🎬 MovieMood - Présentation du Projet

Plateforme Web IA de Recommandation de Films par Émotion

Équipe :

- Gémima ONDELE POUROU
- Fatoumata BAH
- Hector KOMBOU

📋 STRUCTURE DE LA PRÉSENTATION (SLIDES)

SLIDE 1 : Page de Titre

```
🎬 MovieMood
Plateforme Web IA de Recommandation de Films par Émotion

Équipe :
Gémima ONDELE POUROU | Fatoumata BAH | Hector KOMBOU

Projet Académique - Data Engineering
```

SLIDE 2 : 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 :

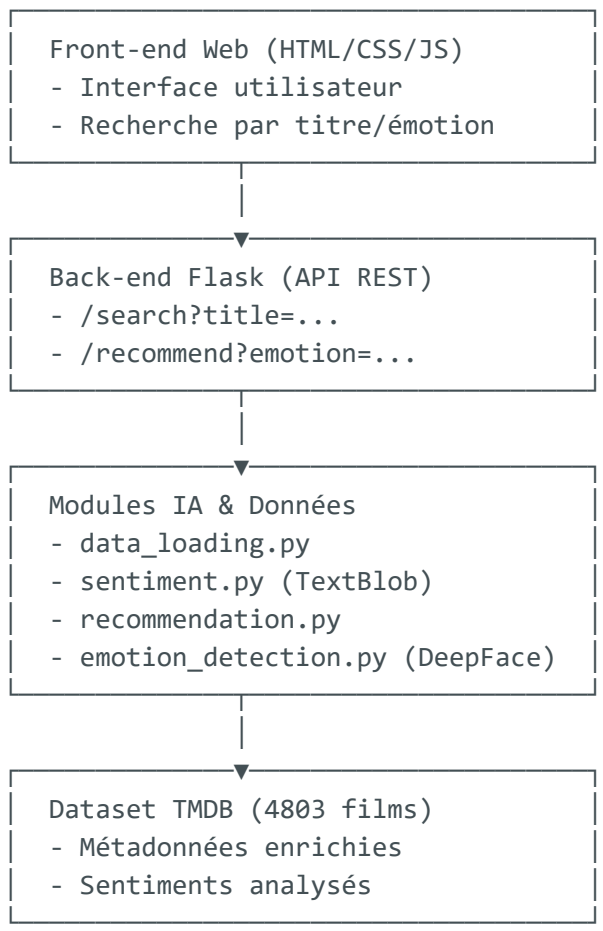
- Votre émotion actuelle
- L'analyse de sentiments des films
- Un système de scoring avancé

Impact : ✅ Économise du temps de recherche ✅ Améliore l'expérience cinématographique ✅ Personnalisation par émotion

---

SLIDE 3 : Architecture Technique

Vision Globale :



SLIDE 4 : 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

📝 DÉVELOPPEMENT PAR PHASES (PROMPTS → RÉPONSES → CODE)

SLIDE 5-6 : PHASE 1 - Prompt Initial

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.
```

CODE RÉSULTANT :

```python
# data_loading.py (Gémima)
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")
    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": row["id"],
```

```
            "title": row["title"],
            "genres": parser_genres(row["genres"]),
            "overview": row["overview"],
            "vote_average": row["vote_average"],
            # ...
        }
        films.append(film)
    return films
```

SLIDE 7-8 : PHASE 1 - Analyse de Sentiments

PROMPT à l'IA :

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

RÉPONSE de l'IA :

```
Utilisation de TextBlob pour l'analyse de sentiments.
Fonction analyser_sentiment_texte() qui retourne :
- score de polarité [-1, 1]
- label (positif/neutre/négatif)
```

CODE RÉSULTANT :

```
# lib_projet.py (Gémima)
from textblob import TextBlob

def analyser_sentiment_texte(texte: str):
    """Analyse le sentiment d'un texte avec TextBlob."""
    if not texte 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

# sentiment.py
```

```python
def ajouter_sentiment_aux_films(liste_films: List[Dict]) -> List[Dict]:
    """Ajoute sentiment_score et sentiment_label à chaque film."""
    for film in liste_films:
        overview = film.get("overview", "")
        score, label = analyser_sentiment_texte(overview)
        film["sentiment_score"] = score
        film["sentiment_label"] = label
    return liste_films
```

SLIDE 9-10 : PHASE 2 - Système de Scoring

PROMPT à l'IA :

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

RÉPONSE de l'IA :

```
Proposition d'un système de scoring combinant :
- Sentiment du film (normalisé 0-1)
- Note moyenne (normalisé 0-1)
- Pondérations ajustables
```

CODE RÉSULTANT :

```python
# lib_projet.py (Hector)
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."""
    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)
```

SLIDE 11-12 : PHASE 2 - Recommandations par Émotion

PROMPT à l'IA :

> "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 score décroissant
```

CODE RÉSULTANT :

```
# lib_projet.py
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":    [],  # Tous les films triés par note
}

# recommendation.py (Hector)
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."""
```

```
    if not emotion:
        return []

    emotion_lower = emotion.lower()

    # Cas spécial pour 'surprise'
    if emotion_lower == "surprise":
        candidats = [dict(f) for f in films if f.get("vote_average", 0.0) > 0]
        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 = []

    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]
```

SLIDE 13-14 : PHASE 3 - Interface Web

PROMPT à l'IA :

> "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
```

CODE RÉSULTANT :

```
<!-- templates/index.html (Fatoumata) -->
<!DOCTYPE html>
<html>
<head>
    <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">
            <input type="text" name="titre" placeholder="Titre de film...">

            <select name="emotion">
                <option value="">Choisissez une émotion</option>
                <option value="triste">😢 Triste</option>
                <option value="heureux">😊 Heureux</option>
                <option value="stressé">😰 Stressé</option>
                <!-- ... -->
            </select>

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

```
# app.py (Hector + Fatoumata)
from flask import Flask, render_template, request

@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))

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

SLIDE 15-16 : Amélioration - Vidéo de Fond

PROMPT à l'IA :

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

RÉPONSE de l'IA :

```
Problème d'autoplay dans les navigateurs modernes.
Solution : Utiliser YouTube Iframe API pour contrôler la lecture.
Ajout d'un bouton pour activer le son.
```

CODE RÉSULTANT :

```
<!-- templates/form.html (Fatoumata) -->
<div id="background-video" class="video-container"></div>
<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,
            'controls': 0,
            'playsinline': 1,
        },
        events: {
            'onReady': function(event) {
                event.target.playVideo();
            }
        }
    });
}

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

```
        isMuted = true;
    }
}
</script>
```

SLIDE 17-18 : Optimisation Performance

PROMPT à l'IA :

> "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.
```

CODE RÉSULTANT :

```python
# cache_manager.py (Hector)
import json
from pathlib import Path

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

def get_cached_films() -> List[Dict]:
    """Charge les films enrichis depuis le cache."""
    if CACHE_FILE.exists():
        with open(CACHE_FILE, "r", encoding="utf-8") as f:
            films = json.load(f)
        return films
    return []

def cache_films(films: List[Dict]) -> None:
    """Sauvegarde les films enrichis dans le cache."""
    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)
```

```css
/* static/css/loading.css (Fatoumata) */
#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;
}

.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;
}
```

SLIDE 19-20 : Détection d'Émotions Faciales

PROMPT à l'IA :

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

RÉPONSE de l'IA :

```
Utilisation de DeepFace (modèle pré-entraîné).
Détection de visage avec OpenCV.
Mapping des émotions DeepFace vers nos émotions.
```

CODE RÉSULTANT :

```python
# emotion_detection.py (Hector)
from deepface import DeepFace
import cv2

def detecter_emotion_image(image_data: bytes) -> Dict[str, any]:
    """Détecte l'émotion à partir d'une image avec DeepFace."""
    # Convertir 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)

    if len(faces) > 0:
        # Analyser avec DeepFace
        result = DeepFace.analyze(
            img_path=tmp_path,
            actions=['emotion'],
            enforce_detection=False
        )

        emotion_detected = EMOTION_MAPPING.get(result['dominant_emotion'],
"neutre")
        return {"emotion": emotion_detected, "confidence": confidence}

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

SLIDE 21-22 : Évaluation & Notebooks

PROMPT à l'IA :

> "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
```

CODE RÉSULTANT :

```
# notebooks/evaluation_notebook.ipynb (Toute l'équipe)
# Analyse de qualité
print(f"Nombre total de films : {len(df)}")
print(f"Taux de complétude : {completeness:.1f}%")

# Évaluation 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, note moyenne:
{avg_rating:.2f}")

# Matrice de confusion
```

```
sns.heatmap(confusion_matrix, annot=True, fmt='d', cmap='YlOrRd')
plt.title('Distribution des Genres par Émotion')
```

SLIDE 23 : Résultats & Métriques

📊 **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 des recommandations : **8.50/10**
- Amélioration de **+2.41 points** vs moyenne globale

✅ **Modèles IA :**

- TextBlob : Analyse de sentiment intégrée
- DeepFace : Détection d'émotions faciales
- Système de scoring : Fonctionnel

✅ **Interface Web :**

- Design moderne et responsive
- Vidéo de fond interactive
- Indicateurs de chargement
- Expérience utilisateur optimisée

SLIDE 24 : Démo Live

🎬 **DÉMONSTRATION :**

1. **Recherche par titre**
    - Tapez "The Matrix"

- Voir les détails du film

2. **Recommandation par émotion**
   - Choisissez "stressé"
   - Recevez 20 films adaptés
3. **Détection faciale** (bonus)
   - Activez la webcam
   - L'IA détecte votre émotion
   - Recommandations automatiques
4. **Vidéo de fond**
   - Ambiance cinématographique
   - Son activable

SLIDE 25 : Technologies Utilisées

🔧 **STACK TECHNIQUE :**

Back-end :

- Python 3.12
- Flask (API web)
- Pandas (traitement données)

IA & ML :

- TextBlob (analyse de sentiment)
- DeepFace (reconnaissance faciale)
- TensorFlow (deep learning)
- OpenCV (traitement d'images)

Front-end :

- HTML5 / CSS3
- JavaScript (ES6+)
- YouTube Iframe API

Données :

- Dataset TMDB (4,803 films)
- Cache JSON pour performance
- Hugging Face (optionnel)

Évaluation :

- Jupyter Notebook
- Matplotlib / Seaborn
- NumPy / Pandas

SLIDE 26 : Défis Relevés

💪 **DÉFIS TECHNIQUES RÉSOLUS :**

1. **Problème d'autoplay vidéo**
    - ✅ Solution : YouTube Iframe API
2. **Conflits de dépendances**
    - ✅ Résolution : Gestion des versions (numpy, Pillow)
3. **Performance au démarrage**
    - ✅ Solution : Système de cache
4. **Évaluation des modèles**
    - ✅ Solution : Notebook complet avec métriques
5. **Erreurs d'import dans Jupyter**
    - ✅ Solution : Réorganisation des imports

SLIDE 27 : Améliorations Futures

🚀 **ROADMAP :**

Court terme :

- Fine-tuning TextBlob sur critiques de films
- Optimisation DeepFace pour conditions réelles
- Tests utilisateurs et feedback

Moyen terme :

- Filtrage collaboratif
- Personnalisation par utilisateur
- Historique des recherches

Long terme :

- Déploiement cloud (AWS/Azure)
- Application mobile
- API publique pour développeurs

SLIDE 28 : 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 dans la recherche
- Découvertes personnalisées
- Expérience immersive

📊 Pour l'équipe :

- Maîtrise des technologies IA
- Expérience en développement web
- Collaboration efficace

SLIDE 29 : 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

SLIDE 30 : Questions ?

❓ **QUESTIONS & RÉPONSES**

Contact :

- Gémima ONDELE POUROU
- Fatoumata BAH
- Hector KOMBOU

Code source : Disponible sur GitHub **Démo :** [Lien si disponible]

📝 NOTES POUR LA PRÉSENTATION

Points Clés à Souligner :

1. **Approche Progressive**
    - Montrer comment chaque prompt a construit sur le précédent
    - Démonstrer l'évolution du code

2. **Collaboration**
    - Mettre en avant la répartition claire des tâches
    - Montrer la complémentarité de l'équipe
3. **Valeur Produit**
    - Insister sur l'utilité réelle
    - Métriques de performance concrètes
4. **Technologies Modernes**
    - Stack technique complet
    - Utilisation d'IA/ML
5. **Qualité**
    - Évaluation rigoureuse
    - Code documenté
    - Tests et métriques

Conseils de Présentation :

- **Démo live** : Préparez bien la démo pour éviter les bugs
- **Timing** : ~15-20 minutes de présentation + 5-10 min questions
- **Interaction** : Posez des questions au public
- **Visuels** : Utilisez des captures d'écran de l'application
- **Code** : Montrez quelques extraits clés, pas tout

🎨 SUGGESTIONS DESIGN (Canva)

- **Couleurs** :
    - Principal : #1a1a2e (Bleu foncé cinématographique)
    - Accent : #e94560 (Rouge/corail)
    - Secondaire : #0f3460 (Bleu moyen)
    - Texte : #ffffff / #f1f1f1
- **Polices** :

    - Titres : Montserrat, Poppins (Bold)
    - Corps : Open Sans, Roboto (Regular)

- **Éléments visuels** :
    - Icônes de films 🎬
    - Graphiques de métriques
    - Screenshots de l'application
    - Schémas d'architecture
    - Timeline du développement

- **Animations** (si PowerPoint) :
    - Apparitions progressives
    - Transitions douces
    - Zoom sur code important