- 🎬 MovieMood - Mapping Prompts → Réponses → Codes
  - Format pour PowerPoint/Canva : Chaque prompt avec sa réponse et son code
  - 📌 PROMPT 1 : Architecture du Projet
    - 🤔 PROMPT (Question)
    - 💡 RÉPONSE DE L'IA
    - 💻 CODE GÉNÉRÉ
  - 📌 PROMPT 2 : Analyse de Sentiments
    - 🤔 PROMPT (Question)
    - 💡 RÉPONSE DE L'IA
    - 💻 CODE GÉNÉRÉ
  - 📌 PROMPT 3 : Système de Scoring
    - 🤔 PROMPT (Question)
    - 💡 RÉPONSE DE L'IA
    - 💻 CODE GÉNÉRÉ
  - 📌 PROMPT 4 : Recommandations par Émotion
    - 🤔 PROMPT (Question)
    - 💡 RÉPONSE DE L'IA
    - 💻 CODE GÉNÉRÉ
  - 📌 PROMPT 5 : Interface Web
    - 🤔 PROMPT (Question)
    - 💡 RÉPONSE DE L'IA
    - 💻 CODE GÉNÉRÉ
  - 📌 PROMPT 6 : Vidéo de Fond (Autoplay)
    - 🤔 PROMPT (Question)
    - 💡 RÉPONSE DE L'IA
    - 💻 CODE GÉNÉRÉ
  - 📌 PROMPT 7 : Optimisation Performance
    - 🤔 PROMPT (Question)
    - 💡 RÉPONSE DE L'IA
    - 💻 CODE GÉNÉRÉ
  - 📌 PROMPT 8 : Détection d'Émotions Faciales
    - 🤔 PROMPT (Question)
    - 💡 RÉPONSE DE L'IA
    - 💻 CODE GÉNÉRÉ
  - 📌 PROMPT 9 : Évaluation & Notebooks
    - 🤔 PROMPT (Question)

🎬 MovieMood - Mapping Prompts → Réponses → Codes

Format pour PowerPoint/Canva : Chaque prompt avec sa réponse et son code

---

📌 PROMPT 1 : Architecture du Projet

🤔 PROMPT (Question)

```
"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
```

💻 CODE GÉNÉRÉ

Fichier : `code/data_loading.py`

```python
# data_loading.py
# ===========================
# Module de Gémima : chargement et préparation des données films

import pandas as pd
from lib_projet import parser_genres, get_main_genre, extraire_annee

def charger_dataframe(path_csv: str) -> pd.DataFrame:
    """
    Charge le fichier CSV TMDB en DataFrame pandas.
    """
    df = pd.read_csv(
        path_csv,
        engine="python",
        encoding="utf-8",
        on_bad_lines="skip",
    )
    return df

def construire_liste_films(df: pd.DataFrame):
    """
    Construit une liste de films avec genres parsés.
    """
    films = []
    for _, row in df.iterrows():
        film = {
            "id": int(row.get("id", 0)),
            "title": str(row.get("title", "")),
            "genres": parser_genres(row.get("genres", "[]")),
            "overview": str(row.get("overview", "")),
            "vote_average": float(row.get("vote_average", 0.0)),
            "popularity": float(row.get("popularity", 0.0)),
            "release_year": extraire_annee(row.get("release_date", "")),
            # ...
        }
        films.append(film)
    return films

def charger_films_prepares(path_csv: str):
    """
    Étape 1 pour Gémima :
    - charge le CSV brut TMDB
    - construit une liste de films avec genres, overview, vote_average, etc.
    """
    df = charger_dataframe(path_csv)
    films = construire_liste_films(df)
    return films
```

📌 PROMPT 2 : Analyse de Sentiments

🤔 PROMPT (Question)

```
"Comment analyser le sentiment des résumés de films pour enrichir nos données ?"
```

💡 RÉPONSE DE L'IA

```
✅ Utilisation de TextBlob (basé sur NLTK)
✅ Fonction analyser_sentiment_texte() qui retourne :
   - score de polarité [-1, 1]
   - label (positif/neutre/négatif)
✅ Intégration dans le pipeline de traitement
```

💻 CODE GÉNÉRÉ

Fichier : `code/lib_projet.py`

```python
from textblob import TextBlob

def analyser_sentiment_texte(texte: str):
    """
    Analyse le sentiment d'un texte en utilisant TextBlob.

    Retourne :
        - score : float entre -1 (très négatif) et 1 (très positif)
        - label : "negatif", "neutre" ou "positif"
    """
    if not isinstance(texte, str) or texte.strip() == "":
        return 0.0, "neutre"

    blob = TextBlob(texte)
    polarite = blob.sentiment.polarity  # entre -1 et 1

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

    return float(polarite), label
```

Fichier : `code/sentiment.py`

```python
from lib_projet import analyser_sentiment_texte, normaliser_sentiment

def ajouter_sentiment_aux_films(liste_films: List[Dict]) -> List[Dict]:
    """
    Parcourt la liste des films et ajoute :
        - sentiment_score (float [-1, 1])
        - sentiment_label ("positif", "neutre", "negatif")
        - sentiment_score_norm (float [0, 1])
    en se basant sur le champ "overview".
    """
    for film in liste_films:
        overview = film.get("overview", "")
        score, label = analyser_sentiment_texte(overview)
        film["sentiment_score"] = score
        film["sentiment_label"] = label
        film["sentiment_score_norm"] = normaliser_sentiment(score)
    return liste_films
```

📌 PROMPT 3 : Système de Scoring

🤔 PROMPT (Question)

```
"Comment créer un système de scoring pour réordonner les films selon l'émotion
de l'utilisateur ?"
```

💡 RÉPONSE DE L'IA

```
✅ Système de scoring combinant :
   - Sentiment du film (normalisé 0-1)
   - Note moyenne (normalisé 0-1)
   - Pondérations ajustables (w_sentiment, w_note)
✅ Formule : score = w_sentiment * sentiment_norm + w_note * note_norm
```

💻 CODE GÉNÉRÉ

Fichier : `code/lib_projet.py`

```python
def normaliser_sentiment(score: float) -> float:
    """Normalise un score de sentiment [-1, 1] en [0, 1]."""
    if score < -1:
        score = -1
    if score > 1:
        score = 1
    return (score + 1.0) / 2.0

def normaliser_note(note: float) -> float:
    """Normalise une note [0, 10] en [0, 1]."""
    if note < 0:
        note = 0
    if note > 10:
        note = 10
    return note / 10.0

def calculer_score_film(film: dict, emotion_user: str) -> float:
    """
    Calcule un score global pour un film en fonction :
        - du sentiment du texte (overview)
        - de la note moyenne du film
    """
    sentiment_score = film.get("sentiment_score", 0.0)
    sentiment_norm = normaliser_sentiment(sentiment_score)

    note = film.get("vote_average", 0.0)
    note_norm = normaliser_note(note)

    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)
```

📌 PROMPT 4 : Recommandations par Émotion

🤔 PROMPT (Question)

```
"Comment mapper les émotions aux genres de films pour les recommandations ?"
```

💡 RÉPONSE DE L'IA

✅ Création d'un dictionnaire emotion_to_genres
✅ Implémentation de recommander_par_emotion() qui :
1. Filtre par genres cibles
2. Calcule le score pour chaque film
3. Trie par note décroissante (vote_average)
✅ Cas spécial pour "surprise" (tous les films)

💻 CODE GÉNÉRÉ

Fichier : code/lib_projet.py

```
# Mapping émotion -> genres (enrichi pour avoir plus de recommandations)
emotion_to_genres = {
    "triste":      ["Comedy", "Family", "Drama", "Romance", "Animation"],
    "stressé":     ["Comedy", "Adventure", "Action", "Animation", "Family"],
    "heureux":     ["Romance", "Music", "Comedy", "Animation", "Family"],
    "nostalgique": ["Drama", "History", "Romance", "Music", "Family", "War"],
    "ennuyé":      ["Action", "Thriller", "Sci-Fi", "Adventure", "Crime",
"Mystery"],
    "colere":      ["Action", "Thriller", "Crime", "War", "Drama", "History"],
    "peur":        ["Horror", "Thriller", "Mystery", "Crime", "Sci-Fi"],
    "surprise":    [],  # Cas spécial : retourne tous les films, triés par note
}
```

Fichier : code/recommendation.py

```
def recommander_par_emotion(emotion: str, films: List[Film], n: int = 20) ->
List[Film]:
    """Filtre les films par genres liés à l'émotion et applique un scoring
simple."""
    if not emotion:
        return []

    emotion_lower = emotion.lower()

    # Cas spécial pour 'surprise' : retourner les meilleurs films notés
    if emotion_lower == "surprise":
        candidats = []
        for film in films:
            if film.get("vote_average", 0.0) > 0:
                film_copy = dict(film)
                film_copy["score_emotion"] = film.get("vote_average", 0.0)
                candidats.append(film_copy)

        # Trier par note décroissante
        candidats.sort(key=lambda f: f.get("vote_average", 0.0), reverse=True)
        return candidats[:n]
```

```
    genres_cibles = emotion_to_genres.get(emotion_lower, [])
    genres_cibles_set = set(genres_cibles)
    candidats = []

    # Filtrer les films correspondant aux genres de l'émotion
    for film in films:
        film_genres = set(film.get("genres", []))
        if genres_cibles_set.intersection(film_genres):
            film_copy = dict(film)
            film_copy["score_emotion"] = calculer_score_film(film_copy, emotion)
            candidats.append(film_copy)

    # Trier uniquement par note décroissante
    candidats.sort(key=lambda f: f.get("vote_average", 0.0), reverse=True)
    return candidats[:n]
```

📌 PROMPT 5 : Interface Web

🤔 PROMPT (Question)

```
"Comment créer une interface web moderne et intuitive pour notre plateforme ?"
```

💡 RÉPONSE DE L'IA

```
✅ Structure Flask avec templates Jinja2
✅ Pages principales :
   - Page d'accueil avec recherche
   - Page de résultats avec cartes de films
✅ Design moderne avec CSS personnalisé
✅ Intégration API Flask
```

💻 CODE GÉNÉRÉ

Fichier : `code/templates/index.html`

```
<!DOCTYPE html>
<html lang="fr">
```

```
<head>
    <meta charset="UTF-8">
    <title>MovieMood - Recommandation de Films</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css')
}}">
</head>
<body>
    <div class="container">
        <h1>🎬 MovieMood</h1>
        <p>Découvrez des films selon votre humeur</p>

        <form action="/search" method="GET" class="search-form" id="search-form">
            <input type="text" name="titre" placeholder="Titre de film..."
                   value="{{ titre or '' }}">

            <select name="emotion">
                <option value="">Choisissez une émotion</option>
                <option value="triste" {% if emotion == 'triste' %}selected{% endif
%}>😢 Triste</option>
                <option value="heureux" {% if emotion == 'heureux' %}selected{%
endif %}>😊 Heureux</option>
                <option value="stressé" {% if emotion == 'stressé' %}selected{%
endif %}>😰 Stressé</option>
                <!-- ... autres émotions ... -->
            </select>

            <button type="submit">🔍 Chercher / Recommander</button>
        </form>
    </div>
</body>
</html>
```

Fichier : code/app.py

```
from flask import Flask, render_template, request
from recommendation import rechercher_par_titre, recommander_par_emotion

@app.route("/search")
def search():
    titre = request.args.get("titre", "").strip()
    emotion = request.args.get("emotion", "").strip().lower()

    resultats = []

    if titre:
        film = rechercher_par_titre(titre, catalogue_films)
        if film:
            resultats.append(film)

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

    # Dédupliquer
    resultats = _dedupe_films(resultats)
```

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

📌 PROMPT 6 : Vidéo de Fond (Autoplay)

🤔 PROMPT (Question)

```
"La vidéo de fond ne se lance pas automatiquement sur Chrome et Edge.
Comment corriger ?"
```

💡 RÉPONSE DE L'IA

```
✅ Problème : Autoplay bloqué par les navigateurs modernes
✅ Solution : Utiliser YouTube Iframe API pour contrôler la lecture
✅ Ajout d'un bouton pour activer le son
✅ Fallback si l'API ne charge pas
```

💻 CODE GÉNÉRÉ

Fichier : `code/templates/form.html`

```
<div id="background-video" class="video-container"></div>
<iframe id="background-video-fallback" class="video-fallback"
        src="https://www.youtube.com/embed/ZsJz2TJAPjw?
autoplay=1&mute=1&loop=1&playlist=ZsJz2TJAPjw&controls=0&playsinline=1"
        title="Vidéo de fond" style="display: none;"></iframe>

<button id="audio-toggle" class="audio-toggle-btn" title="Activer le son">
    <span id="audio-icon">🔇</span>
</button>
```

```
<script>
var tag = document.createElement('script');
tag.src = "https://www.youtube.com/iframe_api";
var firstScriptTag = document.getElementsByTagName('script')[0];
firstScriptTag.parentNode.insertBefore(tag, firstScriptTag);

var player;
var isMuted = true;

function onYouTubeIframeAPIReady() {
    player = new YT.Player('background-video', {
        videoId: 'ZsJz2TJAPjw',
        playerVars: {
            'autoplay': 1,
            'mute': 1,
            'loop': 1,
            'playlist': 'ZsJz2TJAPjw',
            'controls': 0,
            'playsinline': 1,
        },
        events: {
            'onReady': function(event) {
                event.target.playVideo();
            }
        }
    });
}

function toggleAudio() {
    if (!player) return;

    if (isMuted) {
        player.unMute();
        player.setVolume(70);
        isMuted = false;
        document.getElementById('audio-icon').textContent = '🔊';
    } else {
        player.mute();
        isMuted = true;
        document.getElementById('audio-icon').textContent = '🔇';
    }
}
</script>
```

📌 PROMPT 7 : Optimisation Performance

🤔 PROMPT (Question)

"L'application est lente au démarrage. Comment optimiser les performances ?"

💡 RÉPONSE DE L'IA

✅ Mise en place d'un système de cache
✅ Ajout d'indicateurs de chargement visuels
✅ Optimisation du chargement des données
✅ Lazy loading des films enrichis

💻 CODE GÉNÉRÉ

Fichier : code/cache_manager.py

```
import json
from pathlib import Path
from typing import List, Dict

CACHE_FILE = Path(__file__).resolve().parent.parent / "data" /
"enrichment_cache.json"

def get_cached_films() -> List[Dict]:
    """Charge les films enrichis depuis le cache."""
    if CACHE_FILE.exists():
        try:
            with open(CACHE_FILE, "r", encoding="utf-8") as f:
                films = json.load(f)
            logger.info(f"✅ {len(films)} films chargés depuis le cache.")
            return films
        except Exception as e:
            logger.warning(f"⚠️ Erreur lors du chargement du cache: {e}")
    return []

def cache_films(films: List[Dict]) -> None:
    """Sauvegarde les films enrichis dans le cache."""
    try:
        CACHE_FILE.parent.mkdir(parents=True, exist_ok=True)
        with open(CACHE_FILE, "w", encoding="utf-8") as f:
            json.dump(films, f, ensure_ascii=False, indent=2)
        logger.info(f"💾 {len(films)} films sauvegardés dans le cache.")
    except Exception as e:
        logger.warning(f"⚠️ Erreur lors de la sauvegarde du cache: {e}")
```

Fichier : code/static/css/loading.css

```css
#loading-overlay {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(0, 0, 0, 0.8);
    display: flex;
    justify-content: center;
    align-items: center;
    z-index: 9999;
    color: white;
}

.spinner {
    border: 8px solid rgba(255, 255, 255, 0.3);
    border-top: 8px solid var(--accent);
    border-radius: 50%;
    width: 60px;
    height: 60px;
    animation: spin 1s linear infinite;
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}
```

📌 PROMPT 8 : Détection d'Émotions Faciales

🤔 PROMPT (Question)

```
"Comment ajouter la détection d'émotions faciales via webcam ?"
```

💡 RÉPONSE DE L'IA

```
✅ Utilisation de DeepFace (modèle pré-entraîné TensorFlow)
✅ Détection de visage avec OpenCV Haar Cascade
```

✅ Mapping des émotions DeepFace vers nos émotions
✅ Gestion de la qualité d'image

💻 CODE GÉNÉRÉ

Fichier : `code/emotion_detection.py`

```
from deepface import DeepFace
import cv2
import numpy as np

EMOTION_MAPPING = {
    "happy": "heureux",
    "sad": "triste",
    "angry": "colere",
    "fear": "peur",
    "surprise": "surprise",
    "neutral": "neutre",
    "disgust": "ennuyé",
}

def detecter_emotion_image(image_data: bytes) -> Dict[str, any]:
    """
    Détecte l'émotion à partir d'une image avec DeepFace.
    """
    # Convertir les bytes en image OpenCV
    nparr = np.frombuffer(image_data, np.uint8)
    img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

    # Détecter le visage
    face_cascade = cv2.CascadeClassifier(
        cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'
    )
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.1, 4, minSize=(50, 50))

    if len(faces) > 0:
        # Extraire la région du visage
        x, y, w, h = max(faces, key=lambda f: f[2] * f[3])
        face_roi = img[y:y+h, x:x+w]

        # Analyser avec DeepFace
        result = DeepFace.analyze(
            img_path=tmp_path,
            actions=['emotion'],
            enforce_detection=False,
            silent=True
        )

        # Extraire l'émotion dominante
        emotions = result.get('emotion', {})
        emotion_key, score = max(emotions.items(), key=lambda x: x[1])
```

```
            confidence = float(score) / 100.0

            emotion_detected = EMOTION_MAPPING.get(str(emotion_key).lower(), "neutre")

            return {
                "emotion": emotion_detected,
                "face_bbox": (int(x), int(y), int(w), int(h)),
                "confidence": confidence
            }

    return {"emotion": None, "confidence": 0.0}
```

📌 PROMPT 9 : Évaluation & Notebooks

🤔 PROMPT (Question)

```
"Comment évaluer les performances de tous nos modèles et la qualité du dataset ?"
```

💡 RÉPONSE DE L'IA

```
✅ Création d'un notebook Jupyter complet avec :
   - Analyse de qualité du dataset
   - Évaluation du système de recommandation
   - Matrices de confusion
   - Visualisations (distributions, genres, etc.)
✅ Métriques de performance détaillées
```

💻 CODE GÉNÉRÉ

Fichier : `notebooks/evaluation_notebook.ipynb`

```
# Analyse de la complétude des données
print("📊 ANALYSE DE LA QUALITÉ DU DATASET")
print(f"Nombre total de films : {len(df)}")

# Valeurs manquantes
missing = df.isnull().sum()
print("Valeurs manquantes par colonne :")
```

```
print(missing[missing > 0])

# Statistiques descriptives
print("\n📈 Statistiques descriptives :")
print(df.describe())

# Distribution des genres
all_genres = []
for genres_list in df['genres']:
    if isinstance(genres_list, list):
        all_genres.extend(genres_list)

genre_counts = Counter(all_genres)
genre_df = pd.DataFrame(list(genre_counts.items()),
                        columns=['Genre', 'Nombre de films'])
genre_df = genre_df.sort_values('Nombre de films', ascending=False)

# Visualisation
plt.figure(figsize=(12, 8))
top_genres = genre_df.head(15)
plt.barh(top_genres['Genre'], top_genres['Nombre de films'])
plt.xlabel('Nombre de films')
plt.title('Top 15 Genres les Plus Représentés')
plt.gca().invert_yaxis()
plt.show()

# Évaluation des recommandations
for emotion in emotion_to_genres.keys():
    recommendations = recommander_par_emotion_simple(emotion, films, n=20)
    avg_rating = np.mean([f.get("vote_average", 0) for f in recommendations])
    print(f"{emotion}: {len(recommendations)} recommandations, "
          f"note moyenne: {avg_rating:.2f}")

# Matrice de confusion
confusion_matrix = []
for emotion in emotion_to_genres.keys():
    recommendations = recommander_par_emotion_simple(emotion, films, n=20)
    genre_counts = Counter()
    for rec in recommendations:
        genre_counts.update(rec.get("genres", []))
    confusion_matrix.append([genre_counts.get(g, 0) for g in all_unique_genres])

sns.heatmap(confusion_matrix, annot=True, fmt='d', cmap='YlOrRd',
            xticklabels=all_unique_genres,
            yticklabels=emotion_to_genres.keys())
plt.title('Matrice de Confusion : Distribution des Genres par Émotion')
plt.show()
```

📊 RÉSUMÉ DES PROMPTS

#	Prompt	Réponse	Fichiers Créés/Modifiés	Responsable
1	Architecture projet	4 couches + plan de travail	`data_loading.py`	Gémima
2	Analyse de sentiments	TextBlob intégré	`lib_projet.py`, `sentiment.py`	Gémima
3	Système de scoring	Formule combinée	`lib_projet.py`	Hector
4	Recommandations émotion	Mapping + algo	`recommendation.py`	Hector
5	Interface web	Flask + templates	`app.py`, `templates/`	Fatoumata
6	Vidéo autoplay	YouTube API	`form.html`	Fatoumata
7	Optimisation	Cache + loading	`cache_manager.py`, `loading.css`	Hector
8	Détection faciale	DeepFace	`emotion_detection.py`	Hector
9	Évaluation	Notebook complet	`evaluation_notebook.ipynb`	Tous

🎯 COMMENT UTILISER CE DOCUMENT

1. **Pour PowerPoint/Canva** :
   - Copiez chaque section (Prompt → Réponse → Code) dans un slide
   - Ajoutez des visuels (captures d'écran, schémas)
   - Utilisez des couleurs cohérentes
2. **Format Recommandé** :
   - Slide gauche : Prompt (question)
   - Slide droite : Réponse + Code (résultat)

3. **Métriques à Ajouter** :

    - Lignes de code générées
    - Temps de développement
    - Performance des modèles