Rapport de Projet

GlowCare

Application de Skincare avec Chatbot Intégré pour un Accompagnement Personnalisé

Réaliser par :
Khadija Nachid Idrissi
&&
Rajae Fdili

Encadré par :
Ahmed Amamo

1-Contexte et Objectif

Dans un marché saturé de produits cosmétiques, il est souvent difficile pour les utilisateurs d'identifier les soins réellement adaptés à leur peau. C'est dans cette optique que nous avons développé GlowCare : une application web intelligente qui propose des recommandations personnalisées de produits de skincare, accompagnée d'un chatbot interactif pour un accompagnement en temps réel.

Ce projet a été réalisé dans le cadre de notre formation, dans le but de mobiliser des technologies modernes — FastAPI pour le backend, React pour l'interface utilisateur, le machine learning pour l'analyse des besoins, et un chatbot basé sur l'intelligence artificielle pour guider l'utilisateur — afin de répondre à un besoin concret : aider chacun à construire une routine de soin efficace, simple et personnalisée.

Les objectifs de GlowCare sont les suivants :

- Fournir à chaque utilisateur une sélection ciblée de produits en fonction de son type de peau, de son âge, de son budget et de ses problématiques cutanées.
- Générer des conseils d'utilisation pertinents, grâce à un modèle d'apprentissage automatique basé sur les ingrédients des produits.
- Offrir un accompagnement interactif et personnalisé via un chatbot capable de répondre aux questions, de conseiller en temps réel, et d'aider l'utilisateur à mieux comprendre ses besoins et sa routine de soin.

Avec GlowCare, l'intelligence artificielle devient un outil au service de la beauté, en proposant une approche accessible, personnalisée et fiable du soin de la peau, enrichie par une interaction humaine facilitée par le chatbot.

2-Traitement et Préparation des Données

Avant d'effectuer la moindre analyse, il est essentiel de passer par une phase de préparation des données. Le fichier initial présentait plusieurs limitations qui rendaient

son exploitation difficile. On a donc mis en place un pipeline de traitement permettant de nettoyer, structurer et enrichir le dataset afin d'obtenir une base cohérente, fiable et prête à être utilisée.

2.1 Exploration initiale du dataset

Lors du premier chargement du fichier `datasheet.csv`, on a constaté que plusieurs colonnes clés étaient présentes, comme :

- le nom du produit,
- la marque,
- le type de produit,
- les ingrédients,
- le pays d'origine,
- les effets après utilisation.

Cependant, plusieurs problèmes ont été relevés :

- Des noms de colonnes mal orthographiés (`ingridients` au lieu de `ingredients`),
- Une quantité importante de valeurs manquantes, notamment dans les champs `country` et `afterUse`,
- Une forte hétérogénéité dans les formats des chaînes de caractères (majuscule/minuscule, ponctuation, espaces...).

2.2 Nettoyage des données

Dans un premier temps, on a procédé à un nettoyage basique du dataset :

- Suppression des lignes avec trop de données manquantes,
- Correction des noms de colonnes pour les rendre plus explicites,
- Standardisation des textes : passage en minuscules, suppression des espaces inutiles et des caractères spéciaux.

Ce nettoyage a permis d'avoir une structure plus claire et d'éviter les erreurs lors des traitements suivants.

2.3 Traitement des ingrédients

La colonne des ingrédients contenait des listes de composants cosmétiques sous forme de chaînes brutes. Pour la rendre exploitable :

- On a converti chaque chaîne en liste proprement formatée (`clean_ingreds`),
- On a supprimé les doublons et filtré les termes non significatifs (ex. : conservateurs génériques ou mentions répétitives),
- Un nettoyage lexical a été appliqué pour normaliser les noms des ingrédients (acide hyaluronique, niacinamide, etc.).

L'objectif était de pouvoir, par la suite, identifier la composition réelle des produits et réaliser des regroupements ou filtres selon les ingrédients actifs.

2.4 Enrichissement du dataset

Afin de rendre les données plus complètes et utiles pour une éventuelle application ou analyse avancée, on a enrichi le dataset avec plusieurs colonnes supplémentaires :

- **`price`** : le prix du produit, extrait et converti au bon format,
- **`product_url`** : lien vers la page officielle du produit (source e-commerce),
- **`image_url`** : lien direct vers l'image du produit,
- **`type_peau`** : type de peau concerné (sèche, grasse, mixte, sensible, etc.),
- **`probleme_peau`** : problème ciblé (imperfections, sécheresse, acné…),
- **`age_min` / `age_max`** : tranche d'âge pour laquelle le produit est recommandé.

3. Architecture de l'application

L'architecture repose sur un modèle **client-serveur** où :

- Le **frontend** React s'occupe de l'interface utilisateur (saisie des préférences, affichage des produits).
- Le **backend** FastAPI centralise la logique métier, le traitement des requêtes et la communication avec le modèle ML.

- Le **modèle de machine learning**, intégré dans le backend, prédit les conseils d'utilisation à partir des ingrédients.

3.1 Composants du backend (FastAPI)

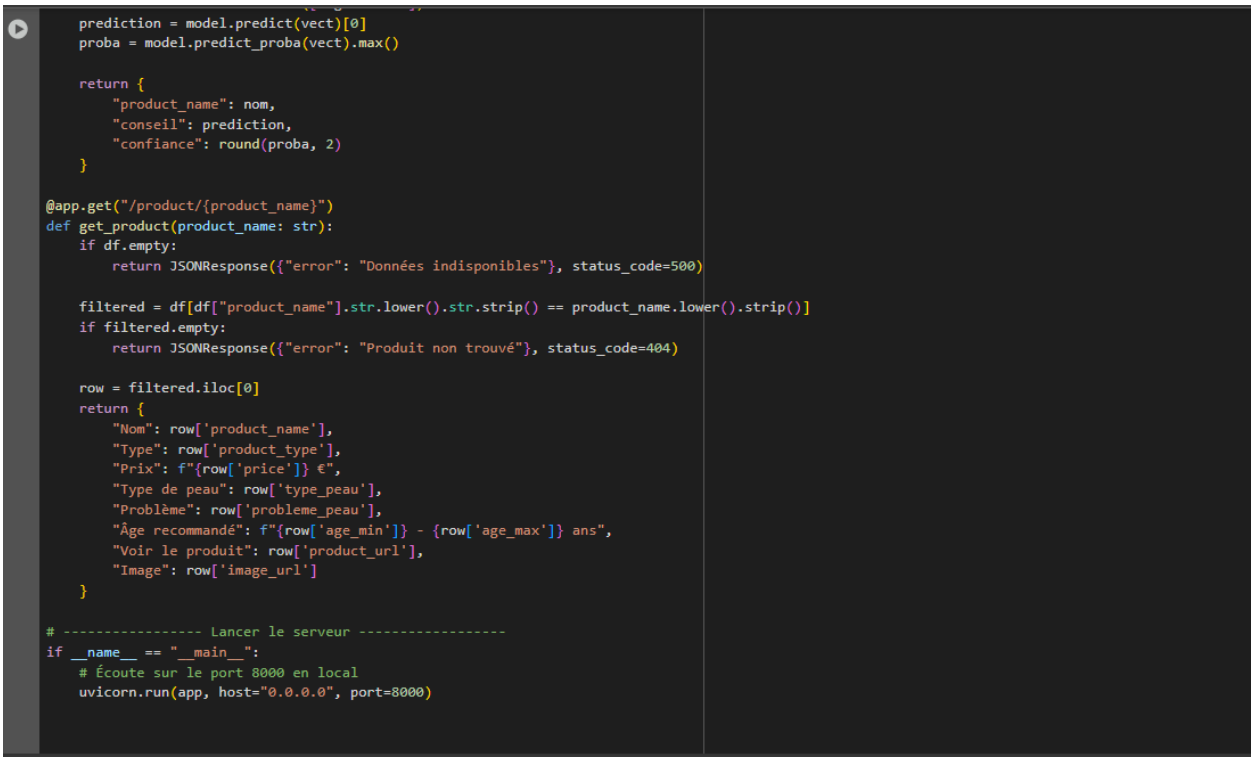
Structure principale

Le fichier principal est `main.py`, qui contient :

- **Chargement des données** : Lecture du fichier `skincare_cleaned.csv`, nettoyage et préparation.
- **Prétraitement ML** :
  - Utilisation de `TfidfVectorizer` pour vectoriser les ingrédients.
  - Entraînement d'un modèle `LogisticRegression` pour prédire des conseils d'utilisation.
- **Définition des modèles de données** avec `Pydantic` pour valider les requêtes.
- **Routes principales** :
  - `GET /` : test du serveur
  - `POST /recommend` : retourne les produits filtrés selon le type de peau, l'âge, le budget et les problèmes de peau.
  - `POST /conseil` : renvoie un conseil d'utilisation basé sur les ingrédients d'un produit.
  - `GET /product/{product_name}` : renvoie les détails d'un produit.

```python
# backend/main.py
from fastapi import FastAPI, HTTPException
from fastapi.responses import JSONResponse
from pydantic import BaseModel
from typing import List
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
import uvicorn

app = FastAPI()

# ---------------- Chargement des données ------------------
try:
    df = pd.read_csv("skincare_cleaned.csv")  # place ton CSV à la racine de backend/
    df["price"] = df["price"].astype(str).str.replace("£", "").astype(float)
    df["probleme_peau"] = df["probleme_peau"].fillna("").astype(str)
    df["clean_ingreds"] = df["clean_ingreds"].fillna("")
    df["conseil_utilisation"] = df["conseil_utilisation"].fillna("")
except Exception as e:
    df = pd.DataFrame()
    print("Erreur de chargement du CSV :", e)

# ---------------- Modèle ML ------------------
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df["clean_ingreds"])
y = df["conseil_utilisation"]
model = LogisticRegression(max_iter=1000)
model.fit(X, y)

# ---------------- Pydantic Models ------------------
class RecommendRequest(BaseModel):
    type_peau: str
    age: int
    budget: float
    probleme_peau: List[str] = []

class ConseilRequest(BaseModel):
    index: int
    clean_ingreds_list: List[str]
    noms_produits: List[str]

# ---------------- Routes ------------------
```

```python
@app.get("/")
def root():
    return {"message": "Backend GlowCare opérationnel ✅"}

@app.post("/recommend")
def recommend_products(req: RecommendRequest):
    filtered = df.copy()
    filtered = filtered[(filtered["type_peau"] == req.type_peau) | (filtered["type_peau"] == "tous")]
    filtered = filtered[(filtered["age_min"] <= req.age) & (filtered["age_max"] >= req.age)]
    filtered = filtered[filtered["price"] <= req.budget]

    if req.probleme_peau:
        filtered = filtered[filtered["probleme_peau"].apply(
            lambda x: any(p.lower() in x.lower() for p in req.probleme_peau)
        )]

    if filtered.empty:
        raise HTTPException(status_code=404, detail="Aucun produit trouvé.")

    produits = []
    for _, row in filtered.iterrows():
        if pd.notna(row["image_url"]):
            produits.append({
                "product_name": row["product_name"],
                "price": row["price"],
                "image_url": row["image_url"],
                "clean_ingreds": row["clean_ingreds"]
            })

    return {
        "nb_resultats": len(produits),
        "produits": produits
    }

@app.post("/conseil")
def get_conseil(req: ConseilRequest):
    index = req.index
    if index >= len(req.clean_ingreds_list):
        raise HTTPException(status_code=400, detail="Index invalide.")

    ingredients = req.clean_ingreds_list[index]
    nom = req.noms_produits[index]

    vect = vectorizer.transform([ingredients])
```



```
    prediction = model.predict(vect)[0]
    proba = model.predict_proba(vect).max()

    return {
        "product_name": nom,
        "conseil": prediction,
        "confiance": round(proba, 2)
    }

@app.get("/product/{product_name}")
def get_product(product_name: str):
    if df.empty:
        return JSONResponse({"error": "Données indisponibles"}, status_code=500)

    filtered = df[df["product_name"].str.lower().str.strip() == product_name.lower().strip()]
    if filtered.empty:
        return JSONResponse({"error": "Produit non trouvé"}, status_code=404)

    row = filtered.iloc[0]
    return {
        "Nom": row['product_name'],
        "Type": row['product_type'],
        "Prix": f"{row['price']} €",
        "Type de peau": row['type_peau'],
        "Problème": row['probleme_peau'],
        "Âge recommandé": f"{row['age_min']} - {row['age_max']} ans",
        "Voir le produit": row['product_url'],
        "Image": row['image_url']
    }

# ---------------- Lancer le serveur ----------------
if __name__ == "__main__":
    # Écoute sur le port 8000 en local
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

3.2 Composants du frontend (React)

L'interface utilisateur de GlowCare a été réalisée avec **React.js**, en suivant une architecture modulaire avec des composants dédiés à chaque fonctionnalité principale : `HomePage`, `RoutineForm`, et `PerformancePage`.

1. HomePage – Accueil & Présentation

Le composant `HomePage` a pour but d'introduire l'utilisateur à la philosophie de GlowCare:

2. RoutineForm – Formulaire de Recommandation

Le composant `RoutineForm` est le cœur de l'expérience utilisateur. Il permet de :

Collecter les informations utilisateur :

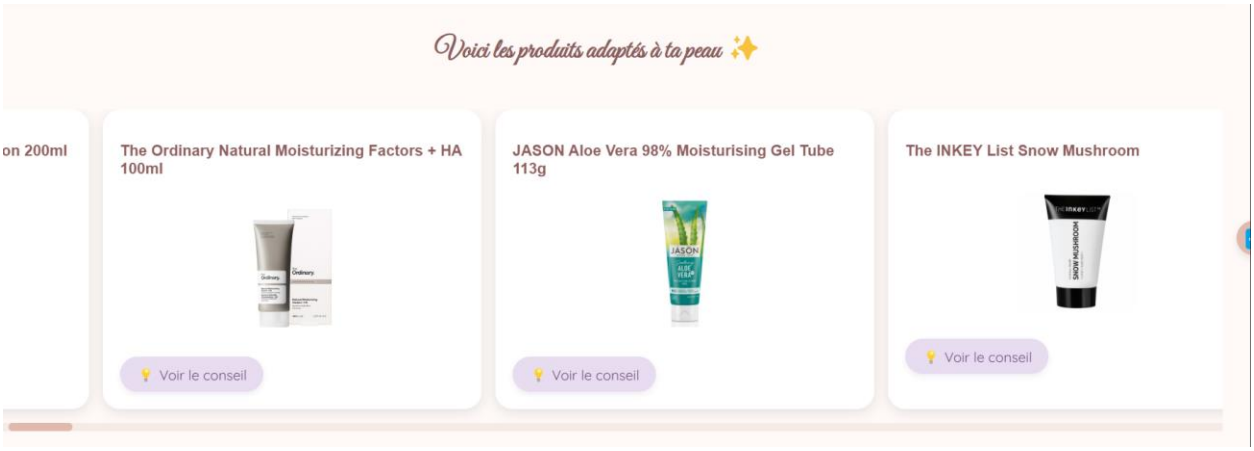
- Type de peau (`select`)
- Âge (`input number`)
- Budget (`input number`)
- Problèmes de peau (`checkboxes` multiples : acné, rides, rougeurs, etc.)

Interactions backend :

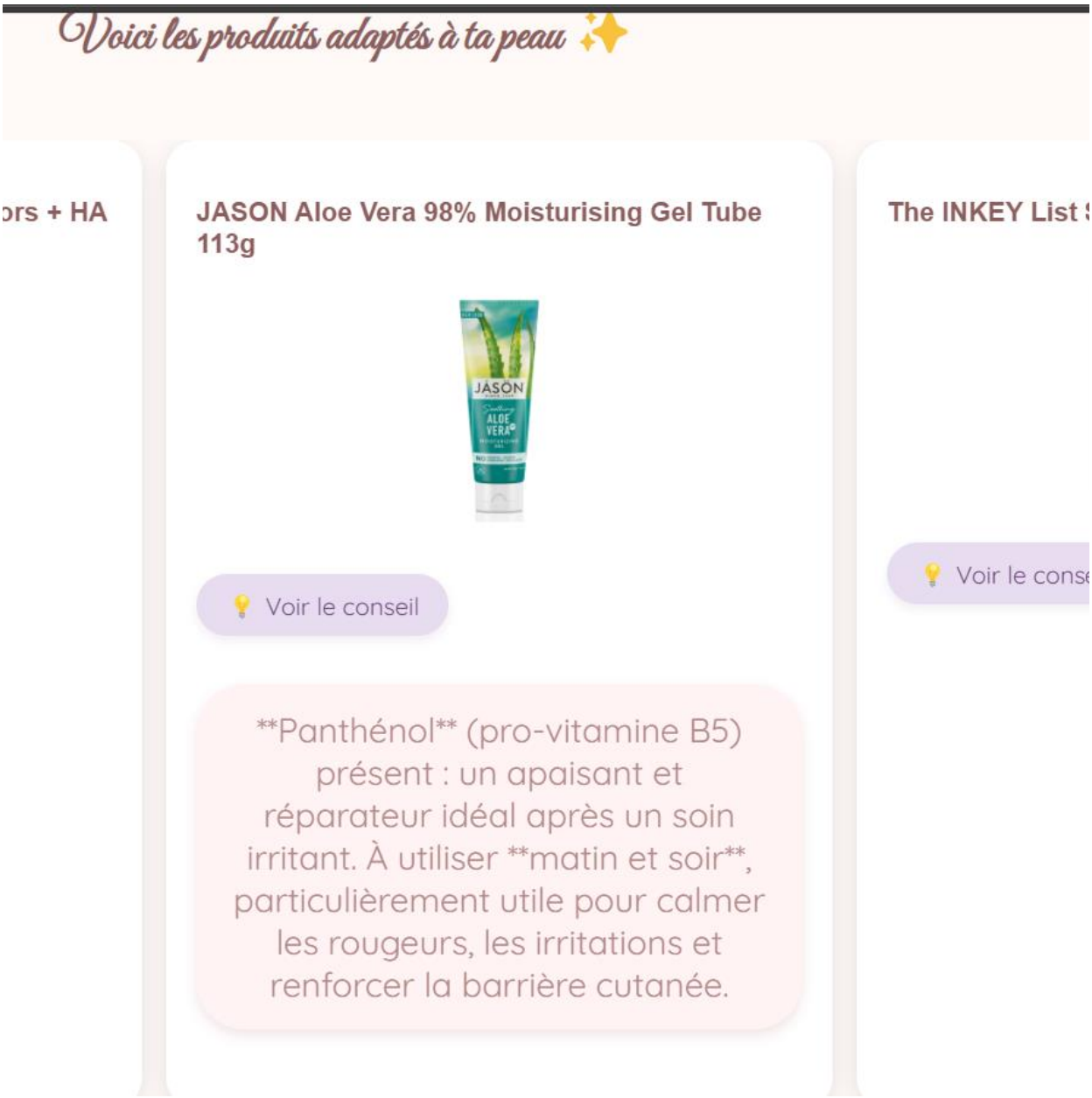
- **POST `/recommend`** : envoie les données utilisateur et récupère une liste de produits recommandés (avec image et nom).
- **POST `/conseil`** : permet d'obtenir un **conseil personnalisé** sur un produit sélectionné.

Interface utilisateur :

- **Carousel horizontal** : les produits recommandés sont affichés dans une interface défilante avec flèches gauche/droite.



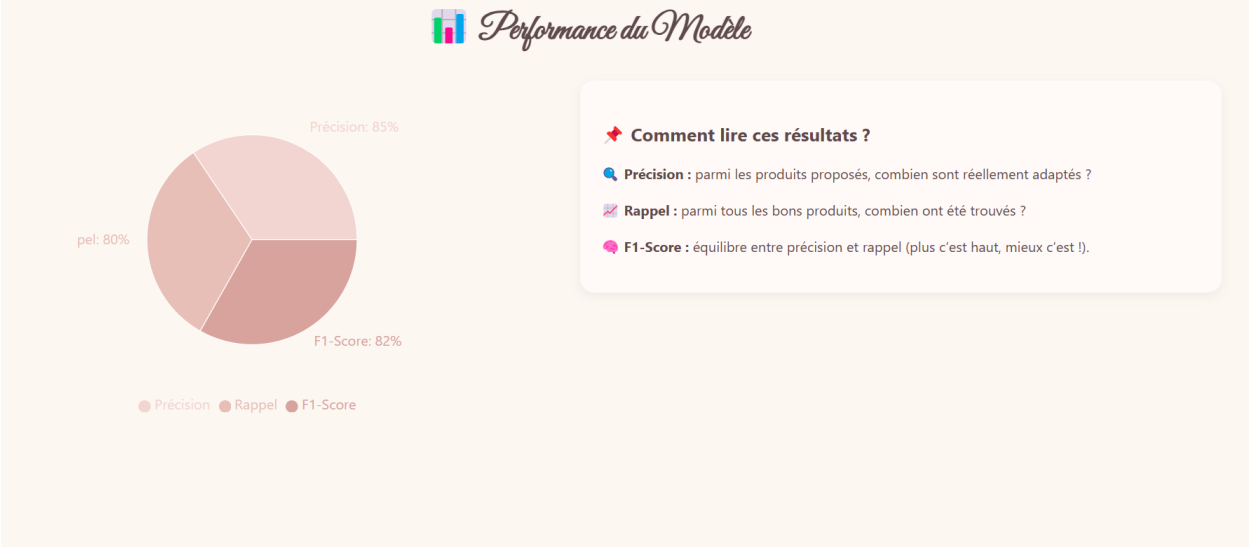
- **Affichage des conseils** : une fois un produit sélectionné, un message explicatif apparaît sous forme de citation.



3. PerformancePage – Visualisation des performances du modèle

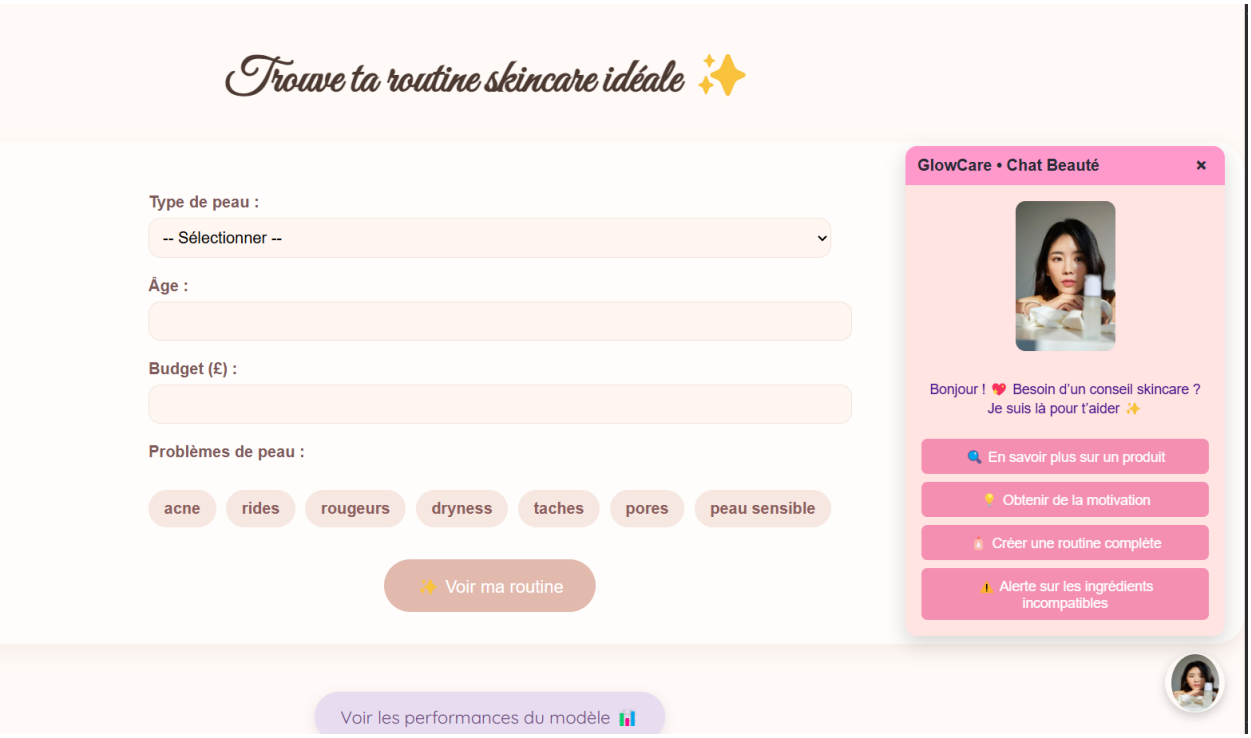
Le composant `PerformancePage` permet d'afficher les métriques de performance du modèle de recommandation via un **graphique circulaire** :

- **Bibliothèque utilisée** : `Recharts`



3.3 Description du Chatbot

Le chatbot **GlowCare** est une application interactive développée avec **React**, destinée à fournir des conseils personnalisés en soins de la peau (skincare). Il offre plusieurs fonctionnalités, comme la recherche d'informations sur des produits cosmétiques, la création de routines adaptées aux différents types de peau, l'envoi de messages de motivation, et des alertes sur les incompatibilités d'ingrédients actifs.



3-4 Comparaison des modules et justification du choix final

Pour ce projet, nous avons testé trois modules différents pour recommander les conseils d'utilisation, les précautions et la fréquence à partir des ingrédients. Chaque module utilise une méthode différente pour améliorer la qualité des prédictions.

Module 1 – Multi-label Logistic Regression simple :
Ce module utilise la régression logistique avec MultiOutputClassifier pour prédire plusieurs étiquettes (usage, précautions, fréquence) à partir des ingrédients. Il utilise une vectorisation TF-IDF simple.
Résultats : F1-score micro = 0.8581, F1-score macro = 0.4359, Exact Match Accuracy = 0.4868.
Il est correct mais pas très bon sur les classes peu fréquentes.

```
F1-score micro : 0.8581
F1-score macro : 0.4359
Hamming Loss : 0.0851
Exact Match Accuracy : 0.4868
F1-score micro : 0.8581
F1-score macro : 0.4359
Hamming Loss : 0.0851
Exact Match Accuracy : 0.4868
```

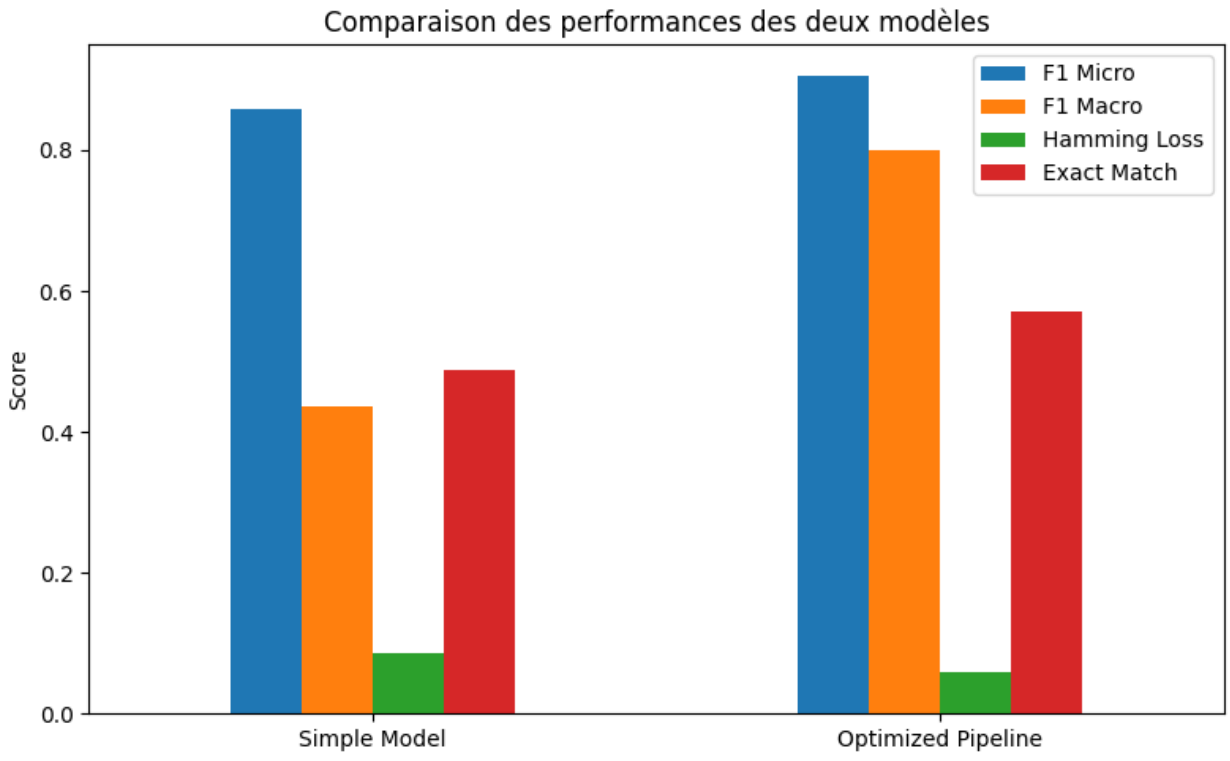
Module 2 – Pipeline amélioré avec GridSearch :

Ce module est une version avancée du premier. Il utilise un Pipeline avec TF-IDF, des *n-grams* (1,2), une pondération des classes (class_weight='balanced') et une recherche automatique du meilleur paramètre C avec GridSearchCV.

Résultats : F1-score micro = 0.9054, F1-score macro = 0.8009, Exact Match Accuracy = 0.5702.

Il est plus performant et plus robuste, surtout sur les classes rares.

```
F1-score micro      : 0.9054
F1-score macro      : 0.8009
Hamming Loss        : 0.0596
Exact Match Accuracy: 0.5702
```



Module 3 – Mono-label Logistic Regression sur les conseils :

Ce module est plus simple : il prédit uniquement un **conseil d'utilisation unique** à partir des ingrédients. Il utilise **LabelEncoder** pour encoder une seule étiquette cible (`conseil_utilisation`) et **LogisticRegression** pour la classification.

Ce modèle est simple mais donne des résultats très satisfaisants pour notre besoin principal.
Ses performances sont :

- F1-score (micro) : 0.8917
- F1-score (macro) : 0.7923
- Accuracy (Exact Match) : 0.8045
- Hamming Loss : 0.0487

Ces résultats montrent que ce modèle est précis, stable et suffisamment robuste, ce qui justifie notre choix final.

Conclusion : Après comparaison, nous avons choisi le Module 3, car il est plus performant pour notre objectif final : recommander un seul conseil clair et pertinent à partir des ingrédients. Il est également plus léger, plus rapide à exécuter et facile à intégrer dans une application.

4. Difficultés rencontrées

4.1 Données non disponibles directement

- Nous n'avons pas trouvé de dataset prêt à l'emploi répondant à notre besoin (produits skincare avec ingrédients et conseils).
- Pour cette raison, nous avons dû :
    - Chercher différentes sources de données (sites, blogs, e-commerces…),
    - Fusionner manuellement certaines informations,
    - Nettoyer en profondeur les données récupérées (colonnes incomplètes, mauvais formats, doublons…),
    - Ajouter nous-mêmes des champs utiles comme le prix, le type de peau, les problèmes cutanés, etc.

4.2 Choix du bon modèle de machine learning

- Nous avons testé **trois modules différents** pour prédire les conseils d'utilisation à partir des ingrédients.
- Ce processus a été complexe, car il fallait :
    - Préparer les données pour chaque module,
    - Adapter les types d'étiquettes (mono-label ou multi-label),
    - Évaluer les performances pour choisir le plus performant.
- Finalement, le **Module 3 (Logistic Regression Mono-label)** a été sélectionné, car il a donné les meilleurs résultats (accuracy = 0.80 environ, F1-score micro = 0.89).

4.3 Difficultés côté frontend (React)

- La gestion des états (state) avec React était compliquée au début, notamment pour l'affichage dynamique des produits recommandés.
- Intégrer des composants comme les filtres, les cartes de produits et les formulaires a demandé du temps.
- La communication avec l'API FastAPI (via Axios) a nécessité plusieurs ajustements.

4.4 Intégration du chatbot dans l'application

- L'intégration du chatbot a posé plusieurs problèmes :
    - D'abord, il fallait construire un chatbot simple mais utile (capable de répondre à des questions de base).
    - Ensuite, le connecter au frontend tout en gardant l'expérience fluide n'était pas facile.
    - Nous avons rencontré des soucis avec la gestion des messages, le style de la fenêtre du chatbot et le traitement des réponses automatiques.
    - Nous avons testé plusieurs solutions avant de réussir à l'intégrer correctement.

5. Conclusion

GlowCare est un projet ambitieux qui combine intelligence artificielle, traitement de données et interface web interactive pour offrir des recommandations personnalisées en skincare.
Malgré les différentes difficultés techniques rencontrées, notamment pour la collecte et

la préparation des données, la sélection du modèle de machine learning, et l'intégration du chatbot, nous avons réussi à développer une application fonctionnelle et utile.

Ce projet nous a permis de mettre en pratique plusieurs compétences techniques :
- Prétraitement des données
- Entraînement et évaluation de modèles ML
- Développement backend avec FastAPI
- Conception d'une interface réactive avec React
- Intégration d'un chatbot pour une meilleure expérience utilisateur