In [14]:
```
# ================================
#   Modèle XGBoost - Classification
#   HALIMA_DRIOUCH
# ================================
import pandas as pd
import xgboost as xgb
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_mat
import joblib
```

In [15]:
```
# ———————————————————————
# Variables globales
# ———————————————————————
df = None
modele = None
encoders = {}
label_encoder_classe = None
colonnes_features = None
colonne_classe = None
X_train = X_test = y_train = y_test = None
```

In [16]:
```
# ———————————————————————
# 1) CHARGEMENT
# ———————————————————————
def charger(fichier_csv, target='is_fit'):
    global df, colonne_classe
    df = pd.read_csv(fichier_csv)
    colonne_classe = target
```

In [17]:
```
# ———————————————————————
# 2) PRÉPARATION DES DONNÉES
# ———————————————————————
def preparer(test_size=0.2, random_state=42):
    global df, X_train, X_test, y_train, y_test, encoders, label_encoder_classe,
    data = df.copy()

    # Encodage des colonnes catégorielles
    for col in ['smokes', 'gender']:
        le = LabelEncoder()
        data[col] = le.fit_transform(data[col].astype(str))
        encoders[col] = le

    # Séparer features et target
    colonnes_features = [c for c in data.columns if c != colonne_classe]
    X = data[colonnes_features]
    y = data[colonne_classe]

    # Encodage de la target
    label_encoder_classe = LabelEncoder()
    y = label_encoder_classe.fit_transform(y)

    # Split train/test
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=random_state
    )
```

In [18]:
```
# ──────────────────────────────────
# 3) ENTRAÎNEMENT
# ──────────────────────────────────
def entrainer():
    global modele, X_train, y_train
    modele = xgb.XGBClassifier(eval_metric='logloss')
    modele.fit(X_train, y_train)
```

In [19]:
```
# ──────────────────────────────────
# 4) ÉVALUATION
# ──────────────────────────────────
def evaluer():
    y_pred = modele.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print("\nÉVALUATION DU MODÈLE")
    print(f"• Accuracy : {acc:.4f} ({acc*100:.2f}%)\n")
    print("Rapport de classification :")
    print(classification_report(y_test, y_pred))

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8,6))
    plt.barh(['TN','FP','FN','TP'], cm.flatten())
    plt.title("Matrice de Confusion")
    plt.xlabel("Nombre")
    plt.show()
```

In [20]:
```
# ──────────────────────────────────
# 5) SAUVEGARDE
# ──────────────────────────────────
def sauvegarder(nom="modele_fitness.pkl"):
    package = {
        'modele': modele,
        'encoders': encoders,
        'label_encoder_classe': label_encoder_classe,
        'colonnes_features': colonnes_features,
        'colonne_classe': colonne_classe
    }
    joblib.dump(package, nom)
    print(f"Modèle complet sauvegardé : {nom}")
```

In [21]:
```
# ──────────────────────────────────
# 6) IMPORTANCE DES VARIABLES
# ──────────────────────────────────
def importance(top=10):
    plt.figure(figsize=(10,6))
    xgb.plot_importance(modele, max_num_features=top, importance_type='weight')
    plt.title("Importance des variables (XGBoost)")
    plt.tight_layout()
    plt.show()
```

In [22]:
```
# ──────────────────────────────────
# 7) PRÉDICTION POUR UN ENREGISTREMENT
# ──────────────────────────────────
def predict_input(dict_data):
    df_input = pd.DataFrame([dict_data])
    for col, le in encoders.items():
        if col in df_input.columns:
            df_input[col] = le.transform(df_input[col].astype(str))
```

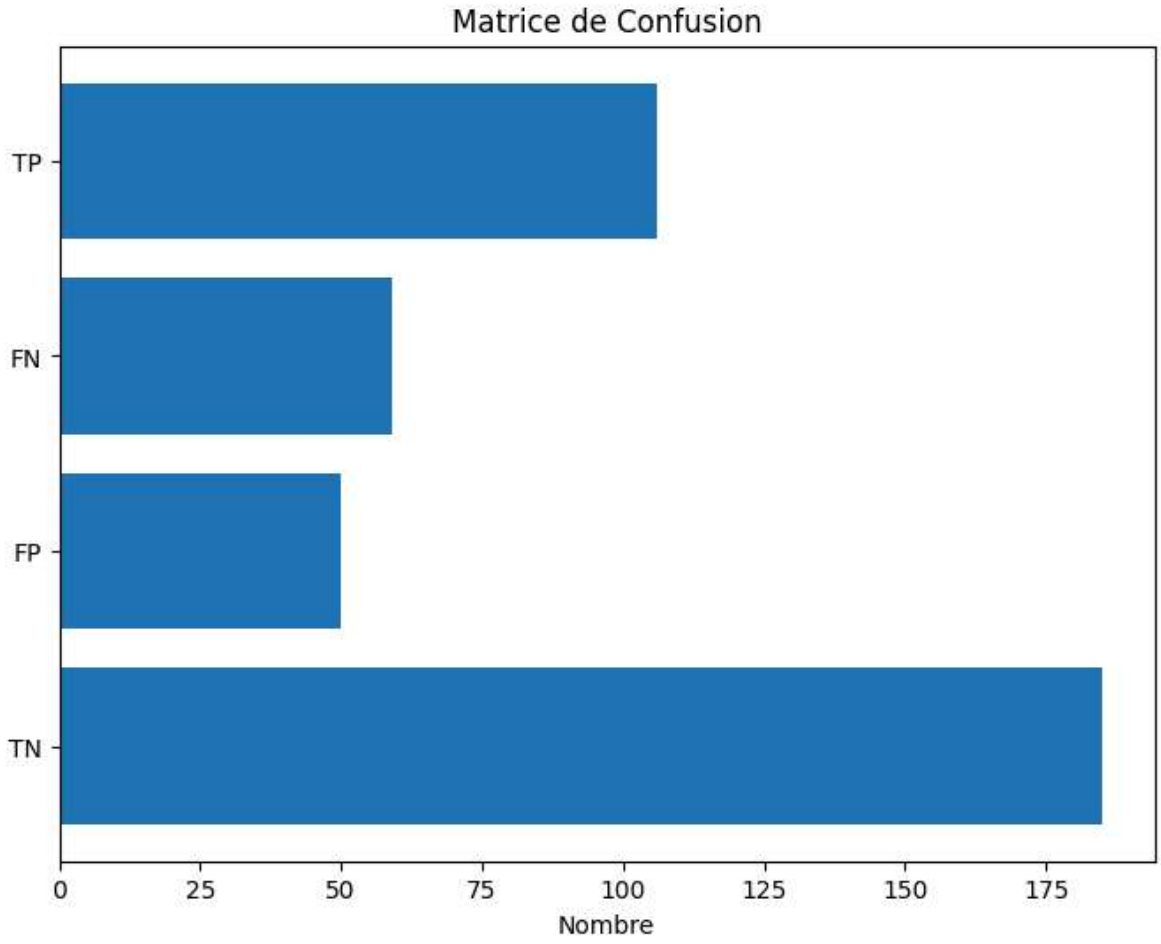
```
    df_input = df_input[colonnes_features]
    pred = modele.predict(df_input)
    return label_encoder_classe.inverse_transform(pred)[0]
```

In [23]:
```
# ──────────────────────────────────
# 8) PRÉDICTION POUR UN CSV
# ──────────────────────────────────
def from_csv(fichier_csv):
    df_input = pd.read_csv(fichier_csv)
    for col, le in encoders.items():
        if col in df_input.columns:
            df_input[col] = le.transform(df_input[col].astype(str))
    X = df_input[colonnes_features]
    preds = modele.predict(X)
    df_input["Prediction"] = label_encoder_classe.inverse_transform(preds)
    return df_input
```

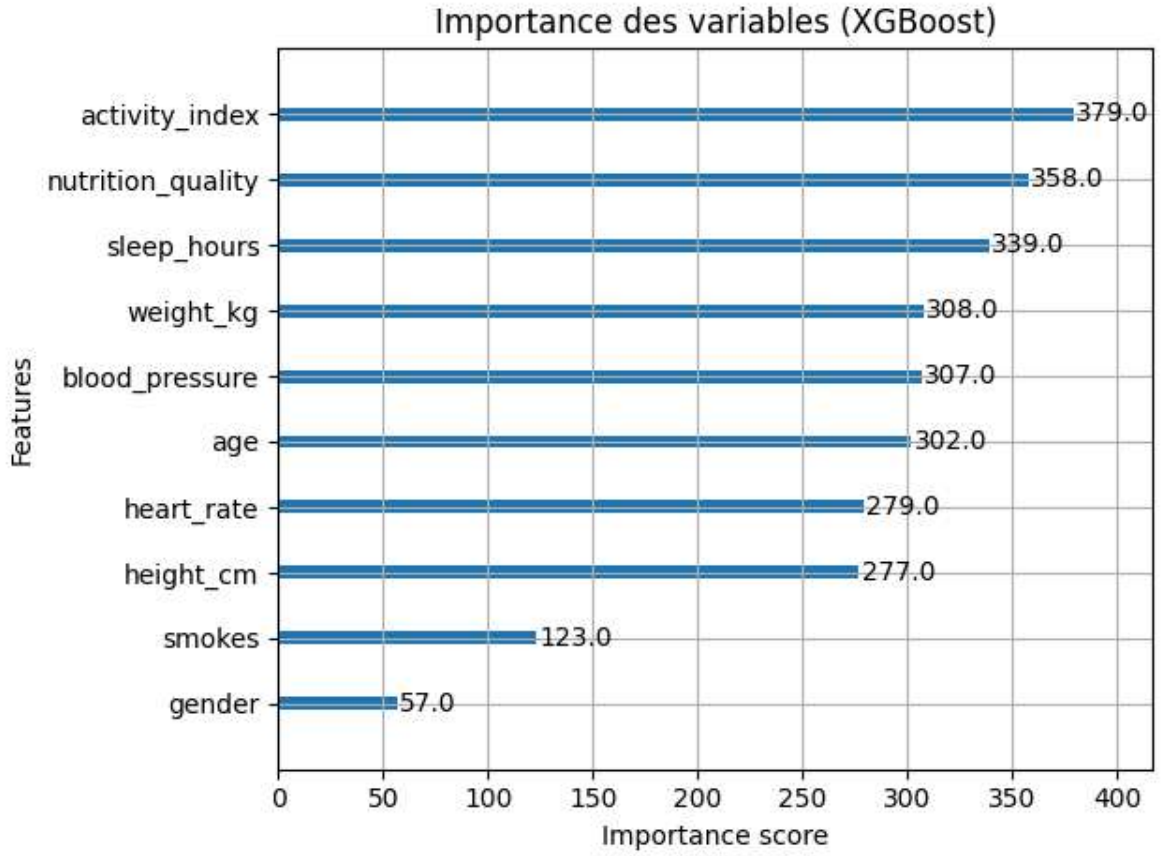
In [24]:
```
# ──────────────────────────────────
# PIPELINE COMPLET
# ──────────────────────────────────
def executer(csv='fitness_dataset.csv', target='is_fit'):
    charger(csv, target)
    preparer()
    entrainer()
    evaluer()
    sauvegarder()
    importance()
    print("Pipeline terminé avec succès.")
```

In [25]:
```
# ──────────────────────────────────
# UTILISATION
# ──────────────────────────────────
executer('fitness_dataset.csv', 'is_fit')
```

```
ÉVALUATION DU MODÈLE
• Accuracy : 0.7275 (72.75%)

Rapport de classification :
              precision    recall  f1-score   support

           0       0.76      0.79      0.77       235
           1       0.68      0.64      0.66       165

    accuracy                           0.73       400
   macro avg       0.72      0.71      0.72       400
weighted avg       0.73      0.73      0.73       400
```




```
Modèle complet sauvegardé : modele_fitness.pkl
<Figure size 1000x600 with 0 Axes>
```




```
Pipeline terminé avec succès.
```