Take-Home Exercise: Loan Orchestrator

Goal

Build a minimal **loan application orchestrator** where an application is processed through a configurable **pipeline of steps** and ends in one of:
APPROVED | REJECTED | NEEDS_REVIEW.

Applications are created via an API (no need for an application form UI).
The UI should be used only to configure pipelines and run them.

A loan application consists of:
applicant_name
amount
monthly_income
declared_debts
country
loan_purpose

What We Value Most

- **Bootstrapping**: ability to set up a working project from scratch quickly.
- **Extensibility**: codebase structured so new steps/features can be added easily.
- **Functionality > Polish**: no need for custom CSS or visual design; leverage frameworks, libraries, or component kits to get to a working result.
- **AI use**: ability to use AI tools to help with fast development and fill skill gaps.

Scope

Backend

- API to:
  - Create applications (POST)
  - Define pipelines (steps + params + terminal rules)
  - Run pipelines (execute steps, persist logs, set final status)

- Fetch runs (history)

Frontend

- **Pipeline Builder**: add/remove/reorder steps, edit step params, edit terminal rules.
- **Run Panel**: pick application_id + pipeline_id, run, see logs + final status.

Step Catalog (Business Rules)

1. **Debt-to-Income Rule (`dti_rule`)**
   - Purpose: checks whether the applicant's debt burden is reasonable.
   - Logic: `dti = declared_debts / monthly_income`
   - Outcome: `pass = dti < max_dti` (default `0.40`)
   - Business meaning: if the applicant is spending more than 40% of their income on debt, they fail.
2. **Amount Policy (`amount_policy`)**
   - Purpose: enforce country-specific loan limits.
   - Params: loan caps by country (defaults: ES=30k, FR=25k, DE=35k, OTHER=20k).
   - Logic: `pass = amount <= cap_for_country`
   - Business meaning: applicants cannot borrow above the maximum allowed for their country.
3. **Risk Scoring (`risk_scoring`)**
   - Purpose: combine previous checks into a simple risk metric.
   - Logic: `risk = (dti * 100) + (amount/max_allowed * 20)`
   - Params: `approve_threshold` (default `45`)
   - Business meaning: lower scores mean safer applicants. If risk ≤ threshold, approve.

Bonus: Agent-Style Step

If you have extra time, implement one additional step that uses an **AI agent**.

- **Goal**: integrate a real agent (e.g., OpenAI API) into the pipeline to make a decision.
- Example:
  - **Sentiment Check (`sentiment_check`)**: analyze the `loan_purpose` text; reject if it contains risky keywords (e.g., "gambling", "crypto").
  - Business meaning: detect risky or speculative loan purposes.

(If you cannot use a real agent in your setup, you may simulate it, but the preferred solution is an actual API call.)

Terminal Rules

Configurable, ordered rules that map step outcomes to a final status.
Default:

1. If DTI or Amount Policy fail → REJECTED
2. If risk ≤ threshold → APPROVED
3. Else → NEEDS_REVIEW

(If you implement the bonus step: insert sentiment check rule before risk scoring.)

Deliverables

- Source code (backend + frontend). Preferably in a single GitHub repository.
- **README** with:
  - How to run the project (backend + frontend).
  - Example cURL/HTTPie commands.
  - How to run tests if included.
  - AI use. A brief explanation on how AI was used to support / speed up the prototyping.

Review Scenarios

We will test your solution with these inputs (default pipeline/params):

1. Ana, 12000 loan, 4000 income, 500 debts, ES → **APPROVED**
2. Luis, 28000 loan, 2000 income, 1200 debts, OTHER → **REJECTED**
3. Mia, 20000 loan, 3000 income, 900 debts, FR → **NEEDS_REVIEW**

*(Bonus step: Eva, 15000 loan, 5000 income, 200 debts, ES, purpose "gambling" → **REJECTED**.)*