Humanoo Case Study

1. Problem Exploration

The central challenge I identified was the high drop-off rate of users within the first two weeks. This issue reflects a lack of personalization and limited perceived value during onboarding.

Key assumptions made:

- Users disengage when recommendations feel generic.
- Notifications or reminders sent at irrelevant times lead to disengagement.
- A lack of visible early progress reduces motivation and retention.

Signals to personalize the early user experience:

- Login frequency and average session length.
- Preferred content types such as workouts, meditation, or nutrition guidance.
- Time-of-day usage patterns to optimize notifications.
- Responses to onboarding questions such as wellness goals or availability.
- Early interactions like likes, skips, or completions.

2. Solution Proposal

My proposed solution was to implement a learning-to-rank recommender system using XGBoost. The model scores candidate activities for each user and ranks them based on predicted engagement.

How this integrates into the user journey:

- Onboarding collects goals, personal preferences, and daily availability.
- The system delivers a starter pack of personalized activities for the initial two weeks.
- Recommendations are updated daily using real interaction signals.
- Notifications are aligned with the user's preferred activity hours.

Alternative approaches considered:

- User clustering methods for handling cold-start problems.
- Motivational message generation using language models.
- Fallback to simple popularity-based defaults for robustness.

3. Prototype Implementation

The prototype was implemented in Python and orchestrated through a Makefile workflow. The project demonstrates not only the modeling process but also reproducibility, evaluation, and service deployment.

Workflow commands:

- make setup – prepare the Python environment and install dependencies.
- make test – run comprehensive unit tests covering training, API, and evaluation.
- make coverage – generate and review test coverage reports (92% achieved).
- make train – train the XGBoost model with dummy data simulating user interactions.

· make run – launch the FastAPI service exposing endpoints for recommendations and feedback.
· make eval – compute offline evaluation metrics such as HitRate@K, MAP@K, policy match rate, and CTR uplift.

Project structure:

· scripts/ – training, evaluation, API, and schema files.
· tests/ – unit test suite.
· models/ – saved model artifacts.
· data/ – synthetic datasets representing user–item interactions.
· docs/ – summary PDF and screenshots including test coverage.

4. Trade-offs and Risks

· Cold start – mitigated with onboarding defaults and popularity priors.
· Explainability – XGBoost interpretability improved with SHAP analysis.
· Bias and fairness – system needs diversity re-ranking to avoid over-representing a single content type.
· Latency – mitigated by pre-computing features and caching results.
· Scalability – production deployment would require distributed serving and data pipelines.

5. Next Steps

If I had additional time, I would:

· Add user and item embeddings for improved candidate retrieval.
· Develop an A/B testing framework to evaluate recommendations online.
· Introduce contextual bandits to manage exploration versus exploitation.
· Expand the dummy dataset to simulate realistic user journeys and feedback loops.