MovieMood

Plateforme Web IA de Recommandation de Films par Émotion

Équipe :
Gémima ONDELE POUROU | Fatoumata BAH | Hector KOMBOU

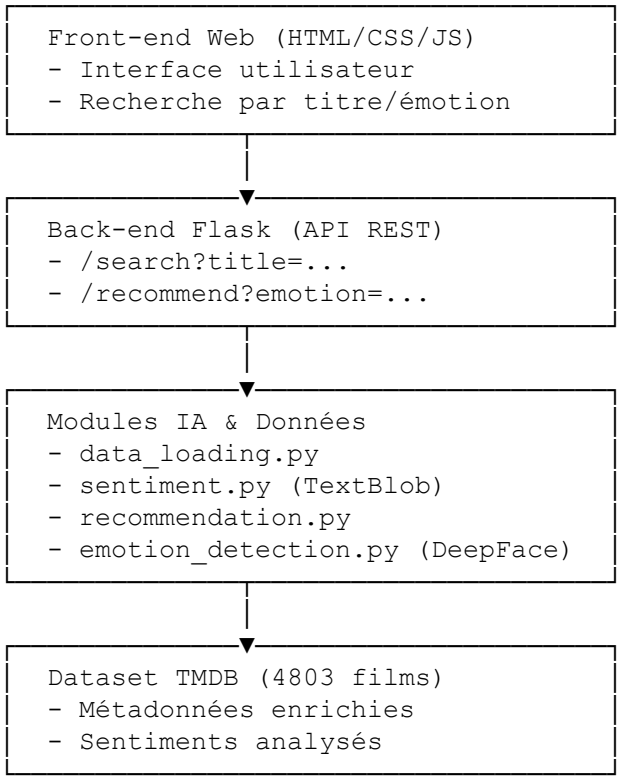
Projet Académique – M1 Data Engineering

Problématique & Vision

- **Le Défi :**
  - *"Comment choisir un film qui correspond à notre humeur du moment ?"*
- ## Notre Solution : 🎯
  - Plateforme web intelligente qui recommande des films basés sur :
    - • Votre émotion actuelle
    - • L'analyse de sentiments des films
    - • Un système de scoring avancé
- ## Impact : ✅
    - • Économise du temps de recherche
    - • Améliore l'expérience cinématographique
    - • Personnalisation par émotion

Architecture Technique

- **Vision Globale :**
-



Plan de Travail & Répartition

- **Phase 1 - Données & Analyse de Sentiments**
  - **Gémima ONDELE**
    - • Chargement dataset TMDB
    - • Analyse de sentiments avec TextBlob
    - • Enrichissement des données
- Phase 2 - Moteur de Recherche
  - **Hector KOMBOU**
    - • Recherche par titre
    - • Recommandations par similarité
    - • Recommandations par émotion
    - • Système de scoring
- Phase 3 - Interface Web
  - **Fatoumata BAH**
    - • Design et UX
    - • Pages HTML/CSS
    - • Intégration API
    - • Expérience utilisateur

PHASE 1 : Architecture du Projet

- 🧐 **PROMPT à l'IA :**
  - *"Nous voulons créer une plateforme web IA de recommandation de films.*
    Nous avons un dataset TMDB avec 5000 films. Comment structurer le projet ?"

- 💡 RÉPONSE DE L'IA :
  - ✅ Proposition d'architecture en 4 couches :
    - 1. Front-end web (HTML/CSS/JS)
    - 2. Back-end Python (Flask/FastAPI)
    - 3. Modules IA/Données
    - 4. Dataset TMDB

- ✅ Plan de travail avec répartition des tâches
  - ✅ Cahier des charges structuré

PHASE 1 : Code Généré

- 💻 **CODE RÉSULTANT :**
  -
```
# data_loading.py (Gémima)
def charger_dataframe(path_csv: str) -> pd.DataFrame:
    """Charge le fichier CSV TMDB."""
    df = pd.read_csv(path_csv, engine="python",
                     encoding="utf-8")
    return df

def construire_liste_films(df: pd.DataFrame):
    """Construit une liste de films avec genres."""
    films = []
    for _, row in df.iterrows():
        film = {
            "id": row["id"],
            "title": row["title"],
            "genres": parser_genres(row["genres"]),
            "overview": row["overview"],
            "vote_average": row["vote_average"],
        }
        films.append(film)
    return films
```

PHASE 1 : Analyse de Sentiments

- 🧐 **PROMPT :**
  - *"Comment analyser le sentiment des résumés de films ?"*
- 💡 RÉPONSE :
  - ✅ Utilisation de TextBlob
  - ✅ Fonction analyser_sentiment_texte()
  - ✅ Retourne : score [-1,1] + label

PHASE 1 : Code Analyse de Sentiments

- 💻 **CODE :**
  - ```
    # lib_projet.py (Gémima)
    from textblob import TextBlob

    def analyser_sentiment_texte(texte: str):
        """Analyse le sentiment avec TextBlob."""
        blob = TextBlob(texte)
        polarite = blob.sentiment.polarity

        if polarite > 0.1:
            label = "positif"
        elif polarite < -0.1:
            label = "negatif"
        else:
            label = "neutre"

        return float(polarite), label
    ```
    ```

PHASE 2 : Système de Scoring

- 🧐 **PROMPT :**
  - *"Comment créer un système de scoring pour réordonner les films ?"*
-

  💡 RÉPONSE :
  - ✅ Scoring combinant :
    - • Sentiment du film (normalisé 0-1)
    - • Note moyenne (normalisée 0-1)
    - • Pondérations ajustables

PHASE 2 : Code Système de Scoring

- 💻 **CODE :**
  - ```
    # lib_projet.py (Hector)
    def calculer_score_film(film: dict, emotion_user: str) -> float:
        """Calcule un score global pour un film."""
        sentiment_norm = normaliser_sentiment(
            film.get("sentiment_score", 0.0)
        )
        note_norm = normaliser_note(
            film.get("vote_average", 0.0)
        )

        w_sentiment = 0.6  # importance du sentiment
        w_note = 0.4       # importance de la note

        score = w_sentiment * sentiment_norm + w_note * note_norm
        return float(score)
    ```
    ```

PHASE 2 : Recommandations par Émotion

- 🤔 **PROMPT :**
  - *"Comment mapper les émotions aux genres de films ?"*
-

  💡 RÉPONSE :
  - ✅ Dictionnaire emotion_to_genres
  - ✅ Fonction recommander_par_emotion()
  - ✅ Filtre par genres → Score → Trie

PHASE 2 : Code Recommandations

- 💻 **CODE :**
  - ```
    # recommendation.py (Hector)
    emotion_to_genres = {
        "triste": ["Comedy", "Family", "Drama"],
        "stressé": ["Comedy", "Adventure", "Action"],
        "heureux": ["Romance", "Music", "Comedy"],
        # ... autres émotions
    }

    def recommander_par_emotion(emotion, films, n=20):
        genres_cibles = emotion_to_genres.get(emotion, [])
        candidats = []

        for film in films:
            if genres_cibles & film["genres"]:
                candidats.append(film)

        # Trier par note décroissante
        candidats.sort(key=lambda f: f["vote_average"],
                       reverse=True)
        return candidats[:n]
    ```
    ```

PHASE 3 : Interface Web

- 🤔 **PROMPT :**
  - *"Comment créer une interface web moderne ?"*
- 💡 RÉPONSE :
  - ✅ Flask + templates Jinja2
  - ✅ Page d'accueil avec recherche
  - ✅ Page de résultats avec cartes
  - ✅ Design moderne CSS

PHASE 3 : Code Interface Web

- 💻 **CODE :**
  - ```
    # app.py (Hector + Fatoumata)
    @app.route("/search")
    def search():
        titre = request.args.get("titre", "")
        emotion = request.args.get("emotion", "")

        resultats = []
        if titre:
            film = rechercher_par_titre(titre, films)
            if film:
                resultats.append(film)

        if emotion:
            resultats.extend(
                recommander_par_emotion(emotion, films, n=20)
            )

        return render_template("results.html",
                               films=resultats)
    ```
    ```

Amélioration : Vidéo de Fond

- 🤔 **PROMPT :**
  - *"La vidéo ne se lance pas sur Chrome/Edge. Comment corriger ?"*
- 💡 RÉPONSE :
  - ✅ Problème : Autoplay bloqué
  - ✅ Solution : YouTube Iframe API
  - ✅ Bouton pour activer le son

Code : Vidéo de Fond

- 💻 **CODE :**
  - ```
    // JavaScript (Fatoumata)
    function onYouTubeIframeAPIReady() {
        player = new YT.Player('background-video', {
            videoId: 'ZsJz2TJAPjw',
            playerVars: {
                'autoplay': 1,
                'mute': 1,
                'loop': 1,
            },
            events: {
                'onReady': function(event) {
                    event.target.playVideo();
                }
            }
        });
    }

    function toggleAudio() {
        if (isMuted) {
            player.unMute();
            player.setVolume(70);
        } else {
            player.mute();
        }
    }
    ```
    ```

Optimisation Performance

- 🤔 **PROMPT :**
  - *"L'application est lente. Comment optimiser ?"*
- 💡 RÉPONSE :
  - ✅ Système de cache JSON
  - ✅ Indicateurs de chargement visuels
  - ✅ Lazy loading des données

Code : Système de Cache

- 💻 **CODE :**
  - ```
    # cache_manager.py (Hector)
    def get_cached_films() -> List[Dict]:
        """Charge depuis le cache."""
        if CACHE_FILE.exists():
            with open(CACHE_FILE, "r") as f:
                films = json.load(f)
            return films
        return []

    def cache_films(films: List[Dict]) -> None:
        """Sauvegarde dans le cache."""
        with open(CACHE_FILE, "w") as f:
            json.dump(films, f, indent=2)
    ```
    ```

Détection d'Émotions Faciales

- 🤔 **PROMPT :**
  - *"Comment ajouter la détection d'émotions via webcam ?"*
- 💡 RÉPONSE :
  - ✅ DeepFace (modèle pré-entraîné)
  - ✅ OpenCV pour détection visage
  - ✅ Mapping émotions DeepFace → nos émotions

Code : Détection d'Émotions

- 💻 **CODE :**
  - ```
    # emotion_detection.py (Hector)
    from deepface import DeepFace
    import cv2

    def detecter_emotion_image(image_data: bytes):
        """Détecte l'émotion avec DeepFace."""
        img = cv2.imdecode(image_data, cv2.IMREAD_COLOR)

        # Détecter visage
        faces = face_cascade.detectMultiScale(gray)

        if len(faces) > 0:
            # Analyser avec DeepFace
            result = DeepFace.analyze(
                img_path=tmp_path,
                actions=['emotion']
            )
            emotion = EMOTION_MAPPING.get(
                result['dominant_emotion'],
                "neutre"
            )
            return {"emotion": emotion}
    ```
    ```

Évaluation & Notebooks

- 🤔 **PROMPT :**
  - *"Comment évaluer les performances de tous nos modèles ?"*
- 💡 RÉPONSE :
  - ✅ Notebook Jupyter complet
  - ✅ Analyse qualité dataset
  - ✅ Évaluation recommandations
  - ✅ Matrices de confusion
  - ✅ Visualisations

Résultats de l'Évaluation

- 📊 **MÉTRIQUES CLÉS :**
  - ✅ Dataset : 4,803 films (99.4% avec genres)
  - ✅ Recommandations : 8/8 émotions (100%)
  - ✅ Note moyenne recommandations : 8.50/10
  - ✅ Amélioration : +2.41 points vs moyenne globale

Résultats & Performances

- 📊 **PERFORMANCES :**
- ✅ Dataset :
  - • 4,803 films chargés
  - • 99.4% avec genres
  - • 98.7% avec notes valides
- ✅ Système de Recommandation :
  - • 8/8 émotions couvertes (100%)
  - • Note moyenne : 8.50/10
  - • +2.41 points vs moyenne globale

🎬 Démonstration Live

- **1. Recherche par titre**
  - Tapez 'The Matrix' → Voir les détails
- 2. Recommandation par émotion
  - Choisissez 'stressé' → 20 films adaptés
- 3. Détection faciale (bonus)
  - Activez webcam → IA détecte émotion
- 4. Vidéo de fond
  - Ambiance cinématographique + Son

Technologies Utilisées

- 🔧 **STACK TECHNIQUE :**
- Back-end : Python 3.12, Flask, Pandas
- IA & ML : TextBlob, DeepFace, TensorFlow, OpenCV
- Front-end : HTML5, CSS3, JavaScript, YouTube API
- Données : TMDB (4,803 films), Cache JSON
- Évaluation : Jupyter, Matplotlib, Seaborn

Défis Techniques Relevés

- 💪 **DÉFIS RÉSOLUS :**
  - 1. ✅ Autoplay vidéo → YouTube Iframe API
  - 2. ✅ Conflits dépendances → Gestion versions
  - 3. ✅ Performance → Système de cache
  - 4. ✅ Évaluation modèles → Notebook complet
  - 5. ✅ Erreurs Jupyter → Réorganisation imports

Améliorations Futures

- 🚀 **ROADMAP :**
- Court terme :
    - • Fine-tuning TextBlob
    - • Optimisation DeepFace
    - • Tests utilisateurs
- Moyen terme :
    - • Filtrage collaboratif
    - • Personnalisation utilisateur
- Long terme :
    - • Déploiement cloud
    - • Application mobile

Conclusion

- 🎯 **CE QUE NOUS AVONS RÉALISÉ :**
  - ✅ Plateforme web complète et fonctionnelle
  - ✅ Système de recommandation intelligent
  - ✅ Analyse de sentiments intégrée
  - ✅ Détection d'émotions faciales (bonus)
  - ✅ Interface utilisateur moderne
  - ✅ Évaluation complète des modèles
- 💡 VALEUR AJOUTÉE :
  - 🎬 Pour les utilisateurs : Gain de temps, découvertes personnalisées
  - 📊 Pour l'équipe : Maîtrise IA, expérience web, collaboration

Remerciements

- 🙏 **REMERCIEMENTS :**
- Merci à notre encadrante pour :
  - • Son accompagnement
  - • Ses conseils précieux
  - • Sa flexibilité
- 📚 RESSOURCES :
  - • Dataset : TMDB 5000 Movies
  - • Modèles : TextBlob, DeepFace
  - • Documentation : Flask, TensorFlow

Questions ?

- ❓ **QUESTIONS & RÉPONSES**

- Contact :
  - Gémima ONDELE POUROU
  - Fatoumata BAH
  - Hector KOMBOU

- Code source : Disponible sur GitHub
- https://github.com/GemimaOndele/Projet_moteur_de_recherche_de_films