’ image au modèle Hugging Face et reçoit les données d’ émotion.
3. Les données analysées (JSON) sont renvoyées à l’ application mobile.
4. L’ application mobile appelle l’ endpoint `/generate` en lui transmettant les données d’ émotion reçues.
5. Le backend interroge OpenRouter avec un prompt basé sur ces données.
6. Le modèle OpenRouter génère la `description` , les `tips` et le `message` au format JSON strict.
7. Le backend renvoie le JSON final à l’ application mobile, qui l’ affiche directement.