tabiya

Upwork detailed deliverables and expectations

Compass is a digital tool that should:

1. **Elicit existing skills and map them to a contextualized skills taxonomy.**
2. **Elicit job and career preferences** (via light self-discovery and vignettes) and represent them as a structured preference vector suitable for downstream matching.
3. **Advise on careers and opportunities** – first at occupational pathway level, and ultimately as concrete jobs and/or upskilling opportunities.

The system must be able to operate:

- **Alongside human counsellors**, who can see, understand and override recommendations.
- **As a standalone counsellor**, interacting directly with youth/jobseekers.

We plan to contract two (2) developers who will collaborate on / split up the deliverables below (to be defined during contracting). They will be overseen by a tabiya internal developer, who can support when questions arise and will do code reviews to ensure code and documentation quality.

Please note that we have clear contribution guidelines, which outline our acceptance criteria for any work done on any of the elements below: https://github.com/tabiya-tech/taxonomy-model-application/?tab=readme-ov-file#contribution-guidelines (the README file on contribution guidelines). All deliverables below are defined must be functional and integrated into the existing Compass product.

The contractors will be required to collaborate closely throughout the engagement. This includes conducting weekly code reviews, with dedicated time allocated, to ensure mutual understanding of each other's work and alignment with project objectives. Because Epic 1 e.g. informs and shapes Epics 2 and 3, and Epic 4 provides the core machinery within the current product (also informing 2 and 3), the contractors must provide, at the outset of or as part of contracting, a detailed work plan that identifies their estimated dependencies across all epics. These interdependencies will be documented and tracked. Furthermore, the contractors must jointly commit to integrating all epics into a single, interoperable product. The final deliverable will include full documentation of all functionalities, confirmation of interoperability, and provision of post-implementation support. Contractors are expected to offer a defined period of complimentary post-implementation support, followed by a clear guideline on daily support costs beyond that period.

CONTENTS

tabiya

EPIC DESCRIPTIONS

General technical quality & Definition of Done

For any feature in any epic, “done” means all of the following are delivered:

- **Code quality**
  - Follows the project’s contribution guidelines and coding standards (as defined in the existing GitHub repo).
  - All new functionality covered by automated tests (unit tests +, where appropriate, integration tests).
  - No high-severity linting or static analysis errors.
  - ./run-before-merge.sh must pass with no errors before any feature is considered done.
- **Documentation**
  - Function-level docstrings for core logic.
  - Short architecture note explaining: inputs, outputs, main components, and how the new functionality integrates with existing modules.
  - End-user or operator instructions for any admin/data-curation functions.
- **Observability**
  - Logs sufficient to trace a user journey through preference elicitation, skills elicitation and recommendation.
  - Error handling implemented with meaningful messages for both users and maintainers.
- **Security & data protection**
  - No sensitive data written to logs.
  - All code must comply with the existing **Sensitive Data Protection** guidance in sensitive-data-protection.md.
  - All external API keys loaded from configuration / environment, not hard-coded.
- **Performance & responsiveness**
  - Conversational actions (ask question → show next message) should complete within e.g. P95 ≤ 3–5s under typical load.
  - Data lookups (jobs, trainings, paths) should feel instant on a normal connection (e.g. P95 ≤ 1s from backend).
- **Conversational UX**
  - Tone & readability
    - Simple, respectful language, avoiding jargon; test scripts should show this.
    - For Swahili: natural, Kenyan Swahili (not overly formal), with at least some reviewed example dialogues.
  - Error & edge cases
    - If the LLM is confused or low-confidence, it should say so and recover, not hallucinate or loop.

Tech stack & repository

All work must be done within the existing Compass monorepo at github.com/tabiya-tech/compass, using:

- The Python backend under backend/ for all server-side logic, data ingestion pipelines, APIs and agent orchestration.
- The TypeScript frontend(s) under frontend-new/ (and frontend/ where relevant) for user-facing flows.
- The IaC configuration under iac/ for any new infrastructure or configuration required by added components. GitHub
  No separate/parallel services or repos should be introduced unless explicitly agreed, and any such service must integrate cleanly with the existing deployment and IaC setup.

Agile delivery & deployment

The contractors are expected to:

1. **Work in an agile manner**:
   - Short iterations (e.g. 1–2 weeks).
   - Regular demos of usable functionality (not just code branches).
   - Early deployment of stable versions to QA/UAT environments.
2. **Deliver incremental improvements**, not just big-bang rewrites:
   - Start from the existing Compass product.
   - Improve incrementally, ensuring that existing functionality is not broken.
3. **Maintain deployment scripts / configs** so that Tabiya engineers can:
   - Deploy new versions to test/production environments.
   - Roll back if necessary.

Epic 1 – Help build/contextualize taxonomy and database

Goal: Provide production-ready, queryable data structures and APIs that support skills mapping, preference mapping, labor demand, jobs, training opportunities and youth data.

We need six primary databases to map responses / recommendations to. The contractor will work in collaboration with the research team to accomplish this (define fields, find data sources etc)

For each of the six databases, the contractor must deliver:

- **Data model & schema**
  - A database schema (tables/collections) with clear field definitions (name, type, constraints, allowed values).
  - Primary and foreign keys that support joining across occupation/skills, jobs, training, preferences, and youth profiles.

- **Data ingestion pipelines**
  - Scripts or services that can pull data from identified sources (e.g. ESCO, KeSCO, job platforms, Swahilipot ecosystem).
  - Basic data cleaning and normalization (deduplication, consistent naming, consistent IDs).
- **Programmatic access**
  - One or more APIs / services (or clearly defined modules) that:
    - Allow querying by occupation, skills, region, and other relevant filters.
    - Are designed to be consumed by the preference elicitation agent and recommender agent (Epics 2 and 3).
- **Data quality checks**
  - Minimal set of automated sanity checks (e.g. non-null fields, value ranges, reference integrity).
  - Logging and reporting of data ingestion errors.

All six databases (taxonomy, demand, jobs, training, preferences, youth) must be implemented as part of the existing **backend/** project:

- Data models defined using the same ORM / data-access layer currently used in backend/.
- Data ingestion pipelines implemented as Python modules/services inside backend/, integrated with existing task runners or cron mechanisms.
- Programmatic access exposed as API endpoints or service functions that follow the existing backend patterns for routing, dependency injection and configuration.

For payment reasons, in the budget, Epic 1a) refers to databases 1-3, and Epic 1b) refers to databases 4-6;

Epic 1a)

1. Contextualized occupation & skills taxonomy: [some of this work has been started]

Functional objectives:

- Maintain a Kenya-relevant occupation and skills taxonomy that:
  - Starts from ESCO, prunes irrelevant occupations, and adds KeSCO occupations and any missing ones (e.g. entrepreneurship roles).
  - Maps each occupation to a set of skills with levels or importance weights.
  - Optionally represents career "paths" as edges in a graph (entry roles → mid career → advanced roles).

Key functional deliverables:

1. **Taxonomy builder module**
   - Functionality to:

        - Import ESCO occupation & skill data.
        - Import KeSCO occupation data.
        - Flag ESCO occupations as “excluded for Kenya context” or keep them.
        - Add new occupations (e.g. local informal jobs, entrepreneurship) with metadata (industry, description, typical tasks).
    - All changes stored with provenance (e.g. “added by contractor on date X”).
2. **Occupation–skill mapping**
    - Data structure linking each occupation to a list of skills (and optionally levels).
    - Query functions such as:
        - get_skills_for_occupation(occupation_id)
        - get_occupations_for_skill(skill_id)
3. **(Optional) Career path graph**
    - Representation of career paths as a directed graph:
        - Nodes = occupations.
        - Edges = feasible transitions (possibly labelled with typical probability or recommended conditions).
    - Simple query functions for:
        - get_next_steps(occupation_id)
        - get_paths(start_occupation_id, horizon=N)

2. Labor demand data / estimates

Functional objectives:

- For each occupation (and possibly industry/region), store a qualitative or quantitative estimate of labor demand.

Key functional deliverables:

- **Labor demand schema**
    - Fields such as:
    - occupation_id
    - region (county / broader region)
    - demand_score (numeric, e.g. 1–5 or continuous)
    - demand_category (e.g. “low / medium / high”)
    - source (e.g. labor force survey YY, LMIS dataset)
    - last_updated_at
- **API / query functionality**
    - Functions like:
    - get_demand_for_occupation(occupation_id, region)
    - get_top_occupations_by_demand(region, k)
- **Integration with recommender**
    - The demand scores must be easily consumable by the recommender agent for ranking and explaining career suggestions.

- **Sources / Pipeline**
  - The information may come from labor force surveys or other LMIS data
  - Clearly document the pipeline of source information, to translation, to final numbers in the database

3. Database of actual jobs (to be scraped from a list of Kenyan job platforms)

Functional objectives:

- Maintain an up-to-date jobs dataset sourced from multiple Kenyan job platforms.

Key functional deliverables:

1. **Job scraping and ingestion**
   - For platforms including but not exclusive to:
     - Brighter Monday
     - Fuzu
     - MyJobMag
     - JobsinKenya
     - Jobweb Kenya
     - Careerjet
   - Scrapers or API clients that:
     - Pull job title, description, location, employer, salary (if available), contract type, posting date, URL, and other relevant fields.
     - Normalize occupations to your taxonomy (e.g. via text matching or embedding-based similarity).
     - This database/pipeline should be easily “refreshable”/”replicable”
2. **Job storage**
   - Schema with:
     - Job metadata (job title, description, date posted, date closed etc.)
     - Status flags (active/expired).
3. **Query layer**
   - Functions like:
     - get_jobs_for_occupation(occupation_id, region, k)
     - search_jobs_by_skills(skill_ids)

Epic 1b)

4. Skills training opportunities

Functional objectives:

- Provide a dataset of training opportunities (public universities, TVET, Swahilipot & partner ecosystem, other upskilling offers) that can be matched to youth profiles and occupations.

Key functional deliverables:

1. **Training schema**
    - Fields such as:
        - training_id, provider, delivery mode (online/offline), region
        - Duration, cost, schedule
        - Eligibility criteria (required qualifications, required skills)
        - target_occupation_ids (which occupations the training leads to)
2. **Eligibility & mapping logic**
    - Logic to:
        - Match a youth profile (skills + qualifications) to training opportunities where they meet eligibility criteria.(requires information on required skills & qualifications for each training opportunity)
        - Map each training to one or more occupations in the taxonomy.
3. **Query functions**
    - get_trainings_for_occupation(occupation_id)
    - get_eligible_trainings_for_youth(youth_profile)

5. Preference elements

Functional objectives:

- Maintain canonical preference dimensions and their descriptions for use in vignettes and for tagging occupations/jobs/trainings.

Key functional deliverables:

- **Preference schema**
    - Dimensions such as (initial list, subject to contextual revisions):
        - Financial compensation (wage/salary ranges, benefits, bonuses)
        - Working hours/flexibility
        - Job security / contract type
        - Work environment & culture
        - Commuting / location
        - Career advancement
        - Inclusivity / accessibility
- **Linking preferences to opportunities**
    - Ability to store, for each occupation/job/training:
        - A vector of tagged preference attributes (e.g. high flexibility, high earnings, low job security).
    - API functions:
        - get_preference_profile_for_occupation(occupation_id)
        - get_occupations_matching_preference_vector(preferences)

6. Youth database

- Containing the actual extracted experiences, skills, preferences and qualifications of jobseekers. This can be part of the module (saved from epic 2-4 process; most likely scenario) OR a 3rd party API where this information is already stored
- Youth profile schema
    - Personal identifier (pseudonymous ID), demographics where appropriate
    - Past experiences (linked to occupations and tasks)
    - Skills vector
    - Preference vector
    - Qualifications (certificates, diplomas, degrees)
    - Interaction history (e.g. sessions with agents, recommendations shown, actions taken)
- APIs to save and retrieve profiles
    - For use by Epics 2–4:
    - save_youth_profile(profile)
    - get_youth_profile(youth_id)

Epic 2 – preference elicitation agent

The **Preference Elicitation Agent** must be implemented as part of the existing backend chat orchestration.

- New agents or flows must plug into the same routing, session and storage mechanisms as the current Compass chatbot.
- New external LLM calls must use the existing configuration mechanism for external services (e.g. environment-driven keys and settings in backend/), not ad hoc HTTP calls scattered through the codebase.

Functional objectives:

- This agent should discuss with the jobseeker what they look for in a job
- Elicit the type of tasks or experiences the person has enjoyed in the past
- Get from the jobseekers their ideal job preferences in the form of vignettes, in order to help them understand tradeoffs and implications. For example, instead of just asking “how important is a high salary for you compared to flexible work hours”, it could present two possible occupations, ask which one the person would enjoy more and then probe why exactly.
- The objective will be to have a vector of preferences (with strength of preference for each) that can also be mapped to occupations/jobs; this means it needs to be information that we can also find out about the occupation side. An initial set of elements in this vector may be (but could be subject to change to match the context based on tests/conversations):
    - Financial Compensation *[work in progress – not a final definition]*:

        - Wage/Salary – in ranges
        - Frequency – per job/day/week/month
        - Basic Benefits – statutory SHA, NHIF, leave days
        - Other benefits – private insurance, overtime
        - Bonuses/13th salary/Equity (e.g. performance-based pay)
    - Non-Wage Amenities *[work in progress – not a final definition]*:
        - Working Hours/Flexibility: Standard hours, flexible schedules, remote work options.
        - Job Security: Likelihood of layoff, type of contract (e.g., permanent vs. temporary/gig work/no contract (per job))/jobs on commission (brokers/sales).
        - Work Environment/Culture: Autonomy, physical demands, social context, relationship with managers, social value/reputation (of career or employer),size of org/startup (individual or registered).
        - Commuting/Location: Distance and ease of travel.
        - Career Advancement: Training opportunities, promotion track.
        - Inclusivity: consideration for PWD needs (ramps/access)
- understand people's preferences directly over industries and occupations, or possibly even skillgroups/task groups; this will require the tool to explain pretty well what each of these mean and entail.
    - Tasks can, for example, be categorized under the following groups and people may have preferences over them: *[work in progress – not a final definition]*
        - Routine Tasks: Those that can be accomplished by following explicit rules (e.g., assembly line work, simple clerical tasks).
        - Non-Routine Cognitive Tasks: Activities like problem-solving, analysis, and strategic decision-making.
        - Non-Routine Manual Tasks: Activities like driving, complex repairs, or in-person service work.
        - Social/Interpersonal Tasks: Requiring communication, coordination, and negotiation.
- **Primary objective:** At the end of the exercise, we want to understand what are the elements of a career that would make people show up and persist, even when it feels hard!

Key functional deliverables

1. **Conversation flow design & implementation**
    - Implement a configurable conversation script or state machine that:
        - Starts with an introduction and consent.
        - Asks experience-based questions (e.g. "Tell me about a task you enjoyed from past work or school").

        - Presents vignettes (e.g. two occupations with different salary/flexibility trade-offs).
        - Probes why the youth prefers one option over the other.
2. **Preference vector construction**
    - Implement logic to:
        - Map conversation events and vignette choices to numerical scores for each preference dimension.
        - Normalize scores and store them in the youth profile.
3. **Configuration & extensibility**
    - Preference dimensions and vignettes should be configurable (via JSON/YAML or admin UI), not hard-coded in the model prompt only.
4. **Technical interfaces**
    - Input: youth ID and optional initial data (e.g. prior experiences from CV upload).
    - Output: preference vector, summary text, and conversation log.

Epic 3 – recommender / advisor agent

The **Recommender/Advisor Agent** must be implemented as part of the existing backend chat orchestration

- New agents or flows must plug into the same routing, session and storage mechanisms as the current Compass chatbot.
- New external LLM calls must use the existing configuration mechanism for external services (e.g. environment-driven keys and settings in backend/), not ad hoc HTTP calls scattered through the codebase.

Functional objectives:

- The main goal is to now synthesize the information elicited from the jobseeker (i.e. skills-vector and preference vector) together with the information stored in the databases described in (1.)
- Using the skills vector and preferences, we will likely want to first use an algorithm based on a graph model (e.g. Node2vec) to find top-k occupations, skill-trainings, career paths, or concrete jobs. *[initial proof of concept exists]*
- The agent itself should then take the top-k suggestions, rank them by perceived importance (e.g. taking preferences, skills and labor demand into account again, but using RAG or something similar rather than embeddings and cosine similarity) and discuss these options with the jobseeker.
- For example, it could first explain why it recommends certain choices.
- It should then have a short conversation with the jobseeker, trying to gage their interest in these suggestions, and nudge them to understand preference vs labor demand trade-offs in understandable language. That is, in the conversation the tool should explain that if the less preferred career/job has significantly higher labor demand it

might nonetheless be a good stepping stone; and provide framing that helps the jobseeker *start somewhere* (i.e. get out of inactivity) and build step by step.

Key functional deliverables

1. **Recommendation engine module**
    - Implement a service or module that:
        - Ingests youth profile (skills vector + preference vector).
        - Queries Epic 1 data sources.
        - Produces a ranked list of opportunities/occupation paths with scores per factor (skills, preferences, demand).
2. **Advisor agent layer**
    - Implement a conversational layer (possibly using RAG over explanations templates) that:
        - Fetches recommendations from the engine.
        - Generates explanations including explicit mention of:
            - Skill alignment
            - Preference alignment
            - Labor demand
        - Logs user feedback (like/dislike, "tell me more", etc.) back into the youth profile.
3. **Tech & integration details**
    - API endpoints or functions:
        - get_recommendations_for_youth(youth_id)
        - start_advisor_conversation(youth_id)

Epic 4: Improve conversation flow of skills elicitation agent

Epic 4 focuses on improving the existing skills elicitation agent and adding a Swahili-capable version.

Epic 4a)

Functional objectives:

Goal: Make the skills elicitation conversation faster, less repetitive and more natural, while still producing a structured skills and qualifications profile.

a. Suggest simplification of the architecture and re-write prompts to be less stringent, while still allowing for the tool to match to taxonomy skills in the backend
b. Specifically, the conversation needs to be (i) faster, (ii) less repetitive, and (iii) more "natural flow" conversational.
c. Consider 2 key user personas (i) one who is predominantly informal, short gigs doing informal tasks. Might/might not have it formalised as a CV but can speak to what they did, they don't necessarily have job titles but their tasks and years service are key (ii) a

more formal/informal worker, has had interventions and upskilling opportunities and has some form of CV with responsibilities.

   a. Both of this users want to understand their skills and present them formally
   b. The latest version of compass that you will work with will have a rudimentary CV upload feature (bulk extract of work experiences and skills). – might require a few modifications to server persona (ii) and qualification f below

d. Run internal evaluations of simulated conversations and make sure the tool has safekeeping flags (e.g. does not go off-topic, does not allow harmful speech or sensitive conversations)
e. Add a qualifications layer – After or during extraction of experiences, add a component to extract certifications such as diploma, degrees and recognize certificates of participation. This is particularly important for artisan jobs such as electrician, plumbing etc. The qualification layer will be useful in job matching where having a qualification is a minimum requirement
f. The extracted information will need to be saved in order to be useful for recommendation of jobs/career pathways and/or skills upskilling/retooling opportunities

Key functional deliverables

1. **Refactored skills elicitation agent**
   - Rewritten prompts and/or model orchestration logic, tested for:
     - Reduced repetition.
     - Shorter time-to-insights.
     - Natural conversational tone.
2. **Persona-aware flows**
   - Logic to detect persona (1) vs (2) or allow the user to self-identify.
   - Tailored question sets based on persona.
3. **Qualifications extraction module**
   - Detect and extract from:
     - Conversational mentions.
     - CV text (degrees, diplomas, artisan qualifications like electrician, plumber, etc.).
   - Store as a structured list in the youth profile for downstream matching.
4. **Persistence**
   - Ensure that all extracted information (experiences, skills, qualifications) is stored in the youth database (Epic 1b, DB 6).
5. **Evaluation & safety**
   - Internal test harness with at least:
     - Synthetic or anonymized conversation scripts.
     - Checks that the agent doesn't go off-topic or produce harmful/sensitive content.

tabiya

Epic 4b)

Goal: Enable the skills and preference elicitation flows (and possibly parts of the advisor) to run in Swahili, including localised occupation and task descriptions.

- Language: What would it take to make the tool work in Swahili? Consider language switching here and localisation. E.g. (Shamba boy & gardener), (hawker, nauza chai, sales man) are potentially all the same grouping. Evaluate and advice best approach: Would Gemini/Gemini Pro or an open source language model like Jacaranda be ideal? Pro's and con's on the technical and functional implementation to be presented and debated for ideal solution
- Unless blocked by major issues, **deliver a working Swahili version of the tool across all Epics**.

Key functional deliverables

1. **Technical evaluation document**
    - Short engineering note comparing model options and recommending one, including:
        - Integration path into current architecture.
        - Expected costs.
        - Risks and mitigations.
2. **Localisation layer**
    - Mapping of common Swahili (and code-switched) job terms to the underlying taxonomy.
    - Reusable dictionary or rules, possibly complemented by model-based semantic matching.
3. **Swahili-enabled conversation flows**
    - Updated prompts and agent workflows that:
        - Can accept user input in Swahili.
        - Generate outputs in Swahili (with correct tone).
        - Still produce the same structured skills and preference vectors.

tabiya

EPIC DELIVERABLES

A. Cross-Epic Deliverables (apply to Epics 1–4)

A1. Technical work plan & dependency map

- A short written plan (Markdown/Doc in the repo or attached to the contract) that:
  - Breaks work into epics and milestones.
  - Shows dependencies between Epics 1–4.
  - States who (which contractor) owns what.
- **Done when**: plan is agreed with Tabiya and kept roughly in sync with reality (updated at least once mid-engagement).

A2. Integrated code in Compass repo

- All features implemented **inside the existing Compass monorepo** (Python backend, TS frontend, IaC), following the existing structure.
- No "sidecar" repos unless explicitly approved.
- **Done when**: all epic branches merge into main (or equivalent) and run-before-merge.sh passes.

A3. Automated tests & evaluation harness

- New automated tests for each epic's core functionality (see per-epic below).
- A small set of "golden" test cases for agents and recommendations.
- **Done when**:
  - CI passes.
  - There are tests that fail meaningfully if a core behaviour breaks (e.g. taxonomy lookups, preference vector shape, recommendation API).

A4. Observability & runbooks

- Structured logs for main flows; basic metrics (e.g. counts, errors, latency).
- Short runbooks:
  - "How to re-run scrapers."
  - "How to debug a broken skills/preference session."
- **Done when**: an on-call dev can use logs + runbooks to diagnose common failures without asking the contractors.

A5. Handover & support plan

- One walkthrough session + brief handover note.
- Written statement of:
  - Complimentary support period length.
  - Daily rate for additional support.
- **Done when**: internal dev confirms they can run, test and deploy without the contractors present.

B. Epic 1a Deliverables – build/contextualize taxonomy and database

B1. Occupation & skills taxonomy module (DB1)

- ESCO-based occupation list:

  - Irrelevant occupations **flagged or removed** for the Kenyan context.
- KeSCO occupations:
  - Imported and mapped into the same taxonomy.
- New occupations:
  - Added (e.g. entrepreneurship/informal roles), each with description and skills.
- Simple query interface:
  - get_skills_for_occupation(occupation_id)
  - get_occupations_for_skill(skill_id)
- **Done when**: for a sample list of Kenyan occupations (provided by you), team can query skills and occupations via code/API and get sensible results.

B2. Career / occupation paths

- Graph or similar structure connecting occupations into career paths.
- Helper functions:
  - get_next_steps(occupation_id)
  - get_paths(start_occupation_id, horizon=N)
- **Done when**: given a few starting occupations, the system returns plausible next steps and short paths.

B3. Labor demand dataset & API (DB2)

- Schema for demand by occupation and region (score + category + source + timestamp).
- Ingestion pipeline from agreed sources (e.g. LMIS / surveys).
- Query functions:
  - get_demand_for_occupation(occupation_id, region)
  - get_top_occupations_by_demand(region, k)
- **Done when**: for several test occupations & regions, the team can fetch demand scores and use them in the recommender.

B4. Jobs scraping & mapping system (DB3)

- Scrapers or API clients for Kenyan job platforms (at least BrighterMonday, Fuzu, MyJobMag, JobsinKenya, Jobweb Kenya, Careerjet).
- Normalization pipeline:
  - Common schema (title, description, employer, location, salary if available, posting date, URL, etc.).
  - Mapping each job to one taxonomy occupation (or a shortlist).
- Query interface:
  - get_jobs_for_occupation(occupation_id, region, k)
  - search_jobs_by_skills(skill_ids)
- **Done when**: in staging, you can call the API and see recent Kenyan jobs correctly mapped to occupations for a curated list of test roles.

B5. Data quality checks

- Automated checks (nulls, invalid IDs, broken references, stale jobs).
- Basic tests that run as part of CI.

- **Done when**: corrupt/stale entries are detected and surfaced; ingestion fails loudly rather than silently poisoning data.

C. Epic 1b Deliverables – Databases 4–6 (Training, Preferences, Youth)

C1. Training opportunities dataset & API (DB4)

- Schema for trainings from:
  - Public universities, TVET, Swahilipot + partners, other upskilling offers.
- Each training includes:
  - Provider, mode, location, duration, cost, schedule.
  - Eligibility criteria (skills and/or qualifications).
  - Mapping to target occupations in taxonomy.
- Query functions:
  - get_trainings_for_occupation(occupation_id)
  - get_eligible_trainings_for_youth(youth_profile)
- **Done when**: for sample youth profiles and occupations, the API returns a realistic list of trainings with eligibility correctly applied.

C2. Preference dimensions registry (DB5)

- Canonical list of preference elements:
  - Financial (wage ranges, frequency, benefits, bonuses).
  - Non-wage amenities (hours, security, environment, commuting, advancement, inclusivity).
- Ability to tag occupations/jobs/trainings with a preference profile.
- Query functions:
  - get_preference_profile_for_occupation(occupation_id)
  - get_occupations_matching_preference_vector(preferences)
- **Done when**: a test preference vector can be used to retrieve and rank occupations with different profiles, and these tags are visible/usable by the recommender.

C3. Youth profile database (DB6)

- Schema to store:
  - Experiences, skills, preferences, qualifications, interaction history.
- CRUD APIs:
  - create_or_update_youth_profile
  - get_youth_profile
- Integrated with Epics 2–4:
  - Skills & preferences written automatically at the end of sessions.
- **Done when**: a youth completes skills + preference elicitation; their full profile can be retrieved and used by the recommender without manual steps.

D. Epic 2 Deliverables – Preference Elicitation Agent

D1. Preference conversation flow implementation

- Implemented conversational flow that:
  - Asks about past enjoyable tasks/experiences.
  - Presents vignettes with trade-offs (e.g. salary vs flexibility).
  - Probes reasons for choices.
- Runs on top of existing Compass backend chat architecture.
- **Done when**: in staging, a test youth can complete a preference session end-to-end without dead-ends, obvious repetition, or prompt errors.

D2. Preference vector computation module

- Logic that converts conversation events + vignette choices into a structured vector over:
  - Financial compensation components.
  - Non-wage amenities + task types.
  - (Optionally) preferences over industries/occupations/skill groups.
- Normalisation & storage rules documented.
- **Done when**: for sample conversation transcripts, the same preference vector is reproducibly produced and stored in DB6.

D3. Configurable vignettes & preference definitions

- Preference dimensions and vignettes stored in configuration (e.g. JSON/YAML or admin interface), not buried only in prompts.
- Ability to add/edit vignettes without code changes.
- **Done when**: you can add a new vignette (via config), restart/redeploy, and it starts appearing in conversations.

D4. Evaluation set for preferences

- A small set of test scripts / simulated conversations with expected qualitative outcomes (e.g. “strong salary preference”, “strong flexibility preference”).
- Automated check that resulting preference vectors match the expected pattern.
- **Done when**: running the evaluation script flags regressions if someone breaks the preference mapping.

E. Epic 3 Deliverables – Recommender / Advisor Agent

E1. Production recommender engine

- Implementation of a recommendation module that:
  - Ingests a youth’s skills + preference vector.
  - Queries all relevant Epic 1 data (taxonomy, demand, jobs, trainings).
  - Uses Node2vec / graph-based or embedding methods to find top-K:
    - Occupations
    - Trainings
    - Career paths
    - Concrete jobs
- **Done when**: an API like get_recommendations_for_youth(youth_id) returns structured, scored recommendations for test youth profiles.

E2. Ranking & scoring logic

- Explicit scoring function that combines:
  - Skill match.
  - Preference match.
  - Labor demand.
- Ability to inspect component scores per recommendation (for debugging & explanations).
- **Done when**: for given test profiles, changing preferences or demand leads to predictable, explainable changes in ranking.

E3. Advisor agent conversation

- Conversational layer that:
  - Calls the recommender.
  - Explains *why* top recommendations were chosen (referring to skills, preferences, demand).
  - Discusses trade-offs and stepping-stone roles.
  - Logs user feedback (like/dislike, "tell me more").
- **Done when**: in staging, a youth can get recommendations and have at least one short follow-up exchange where their feedback is logged and can be seen in their profile / logs.

E4. Recommendation evaluation harness

- A small set of "golden" youth profiles with:
  - Expected types of occupations/jobs (not necessarily exact IDs, but reasonable ranges/categories).
- Script or notebook to:
  - Run the recommender over these profiles.
  - Check basic expectations (e.g. not suggesting wildly irrelevant jobs).
- **Done when**: running the harness gives a simple pass/fail signal and is part of regular testing.

F. Epic 4a Deliverables – Skills Elicitation & Qualifications Layer

F1. Refactored skills elicitation flow

- Rewritten prompts / orchestration so that:
  - Conversations are shorter, less repetitive, more natural.
  - Skills are still mapped to the taxonomy correctly.
- Before/after comparison for at least a few test personas.
- **Done when**: internal tests confirm reduced turn count and repetition vs previous version, and mapping accuracy is not degraded.

F2. Persona-aware flows

- Logic to detect or select:
  - Persona 1: informal work, no formal titles.
  - Persona 2: more formal, has a CV.

- Different question templates and probing strategy per persona.
- **Done when**: starting a session as Persona 1 vs Persona 2 leads to clearly different conversation patterns, both ending in a skills vector.

F3. CV integration

- Use of existing CV upload feature to:
  - Extract work experiences and skills.
  - Merge them with conversational output.
- **Done when**: for Persona 2, uploading a CV and talking to the agent yields a combined experience/skills list, not two separate incompatible sets.

F4. Qualifications extraction layer

- Component that:
  - Extracts qualifications (degrees, diplomas, artisan certificates, participation certificates) from both:
    - Conversation text.
    - CV text.
  - Stores them in the youth profile in a structured way.
- Integrated with training and job eligibility logic.
- **Done when**: a test profile with explicit mentions of qualifications ends up with correct entries in DB6, and these influence which jobs/trainings are shown as eligible.

F5. Safety & evaluation for skills agent

- Guardrails to:
  - Avoid harmful/off-topic content.
  - Handle sensitive topics gracefully.
- Internal evaluation:
  - Simulated conversations that test both normal and edge-case behaviours.
- **Done when**: running the simulation suite shows no critical policy violations and highlights any regressions.

G. Epic 4b Deliverables – Swahili & Localisation

G1. Swahili model options assessment

- Short engineering note comparing:
  - Gemini/Gemini Pro.
  - Jacaranda or other Swahili-capable models.
- For each: pros/cons on:
  - Functional quality (Swahili, code-switching).
  - Latency/cost.
  - Integration complexity.
- Clear recommendation of one approach.
- **Done when**: Tabiya can sign off on a model choice (or small set of choices) based on this note.

G2. Localisation & synonym mapping

- Dictionary or ruleset mapping Swahili/Kenyan terms to taxonomy concepts, e.g.:
  - "shamba boy", "gardener" → shared occupation/skill group.
  - "hawker", "nauza chai", "salesman" → appropriate grouping.
- Accessible as a module used by skills/preference/recommender flows.
- **Done when**: test sentences in Swahili or code-switched language are correctly mapped to the same underlying occupations as their English counterparts.

G3. Swahili-enabled flows

- Skills elicitation and (where feasible) preference elicitation running in Swahili:
  - Inputs accepted in Swahili.
  - Outputs generated in natural Kenyan Swahili.
  - Same structured outputs (skills, preferences, qualifications) as in English.
- Language switching supported where appropriate.
- **Done when**: a test user can complete a full Swahili skills/preference journey and get recommendations; underlying stored profile is usable by the recommender.

G4. Swahili evaluation & examples

- A small set of Swahili test scripts and expected outcomes.
- A few "golden" Swahili transcripts that demonstrate:
  - Correct tone.
  - Correct mapping to taxonomy.
- **Done when**: evaluation scripts run cleanly and are part of regression checks when prompts/models change.

Milestone deliverables for contract payment are as below:

Milestone 1 (Week 1-2): Technical Work Plan & Epic 1a Foundation - $1,000

- Technical work plan with dependency mapping
- Database schemas for taxonomy, labor demand, and jobs
- ESCO/KeSCO taxonomy builder module operational
- Job scraping infrastructure setup for all 6 platforms

Milestone 2 (Week 3-4): Epic 1a & 1b Databases Complete - $1,250

- Complete job scraping for all platforms (BrighterMonday, Fuzu, MyJobMag, - JobsinKenya, Jobweb Kenya, Careerjet)
- Labor demand dataset and APIs
- Career path graph
- Training opportunities database
- Preference dimensions registry
- Youth profile database with CRUD APIs

Milestone 3 (Week 5-6): Epic 4a Skills Elicitation Complete - $1,250

- Refactored skills elicitation flow (faster, less repetitive, natural)
- Persona-aware conversation logic
- Qualifications extraction module
- CV integration
- Safety evaluations and test harness

Milestone 4 (Week 7-8): Epic 4b Swahili Implementation - $1,000

- Swahili model evaluation document
- Localization layer and synonym mapping
- Swahili-enabled conversation flows
- Swahili evaluation suite

Milestone 5 (Week 9): Final Integration & Handover - $500

- All automated tests and data quality checks
- Complete documentation and runbooks
- Integration testing across all epics
- Handover session