```
import os
import joblib
import pandas as pd
import shap
from flask import Flask, jsonify, request
import warnings

warnings.filterwarnings("ignore", message="LightGBM binary classifier with
TreeExplainer shap values output has changed to a list of ndarray")

# Initialisation de l'application Flask
app = Flask(__name__)

# Définir le répertoire courant
current_directory = os.path.dirname(os.path.abspath(__file__))

# Charger le modèle
model_path = os.path.join(current_directory, "saved_model",
"best_lgbmb_model.joblib")
print(f"Chargement du modèle depuis {model_path}")
model = joblib.load(model_path)
print("Modèle chargé avec succès")

# Charger le DataFrame
df_train_smote_path = os.path.join(current_directory, "saved_model",
"df_train_smote_corrected_100rows_with_id.joblib")
print(f"Chargement du DataFrame depuis {df_train_smote_path}")
df_train_smote = joblib.load(df_train_smote_path)
print("DataFrame chargé avec succès")

# Liste des colonnes attendues par le modèle
model_columns = model.booster_.feature_name()
print(f"Colonnes du modèle : {model_columns}")

@app.route("/")
def home():
    return "API de prédiction avec Flask"

@app.route("/predict", methods=['GET'])
def predict():
    print("Requête reçue pour prédiction")

    sk_id_curr = request.args.get("SK_ID_CURR")
    if sk_id_curr is None:
        print("Erreur : SK_ID_CURR est manquant dans la requête")
        return jsonify({"error": "SK_ID_CURR est manquant dans la requête"}),
400

    if not sk_id_curr.isdigit():
        print(f"Erreur : SK_ID_CURR contient des caractères non numériques :
{sk_id_curr}")
        return jsonify({"error": "SK_ID_CURR doit être un entier numérique"}),
400
```

```
    if 'SK_ID_CURR' not in df_train_smote.columns:
        print("Erreur : La colonne 'SK_ID_CURR' n'existe pas dans le DataFrame")
        return jsonify({"error": "La colonne 'SK_ID_CURR' n'existe pas dans le
DataFrame"}), 404

    print(f"Filtrage du DataFrame pour SK_ID_CURR: {sk_id_curr}")
    sample = df_train_smote[df_train_smote['SK_ID_CURR'] == int(sk_id_curr)]

    if sample.empty:
        print(f"Erreur : Aucun échantillon trouvé pour SK_ID_CURR:
{sk_id_curr}")
        return jsonify({"error": f"Aucun échantillon trouvé pour SK_ID_CURR:
{sk_id_curr}"}), 404


    # Mapping des noms de colonnes
    columns_mapping = {i: f"Column_{i}" for i in range(len(sample.columns))}

    # Renommer les colonnes du DataFrame
    sample.rename(columns=columns_mapping, inplace=True)

    print(f"Colonnes après renommage : {sample.columns.tolist()}")

    # Garder uniquement les colonnes attendues par le modèle
    sample_for_prediction = sample[model_columns]

    # Prédire
    print("Lancement de la prédiction")
    prediction = model.predict_proba(sample_for_prediction)
    proba = prediction[0][1]
    print(f"Probabilité calculée : {proba}")
    print("Prédiction terminée")
    # Calculer les valeurs SHAP pour l'échantillon donné
    print("Calcul des valeurs SHAP")
    explainer = shap.TreeExplainer(model)
    shap_values = explainer.shap_values(sample_for_prediction)

    if isinstance(shap_values, list) and len(shap_values) == 1:
        shap_values = shap_values[0]


    # Limiter les données renvoyées pour la lisibilité (par exemple, les 10
premières features)
    num_features_to_show = 10
    limited_shap_values = shap_values[0][:num_features_to_show].tolist()
    limited_feature_names =
sample_for_prediction.columns[:num_features_to_show].tolist()
    limited_feature_values =
sample_for_prediction.values[0][:num_features_to_show].tolist()

    print("Réponse générée avec succès")

    return jsonify({
        'probability': round(proba * 100, 2),
```

```
        'shap_values': limited_shap_values,
        'feature_names': limited_feature_names,
        'feature_values': limited_feature_values
    })

if __name__ == "__main__":
    port = os.environ.get("PORT", 5000)
    print(f"Lancement de l'application sur le port {port}")
    app.run(debug=False, host="0.0.0.0", port=int(port))
```