CineRecommender: Personalized Movie Recommendation System

1. Project Overview and Goals

1.1. Introduction

The CineRecommender system is a web application designed to combat **information overload** and **decision fatigue** common in modern streaming platforms. The core objective is to move beyond generic "Top 10" lists and provide users with highly personalized, accurate, and context-aware movie recommendations. The system is built on a robust foundation of the Django web framework and a powerful Collaborative Filtering algorithm.

1.2. Problem Statement

Users often spend excessive time searching for content that matches their current preferences. Existing services frequently lack transparency regarding *why* a movie was recommended. The goal of CineRecommender is to use predictive modeling to solve this by learning latent user interests and providing full transparency into the prediction process via a detailed analysis view.

1.3. Key Project Goals

1. Achieve high prediction accuracy for unseen movies (evaluated using RMSE).
2. Develop a scalable, secure, and fully functional web application (MVP).
3. Implement a contextual recommendation layer (Mood-based filtering).
4. Maintain a professional and responsive User Interface (UI/UX).

2. Architecture and Technology Stack

2.1. System Architecture

The application follows the industry-standard **Model-Template-View (MTV)** architecture provided by the Django framework. This architecture enforces a clear separation of business logic (Views/Controllers), data structure (Models), and presentation (Templates).

2.2. Technology Stack

Layer	Technology	Purpose
Backend Framework	Python 3.x, Django 3/4+	Core web framework, routing, server logic, and ORM.
Recommendation Engine	Python, NumPy	Mathematical processing for vector generation and similarity calculation.

Frontend Styling	HTML5, CSS3, Bootstrap 5	Responsive, modern, and dark-themed user interface.
Database	(Assumed PostgreSQL/SQLite)	Storage for Movie metadata, User Profiles, and interaction history.
Security	Django Built-in Auth	User authentication, session management, and CSRF protection.

3. The Recommendation Engine (CF/DL Algorithm)

The core intellectual property of the project lies in the recommendation engine, which employs a **Simulated Deep Learning (DL) approach using Matrix Factorization (MF)** principles, as detailed in `project_algorithm.md`.

3.1. Latent Factor Prediction

The system relies on finding latent features (hidden reasons for preference) that neither the user nor the movie explicitly states.

- **Embeddings Generation:**
  - **User Embedding (V_u):** A fixed-size vector representing the user's learned **taste profile**. It is generated based on the unique `user.pk`.
  - **Movie Embedding (V_i):** A fixed-size vector representing the movie's latent **content features** (e.g., style, pace, subtle genre blend). It is generated based on the unique `movie.pk`.
- **Prediction Score Calculation:** The system predicts the score ($\hat{r}_{u,i}$) a user u would give to movie i by calculating the **dot product** of their respective vectors:

$$\hat{r}_{u,i} = V_u \cdot V_i$$

A higher resulting score indicates greater mathematical similarity between the user's taste and the movie's features, leading to a higher recommendation ranking.

3.2. Implicit Feedback & Data Collection

The model trains on implicit feedback, which represents real-world user engagement (see `project_algorithm.md`):

Data Source	Signal Strength	Purpose
`Watchlist`	Positive (Strong)	User explicitly saved the movie (Intent to watch).
`RecentlyViewed`	Implicit (Weaker)	User started or completed the movie.

Filtering:	Movies present in the user's combined `interaction_ids` are excluded from the recommendation output.

3.3. Cold Start Mitigation

The system mitigates the "Cold Start Problem" (lack of data for new users) using **Profile-Based Filtering**:

1. **Initial Data Gathering:** New users are prompted to select **Favorite Genres** via the `UserProfileForm` (referencing `movies/forms.py`).
2. **Initial Recommendation:** Before sufficient viewing history is established, the system defaults to recommending highly-rated movies that strictly match the user's selected **Favorite Genres**, ensuring immediate, relevant content.

3.4. Feature Engineering: Mood-Based Context

The system integrates **contextual awareness** through the `current_mood` selection (from `movies/forms.py`) for short-term preference adjustment.

- **Mechanism:** The selected mood (`Chill`, `Exciting`, etc.) acts as a **short-term weight adjustment** applied dynamically to the User Embedding (V_u).
- **Example:** Selecting **'Exciting'** temporarily amplifies the weights in V_u that align with 'high action' or 'intense' movie features, shifting the resultant recommendations without permanently altering the user's long-term learned taste profile.

4. Key Features and Implementation Details

4.1. User Authentication and Account Management

The system relies on secure, built-in Django User Authentication.

- **Login Implementation:** The `login.html` template utilizes the standard Django `AuthenticationForm` and enforces **CSRF protection** (`{% csrf_token %}`). It includes clear visual feedback for success/error messages and directs users to the registration page.
- **Registration Implementation:** The `register.html` template uses the Django `UserCreationForm`, ensuring secure password hashing and validation policies are enforced before account creation.

4.2. Profile Management and Preference Tracking

The user profile serves as the primary input mechanism for the recommendation algorithm.

- **Template:** `profile.html` is the frontend interface for managing user data.
- **Form Implementation (`movies/forms.py`):**
  - The `UserProfileForm` uses `forms.ModelForm` linked to the `UserProfile` model.

- **Favorite Genres:** Implemented using a `forms.MultipleChoiceField` with a `CheckboxSelectMultiple` widget to allow multi-select.
- **Current Mood:** Implemented using a `forms.ChoiceField` with a stylized `forms.Select` widget.
- **Critical Data Handling Fix:** The `__init__` method in `UserProfileForm` and the `update_profile` view logic (`movies/views.py.bak`) include a necessary fix: converting the stored comma-separated string of genres back into a Python list (`.split(',')`) for correct rendering of selected checkboxes, and converting the list back into a string (`",".join(...)`) upon saving.

4.3. User Interaction Tracking

User behavior is critical for the CF model and is tracked using dedicated models and efficient view logic.

- **Watchlist Management:**
  - `add_to_watchlist` view logic uses `get_or_create` to prevent duplicate entries for the same user/movie pair.
  - `remove_from_watchlist` view logic uses `Watchlist.objects.filter(...).delete()` (referencing `movies/views.py.bak`) for secure, user-specific removal.
- **Recently Viewed:** Displayed on `profile.html` , this list is critical for generating the short-term `interaction_ids` data set for the algorithm.

5. Database Design and Security

5.1. Core Models (Schema Representation)

The system relies on models that capture both static movie data and dynamic user interaction data.

Model	Purpose	Key Fields	Interaction Type
`Movie`	Static movie metadata	`title` , `year` , `director` , `imdb_rating` , `poster_url`	Content
`UserProfile`	User preferences	`user` (OneToOne), `favorite_genres` (String), `current_mood` (String)	Preference/Context
`Watchlist`	User interaction (Positive)	`user` (ForeignKey), `movie` (ForeignKey), `added_at`	Implicit Feedback
`RecentlyViewed`	User interaction (Implicit)	`user` (ForeignKey), `movie` (ForeignKey), `viewed_at`	Implicit Feedback

5.2. Security and Data Privacy

Security was a primary consideration, utilizing the built-in defenses of the Django framework (Referencing Q&A Prep Sheet, Q9 and `team_roles.md`, Role 4).

- **Authentication:** Utilizes Django's robust hashing algorithms for storing passwords (never plaintext).
- **Data Masking:** The core algorithm operates exclusively on numerical **User IDs** (`user.pk`) and **Movie IDs** (`movie.pk`) to generate embeddings. This practice ensures that no Personally Identifiable Information (PII) is exposed to the mathematical model.
- **Cross-Site Request Forgery (CSRF) Protection:** The `{% csrf_token %}` template tag is mandatory in all POST forms (`login.html` , `register.html` , `profile.html`) to prevent malicious attacks.
- **SQL Injection Prevention:** All database operations use Django's Object-Relational Mapper (ORM), which automatically sanitizes query inputs.

6. Project Team Structure and Roles

The project was executed by a 5-member team structure, ensuring comprehensive coverage from the algorithm core to the final user interface and documentation. (Detailed in `team_roles.md`)

6.1. Project Lead & Backend Architect (Team Member 1)

- **Focus:** Overall project oversight, defining the MTV architecture, and deployment strategy.

6.2. Machine Learning & Algorithm Specialist (Team Member 2)

- **Focus:** Core algorithm implementation (Matrix Factorization), Feature Engineering (Mood Adjustment), and Model Evaluation (RMSE).

6.3. Frontend & User Experience (UX) Developer (Team Member 3)

- **Focus:** UI/UX design, Bootstrap integration, responsiveness, and implementation of all user-facing templates (`home.html` , `profile.html`).

6.4. Database & Security Engineer (Team Member 4)

- **Focus:** Designing data models, implementing Django User Authentication, and ensuring all security protocols (CSRF, Data Masking) are adhered to.

6.5. Quality Assurance (QA) & Documentation Specialist (Team Member 5)

- **Focus:** Comprehensive testing, bug reporting (e.g., the `favorite_genres` loading fix in