AI Engineer

Dataset → https://drive.google.com/drive/folders/1SQ-lt7ikWidpNAigz5Z4TPIACN9_YCXG?usp=sharing

Technical Test – AI Engineer (Recommendation System)

Goal

Build a **small working recommendation app** for a streaming platform (TV, Movies, Series, Microdrama) that:

- Reads interaction data (watch history) from CSV files
- Generates **global popular** recommendations
- Generates **personalized** recommendations for a user
- Exposes the recommendations through **either**:
    - a **CLI program**, or
    - a **simple web API** (FastAPI/Flask)

You have full freedom on model choice, as long as it uses the provided data.

---

1. Data (Input)

You will be given **three CSV files**:

1.1 `users.csv`

Columns (example):

- `user_id` (string)
- `age` (integer)
- `gender` (string, e.g. "M", "F", "O")

- `region` (string, e.g. "Jakarta", "Bandung")

Example:

```
user_id,age,gender,region
u1,25,M,Jakarta
u2,32,F,Bandung
u3,19,F,Surabaya
```

1.2 `items.csv`

Columns (example):

- `item_id` (string)
- `title` (string)
- `content_type` (string, e.g. "tv", "movie", "series", "microdrama")
- `genre` (string, e.g. "drama", "family", "romance")

Example:

```
item_id,title,content_type,genre
i1,Drama Series A,series,drama
i2,Family Show B,tv,family
i3,Microdrama C,microdrama,romance
```

1.3 `events.csv`

Columns (example):

- `user_id` (string)
- `item_id` (string)
- `event_type` (string, e.g. "play")
- `watch_seconds` (integer, seconds watched)
- `timestamp` (ISO datetime string)

Example:

```
user_id,item_id,event_type,watch_seconds,timestamp
u1,i1,play,1200,2025-01-01T10:00:00
u1,i2,play,300,2025-01-01T11:00:00
u2,i2,play,1800,2025-01-02T09:00:00
u3,i3,play,600,2025-01-02T12:00:00
```

Assumptions:

- `watch_seconds` = implicit feedback strength (more = stronger interest).
- Data volume is small enough to fit in memory.

---

2. Functional Requirements

You can choose **one** of these implementation styles:

- **Option A – CLI app**
- **Option B – Web API**

The underlying logic should be the same.

2.1 Core Features (must-have)

2.1.1 Load Data

On start, the app must:

- Load `users.csv`, `items.csv`, `events.csv`
- Handle basic data issues gracefully (e.g., missing values, unknown IDs)

No need for fancy logging, but errors should not crash without explanation.

2.1.2 Global Popular Recommendations

Implement a **global popularity recommender**:

- Compute a popularity score for each item using `events.csv`.
    - You can use:

        - total `watch_seconds` per item, or
        - number of events per item, or
        - a combination.
- Implement a function like:

```
def recommend_popular(k: int = 10) -> list:
    """
    Return a list of top-k items by global popularity.
    Each item should at least include: item_id, title.
    """
```

Expected behavior:

- Returns top `k` items sorted by popularity (highest first).
- If `k` is larger than available items, just return all items.

2.1.3 Personalized Recommendations

Implement a **simple personalized recommender** using watch history:

- Use `events.csv` to build a user–item interaction signal (e.g. matrix).
- You can choose the method, for example:
    - Item-based similarity (cosine similarity using interaction vectors).
    - User-based similarity.
    - Simple matrix factorization (e.g. SVD).
- Must implement a function like:

```
def recommend_for_user(user_id: str, k: int = 10) -> list:
    """
    Returns a ranked list of recommended items for the given user.
    Do not recommend items the user has already heavily watched.
    Each item should include at least: item_id, title.
    """
```

Requirements:

- If `user_id` exists:
    - Use historical interactions of that user and others to compute recommendations.
- Do **not** recommend items that user has already watched a lot (you can define the threshold yourself, e.g. more than X seconds).
- If `user_id` does not exist / has no history:
    - Fall back to `recommend_popular(k)`.

2.2 Interface Requirements

Option A – CLI App

Create a CLI script, for example `main.py`, with at least:

1. Get popular items:

    ```
    python main.py popular --k 5
    ```

    Expected output (example, text in console):

    ```
    Top 5 Popular Items:
    1. i2 - Family Show B
    2. i1 - Drama Series A
    3. i3 - Microdrama C
    ...
    ```

2. Get recommendations for a user:

    ```
    python main.py recommend --user_id u1 --k 5
    ```

    Expected output:

    ```
    Recommendations for user u1:
    1. i3 - Microdrama C
    ```

```
2. i5 - Action Movie X
...
(fallback to popular if user not found)
```

You may use `argparse`, `Typer`, or any CLI library you like.

Option B – Web API

Create a simple API (FastAPI or Flask), with at least:

1. Health check:
   - `GET /health`
   - Response:

     ```
     {"status": "ok"}
     ```

2. Global popular:
   - `GET /popular?k=5`
   - Response example:

     ```
     {
       "k": 5,
       "items": [
         {"item_id": "i2", "title": "Family Show B"},
         {"item_id": "i1", "title": "Drama Series A"}
       ]
     }
     ```

3. Recommendations for a user:
   - `GET /recommendations?user_id=u1&k=5`
   - Response example:

     ```
     {
       "user_id": "u1",
     ```

```
  "k": 5,
  "items": [
   {"item_id": "i3", "title": "Microdrama C"},
   {"item_id": "i5", "title": "Action Movie X"}
  ],
  "fallback_used": false
}
```

If user not found or cold start:

```
{
  "user_id": "u999",
  "k": 5,
  "items": [
   {"item_id": "i2", "title": "Family Show B"},
   {"item_id": "i1", "title": "Drama Series A"}
  ],
  "fallback_used": true
}
```

2.3 Optional/Nice-to-Have (if time allows)

These are **not required**, but good to see if you have extra time:

- Return a simple `"reason"` for each recommendation (e.g., “similar to item you watched: Drama Series A”).
- Filter recommendations by `content_type` or `genre` (e.g., only microdrama).
- Add a small function/endpoint to show a user’s watch history.

3. Non-Functional Requirements

- Use **Python** for the implementation.
- Code should be structured (e.g., separate data loading, model logic, and interface).

- App should run from a clean start with a short README or clear instructions (one or two commands) on how to run it:
  - For CLI: how to call the script.
  - For API: how to start the server and example URLs.

---

4. What We Will Look At

When we review your solution, we will look at:

- How you transformed watch history into a usable signal.
- How you designed the simple recommendation logic.
- How clear and maintainable your code is.
- If the app works end-to-end from input (CSV) to output (CLI/API).
- How you handle edge cases (unknown user, no data, etc.).