Humanoo Case Study — Applied AI Engineer Submission

1. Problem Exploration

I identified the core problem as the early drop-off of users within the first two weeks. This suggests insufficient personalization and lack of perceived value during onboarding.

Key assumptions I made:

- Users disengage when recommendations are too generic.
- Reminders at the wrong times cause users to ignore notifications.
- Without early progress, motivation declines quickly.

Signals I would track to personalize the early user experience:

- Login frequency and session length.
- Preferred content types (workouts, meditation, nutrition).
- Time-of-day usage patterns.
- Responses to onboarding surveys (e.g., fitness vs. mindfulness goals).
- Early conversions (likes, skips, completions).

2. Solution Proposal

My main proposal was to build a **learning-to-rank recommender system** using XGBoost.

How it fits into the user journey:

- Onboarding collects goals, interests, and availability.
- The system provides a 'starter pack' of tailored content for the first 2 weeks.
- Recommendations re-rank daily based on interaction signals.
- Notifications are aligned to the user's active

hours. Alternative ideas considered:

- User clustering for cold-start handling.
- LLM-generated motivational messages.
- Simple popularity-based defaults as fallback.

3. Prototype Implementation

I implemented the prototype in Python with a Makefile workflow.

- make setup – prepares the environment and installs dependencies.
- make test – runs the unit tests for functionalities and integrations
- make coverage – shows the coverage report for the unit tests
- make train – trains the XGBoost learning-to-rank model with dummy data.
- make run – launches the FastAPI service with a `/recommend` and '/feedback' endpoint.
- make eval – evaluates offline metrics (HitRate@K, MAP@K, Policy Match Rate, CTR uplift). The project structure includes:

- scripts/ folder with training, evaluation, API, and schema files.
- tests/ for unit testing.
- models/ for saving artifacts.
- data/ containing dummy user–item interaction data.

4. Trade-offs and Risks

- Cold start – mitigated with defaults and popularity priors.
- Explainability – XGBoost interpretability can be improved with SHAP.
- Bias/fairness – need re-ranking for diversity to avoid overfitting to one type of activity.
- Latency – caching and feature pre-computation are needed for fast responses.
- Scalability – production would need distributed serving and feature pipelines.

5. Optional (Next Steps)

If I had 2–3 more days in a discovery sprint, I would:

- Build user and item embeddings for better candidate retrieval.
- Add an A/B testing pipeline with proper logging.
- Introduce contextual bandits for balancing exploration and exploitation.
- Expand dummy data to simulate richer user journeys and more realistic feedback loops.