Sri Lanka Institute of Information Technology

Information retrieval and web analytics
IT3041

IRWA - Final Report 2025

Group Details

Group: Y3.S1.WD.DS.01

Submitted On: 2025.10.29

IT Number	Name	Email	Contact Number
IT23192782	A V Amarathunga	it23192782@my.sliit.lk	071 639 3153
IT23264366	M S R V Amararathna	it23264366@my.sliit.lk	077 190 7511
IT23216778	S A R U Amarasinghe	it23216778@my.sliit.lk	071 799 5000
IT23140752	A A Gunawardena	it23140752@my.sliit.lk	071 279 2376

Table of contents

Introduction

The **IRWA Agentic AI project** tackles a real workplace gap: many employees struggle to advance because they lack clear, evidence-based guidance to grow the right skills at the right time. We frame the solution as a **multi-step problem solver**—a *problem analyser* + *insight analyser*—designed to surface a user's pain points and turn them into actionable learning paths that improve employability and impact. The project centers on common blockers (from weak skill demonstration to difficulty positioning current skills in the market) and responds with a structured, AI-assisted process rather than ad-hoc tips.

Our **value proposition** is personalization and depth. Unlike instant plan generators, the system first discusses the user's needs and captures essential insights; only then does it create a tailored plan and learning resources. Beyond "what to learn," it explicitly aims to strengthen *how* users think—nudging outside-the-box, IQ-oriented development so people perform in cognitively demanding environments, not just follow a resource list.

Architecturally, the platform is built as a set of **agentic roles**—(1) User Analysis and (2) User Identification, supported by (3) Planning and (4) Execution/Verification in the fuller vision. For the MVP, we implemented the first two agents to keep scope focused while validating the approach. The runtime is split into an **Agentic AI pipeline** (from user prompt to response, with a Decision Gate that keeps turns in scope) and a **RAG pipeline** that supplies grounded knowledge to answers.

The **RAG knowledge base** is engineered for provenance and freshness. We converted curated PDFs into structured, versioned artifacts (Docling markdown + JSON blocks), then indexed them with a **hybrid FAISS (vectors) + BM25 (lexical)** scheme to balance semantic recall and exact-term precision. The runtime plans queries, retrieves and re-ranks evidence, and composes answers with clear traceability—reducing hallucinations and keeping guidance auditable and current.

On the **product side**, we designed a clean, accessible UI in **Figma**, and implemented the MVP with **Python**, **React**, **Tailwind CSS**, **MongoDB**, and a local **RAG KB** (Docling-based). This stack supports a responsive front end, efficient data handling, and reliable retrieval without unvetted external calls. We also embedded **Responsible AI practices** throughout: a Decision Gate that enforces task-scope fairness, user-led insight collection via surveys/encouragements, transparent boundaries and source chips in answers, and strict avoidance of demographic profiling. For commercialization, the MVP deliberately targets the **data-science job market**, beginning with the **junior level**, where demand and skill gaps are most acute.

In short, this team-of-four project delivers a focused, ethically grounded agentic system that first understands the user, then plans and teaches with traceable, high-quality evidence—so learners can build the right skills and the cognitive habits to use them well.

1. Project Methodology

> The topic at our hand is,
> **Multi-Step Problem Solver –> Problem analyser + insight analyser**

Under this section we will be exploring what system of methods and principles we have implemented or used to execute the task that our system is composed of. As per given by the university our domain name is a **step by step problem solver**. So, we decided to combine this topic to solve problem that employees are facing in the employment regarding upgrading their skills and preparing to be well prepared skilful, impactful individuals.

Problems identification In employment.

The main reason why lot of people get job rejections and certainly an employed person to finding it hard to climb the promotion ladder up is because of the following factors,

- Lack / poor intelligence and cognitive power to challenge the employee base.
- Skill demonstration is comparatively low in the selected field.
- Unable to understand how to place its current skill set in the job market /office environment and unable to find methods enhance current skill set.
- Unable to differentiate itself from rest of the people or be an impactful person.

Clear, feasible business idea.

In future, with the current speed in the advancement of technology and the rise in competition among people, job employment will be a crucial competitive factor. Learning and advancing their skills for their carrier is more important to stay ahead of the competition.

Value preposition –> Personalised plan

Unlike other platforms like chatGPT where per each chat the system will be making plans without understanding the users pain points, difficulties etc and generate plan all most instantly when a prompt is given from user about the need to improve their skills, relativity ai first discusses the skills needed to be advanced and take important user insights needed to produce a personalised plan. Bases in these collected insights and after understanding user's skills requirement, a personalised plan will be made and then executed where user will be gaining access to resources (videos, ai generated transcripts etc) to learn and advance their skills

Value preposition –> *Helping to think outside the box*

We not only build skills, but we increase the IQ bringing whole new idea to skill development. The current platforms ONLY provide people with guidance to boosts skills by suggesting resources and a roadmap. To perform well in every carrier, people should essentially think and work intelligently. learning things that is required for the field can only enhance intelligent power to a certain level and not to suit for a critically thinking environment. Not only learning things trough resources but also building the structural brain they need along with it.

Future upgrades we can venture into,

- **Skill-Gap Check:** Quick diagnostics across programming/SQL, statistics, ML basics, data handling, model evaluation, and simple deployment.
- **Personalized Learning Plan:** Only the essentials for Junior DS (no senior-level detours).
- **Hands-On Projects (portfolio-first):** Guided projects with real-world framing (EDA → features → baseline → iteration → error analysis → readme).
- **Portfolio Packaging:** GitHub structure, README templates, notebooks, demo scripts, and optional lightweight deployment tips (e.g., Streamlit).
- **Assessment & Feedback:** Short quizzes, rubric-based project checks, and improvement tips.
- **Interview Readiness:** Bank of typical junior questions, mock interview structure, and project storytelling practice.
- **Job-Search Basics:** Simple tracker (role/skills match), CV pointers tailored to DS, and LinkedIn/GitHub hygien

Agentic AI roles / requirements

The system or the user engagement platform is broken into 4 sub parts as described above and each substage is worked one by one methodically. All these 4 sub parts will be communicating and working together. Each sub part has its own function to perform in the game.

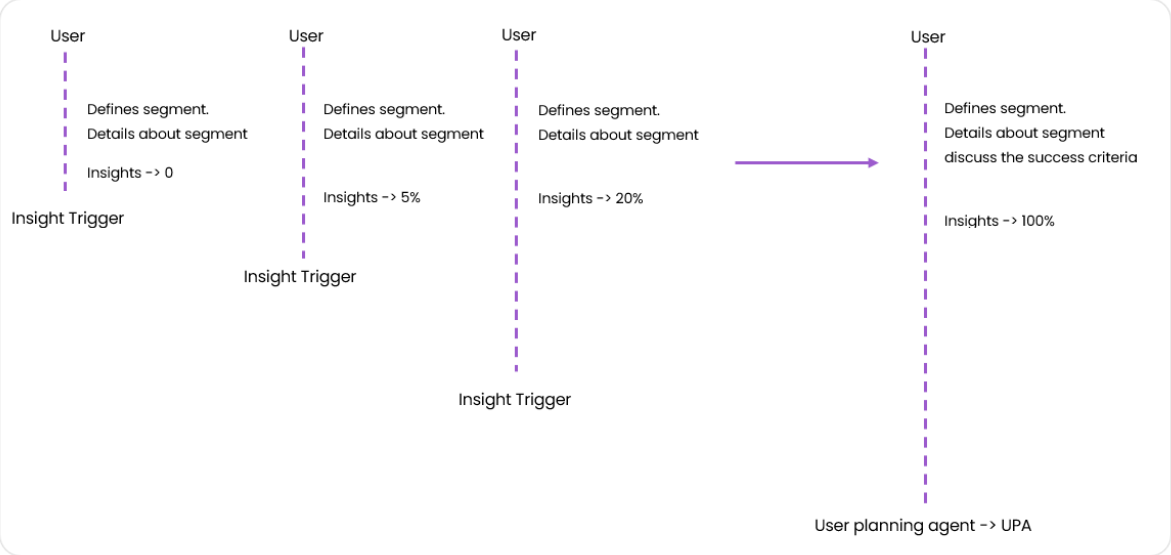
1) User Analysis agent + User identification agent
2) Planning agent
3) Execution agent
4) Verification agent

The complete system is composed of these agents. But as per considering the workload + requirements from university to initiate maximum of 2 interacting agents for now. Therefor the system will be made only for User analysis agent and user identification agent. Following is brief about the agents which we are going to build, what each agent does and what is its objective for the considered target users.

User Analysis AGENT (UAA) : requirement justification

Objective.
This section is dedicated to understanding and focusing on the user problem. We are encouraging the user to discuss the problem / pain points / weaknesses it has using the UAA. Here we do not do any planning on the skills user wants to upgrade but a formal discussion is being carried on.

Following are various paths that user who enters the user analysis agent (UAA) platform can take. This platform never does the planning for the problem user is facing but behind the scenes takes the insights that are necessary to craft the personalised plan for the user. Under insights in each path, it shows during the conversation how much of insights we were able to obtain as a percentage. An insights trigger will happen if the insights that needed to be obtain has not been obtained from the user completely and insight trigger will not happen if all required insights has been already obtained from the user.

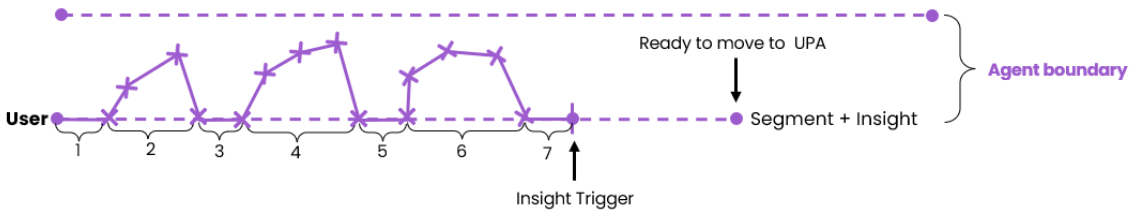


Under each conversation session now we need to define the strategy to take the insights/segment + problem discussion with the user. Here the maximum accuracy of the UAA could be obtained only if the user follows the pattern or strategy that UAA follows to get its objective done. But in most of the cases this will not happen as user tends to go off the track and deviates and only from the objective leaving around 0-99% only completed.

Here what we define by a problem is **employment category** + it associated **pain points** around. Some functions are listed out as follows,

- **#User engorgement** : Encourage user to discuss the pain points, strengths, weaknesses etc. which will help the UIA to capture the necessary insights using the 2 stages defined in UIA as well as the segment. If user deviates from this main objective implement strategies to bring back to the course.

This is the original path that user could take if he does not deviate. This will make sure that segment + insights that are needed



Each label 1-6 in the above diagram can be described as below,

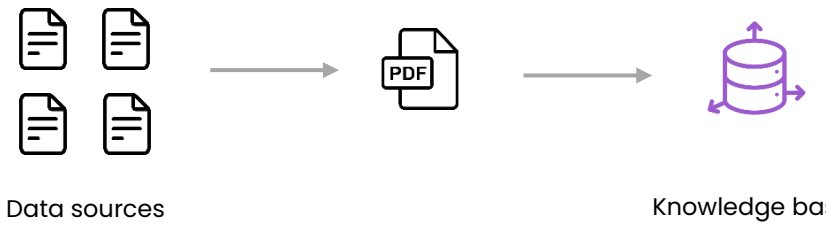
1) **Label 01** -> User tell about the segment.
2) **label 02 / label 04/label 06** -> with the 2 cross marks showing each chat placed by the user in the platform. This shows how user has deviated from the real path. For an example user asked to "*explain about the segment*" which they gave before.
3) **label 03 / label 05 / label 07** -> By implementing a strategy we bring the user that went off the track back to the track. This can involve asking a question about an insight which has to be collected. In this segment user now tells about the insight in the prompt.

Until insight trigger points the system currently was able to only take 15% of the required insights. Insight trigger point will trigger to collect the remaining the 85%. The labels 3,5,7 were the sections that took this 15% information.

- **#UIA Assiter** : Helping the user insight agent which is responsible for collecting insights to collect them without specifically generating surveys per each chat in the conversation, by making the user to discuss pain points strategically.
- **#Strictly no planning** : No plan will be given to develop the skill. Planning will be done only after required insights are collected which in turn produces the personalised plan.
- **#Information retrieval** : Provide user with information needed for the prompt given. Boundary has defined the scope that defines the possible segments which question from users could show up. This information is stored in knowledge base and will be used to answer users question.

Methode : Knowledge base for UAA

We designed the UAA's knowledge base and RAG layer using an information-need–driven approach. First, we mapped the exact question types UAA must answer and set a strict domain boundary (junior tech roles, learning paths, skill-gap analysis). We curated authoritative sources, split them into single-topic evidence chunks with owners, versions, and review dates, and indexed them for hybrid retrieval with scope filters. For each prompt, the system retrieves and re-ranks the most relevant evidence, and the LLM composes an answer strictly from those chunks with clear citations. Stale or conflicting content is handled via freshness rules and confidence, and the system refuses when evidence is insufficient. Quality is monitored with retrieval and groundedness metrics, and every answer is fully traceable, making UAA's guidance accurate, current, and auditable by design.



Purpose & Outcomes

The UAA's knowledge base and RAG layer ensure answers are grounded in verified evidence, with clear citations so users can trust and audit every response. This architecture reduces hallucinations, keeps guidance within scope, and helps the team maintain quality over time.

- Deliver answers strictly from approved evidence, with inline citations and a sources list.
- Keep responses accurate, current, and explainable by design.
- Enforce the UAA's domain boundary (e.g., junior-level tech roles, learning paths, skill-gap analysis).
- Provide auditable traces for each answer (what was retrieved, what was cited, and why).

Information-Need–Driven Design

We start from the questions UAA must answer and work backwards to the data and rules needed to answer them reliably.

- Map question types to specific evidence needs (e.g., skills matrices, curricula, role ladders, prerequisites, time estimates).
- Define an explicit boundary of coverage (what UAA will and won't answer).
- Curate a sources catalog with owners, update cadence, trust level, and expiry/refresh windows.
- Write answerability rules: if required evidence is missing or stale, UAA should state "insufficient evidence" and explain what's needed.

Knowledge Modeling & Chunking

We structure content as small, single-topic "evidence chunks" that are easy to retrieve, compare, and cite.

- Size chunks to fit one clear idea; split along headings and lists; keep terminology consistent.
- Attach rich metadata (owner, version, last reviewed, valid-until/refresh-by, domain, role level, tags).
- Normalize acronyms and units; keep canonical terms for the same concept.
- Use light cross-references so related chunks are discoverable without duplication.

Indexing & Retrieval

We combine semantic and keyword search with strict filters so only in-scope, high-quality evidence is surfaced.

- Build an embeddings index but also support keyword/BM25 and field filters (domain, role level).
- Apply boundary filters at retrieval time (e.g., junior level only for UAA).
- Re-rank top candidates to select the most on-point evidence for the exact question.
- Prefer newer or higher-confidence chunks when content conflicts or nears expiry.

Why a RAG?

Reduces hallucinations: The model cites concrete passages from our data instead of guessing
Keeps answers current: We can update the knowledge base without retraining the model.
Trust & traceability: Users can see *why* an answer is correct (linked evidence).

1) User Identification AGENT (UAA) : requirement justification

Goal: collect the insights needed *before* planning.
For any specific problem, before we deliver the personalised plan to the user, we need to collect the necessary insights needed to plan well. Before we carry out planning, execution steps, it is essential to know about *user insights*, then decide HOW the problem should be handled(planned, executed). This is done in 2 stages,

Stage 1 : **Auto-inference method**
first by observing user behaviour under UAA platform based on what user gives in prompt and the answer we provided to it, specific insights needed will be taken as a summary. This is taken based on what the user has given under the prompt. we have predefined set of insights in a vault. This vault is used to extract the phrases using an LLM to storage in database.

Example :
"I'm actually having a **difficulty** in **managing** my **time** and also the **workload**"
insights : time management : difficult
task management : difficult

Stage 2 : **Targeted micro-surveys method**
As mentioned to provide a personalised plan we need to obtain insights from the user. Per each conversation if the prompt placed by the user is relevant to an insight/insights we should collect from the insight collection, UIA will trigger a survey along with the answer which is given for the prompt placed by user asking the users to fill.

Example
"I prefer to **learn things** well using **visuals** mostly it is in fact really easy "

1) **UAA produce response for this prompt**,

2) **Main insight it falls under : LEARNING_PREFERENCES_MODALITIES**
UIA Triger question as surveys to be collected,

- **Primary modality (choose 1–2):** Visual, Text-first, Audio, Hands-on / Interactive.
- **Interactivity level**: Passive-first, Exercise-first, Project-first.
- **Memory supports (choose 1–2):** Spaced repetition / Flashcards, Concept maps / Mind maps, Annotated examples Teach-back.

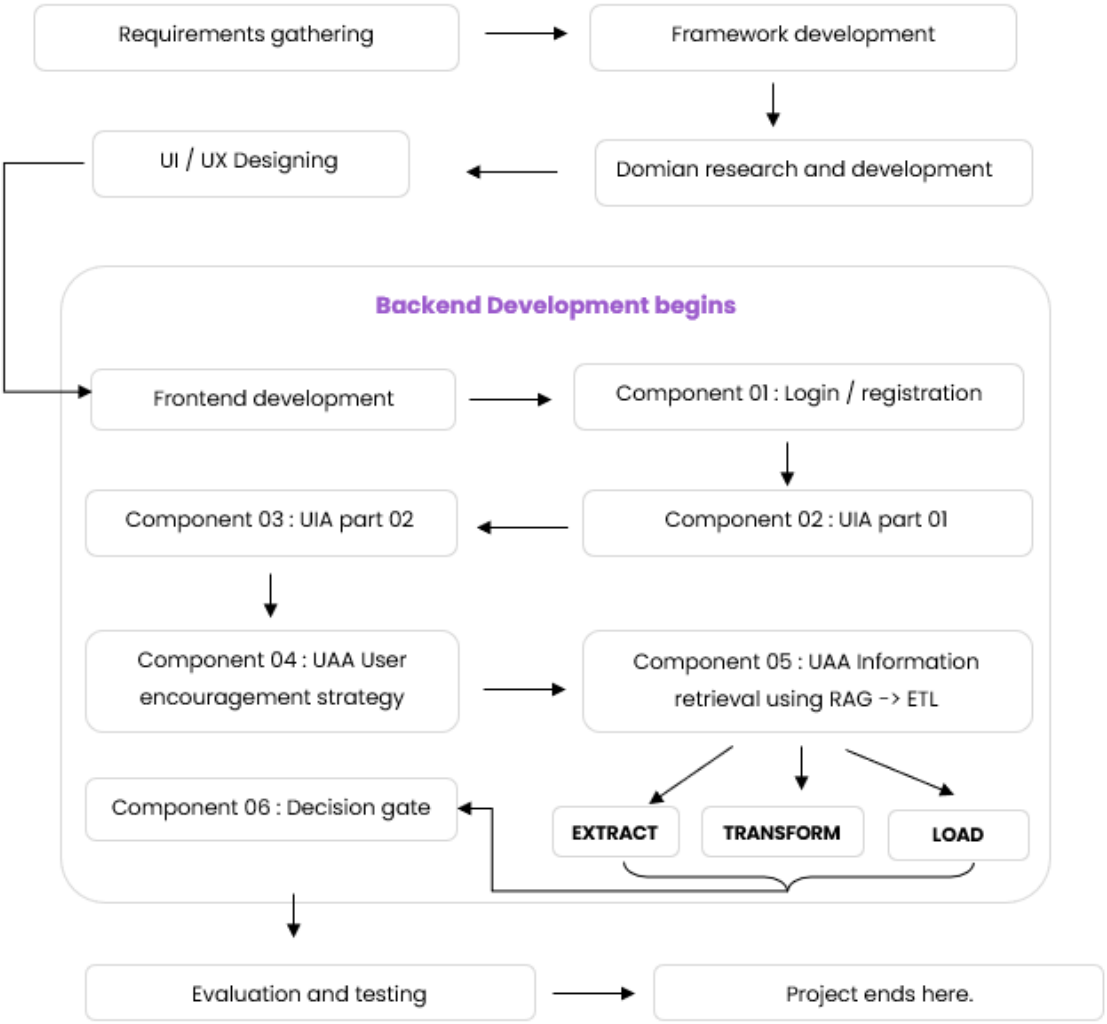
Later under evaluation analysis we will be exploring how each of the mentioned requirements are perfectly satisfied using results obtained from running the agentic ai system.

Project workflow Architecture (#1)

how we mastered the project

This section outlines how we carried out the development process as a team. With a disciplined procedure as guided by the leader of the team, we were able to reach our requirements milestones properly. Below is an architecture diagram that represents the whole development process,

Now we will analyse each section in the architecture diagram closely



1. #section 01 : Requirements gathering

Requirement gathering is the process of identifying, documenting, and managing the needs and expectations of stakeholders for a project. This critical initial phase establishes a clear understanding of what the project must achieve, providing a foundation for its design and development.

Justification :

We began by gathering all the essential requirements needed to complete the project. In an agentic AI system, requirement gathering mainly focuses on understanding what the final agent must

accomplish its key objectives, core tasks, and any special functions it should perform. This ensures the system is purpose-built around the agent's intended behaviour and operational goals.

2. #section 02 : Framework development / architecture diagram construction

We analysed and developed a comprehensive framework that encapsulates all the core concepts and techniques applied within the agentic AI system.

Justification :

This framework primarily focuses on three fundamental components: Large Language Models (LLM), Information Retrieval (IR), and Security. It provides a structured overview of how the concepts and methods under each component contribute to the overall functioning of the system. The framework serves as a foundational reference for designing the system architecture, system design, and agent communication flow, aligning with the requirements set by the university for the mid-evaluation and final presentation. Each concept within the framework is briefly or comprehensively described, offering an opportunity for further independent research and deeper understanding where desired. The following sections are all covered under high level architecture diagram drawn later.

System Architecture /system design

System architecture is an overview of the entire system that represents the structure and the relationships between various components. It helps to visually represent how different parts interact and function.

Communication flow diagram:

A communication flow chart is a graphical representation that illustrates how information flows within a project. It maps out the process, components, individuals, and communication channels involved.

3. #section 03 : Domain research and development

This is the first stage of ETL(Extract/ transform / Load) process which is "EXTRACT".

Justification

This section is used for Data collection. This involves collecting relevant data for the domain. Building a robust data pipeline begins with the collection of a wide range of data types from many different places. Businesses frequently mix unstructured data from papers or web pages, semi-structured data like logs and JSON files, and structured data from databases. The data we collect can be semi structured (structured and unstructured).

Before transforming and loading our data, the primary step is data collection from all relevant internal and external sources. This can include:

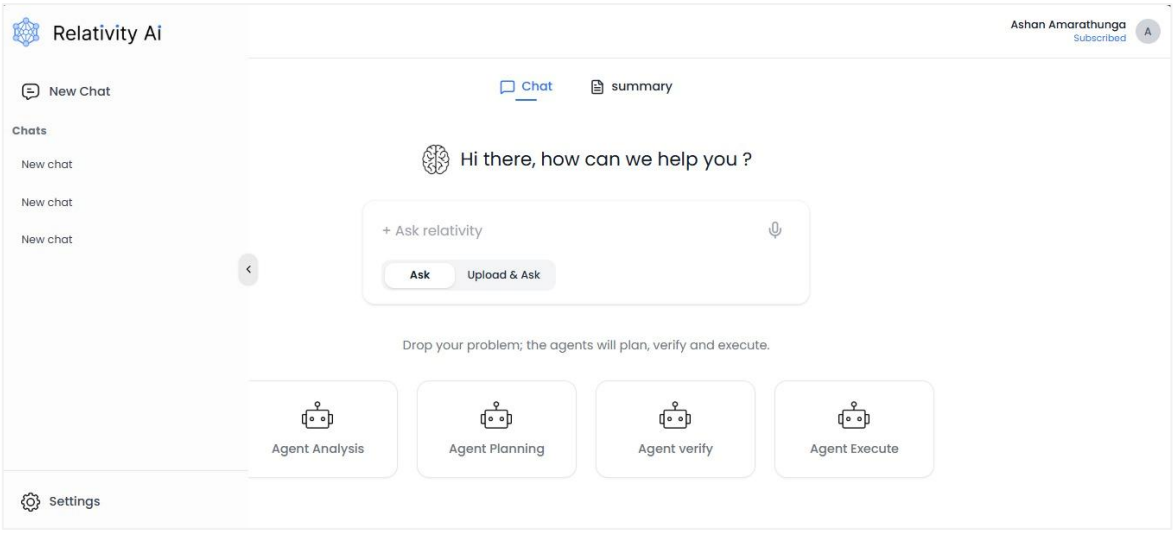
- ✓ Structured data from databases, unstructured data, or sensitive data stored in data warehouses.

In our case we mainly used information collected from chatGPT + google into pdfs. Together, these provide a broad representation of human language and knowledge. A set of 6 pdfs were prepared based on this data we collected.

4. #section 04 : UI / UX designing

This is one of the most critical stages in the development pipeline. A clean, responsive, and user-

friendly interface is essential to ensure a seamless user experience and maintain user engagement. The design phase focuses on creating layouts that are intuitive, accessible, and visually consistent with the overall theme of the system.



Justification

We used **Figma** as our primary design tool to plan and prototype the user interface. Figma allowed our team to collaboratively design and refine each component, ensuring proper alignment between functionality and aesthetics. The design process emphasized responsiveness across devices, clarity in navigation, and consistency in typography and color schemes.

A well-structured UI/UX design not only enhances usability but also supports the logical flow of agent interactions, helping users understand the system's features effortlessly and ensuring smooth communication between the user and the AI agents.

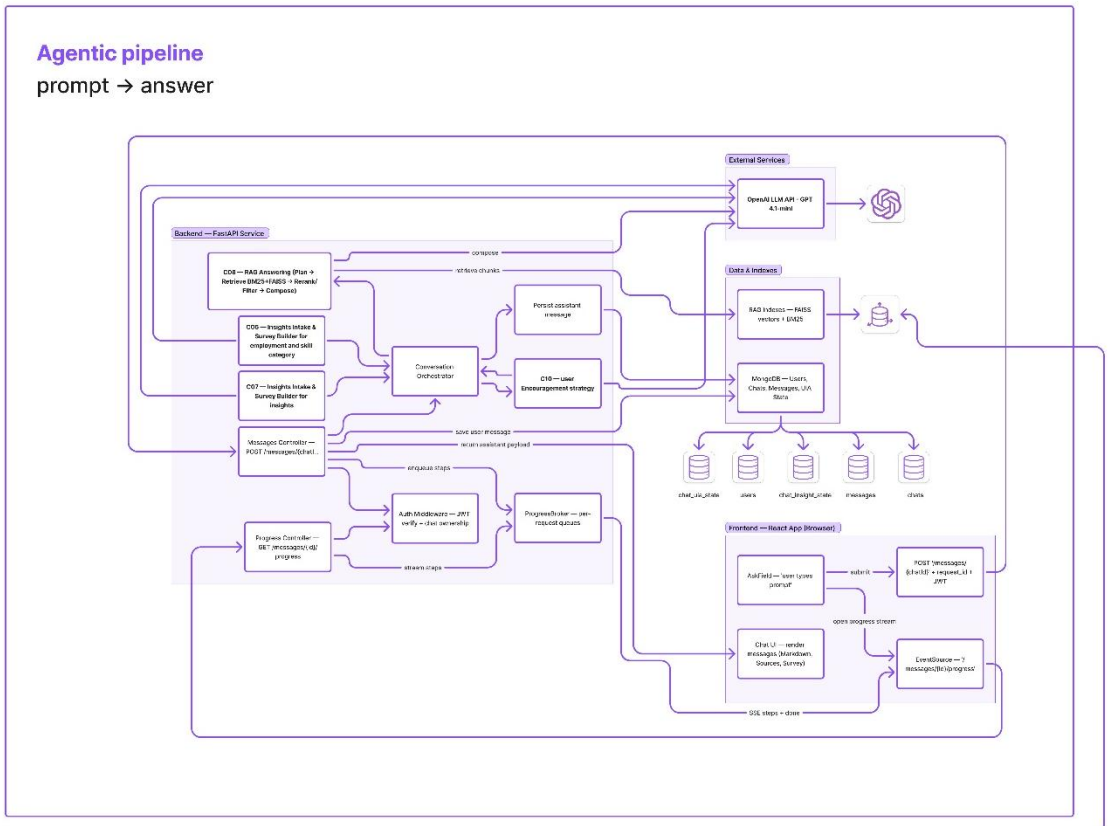
5. #section 05 : Frontend + Backend development

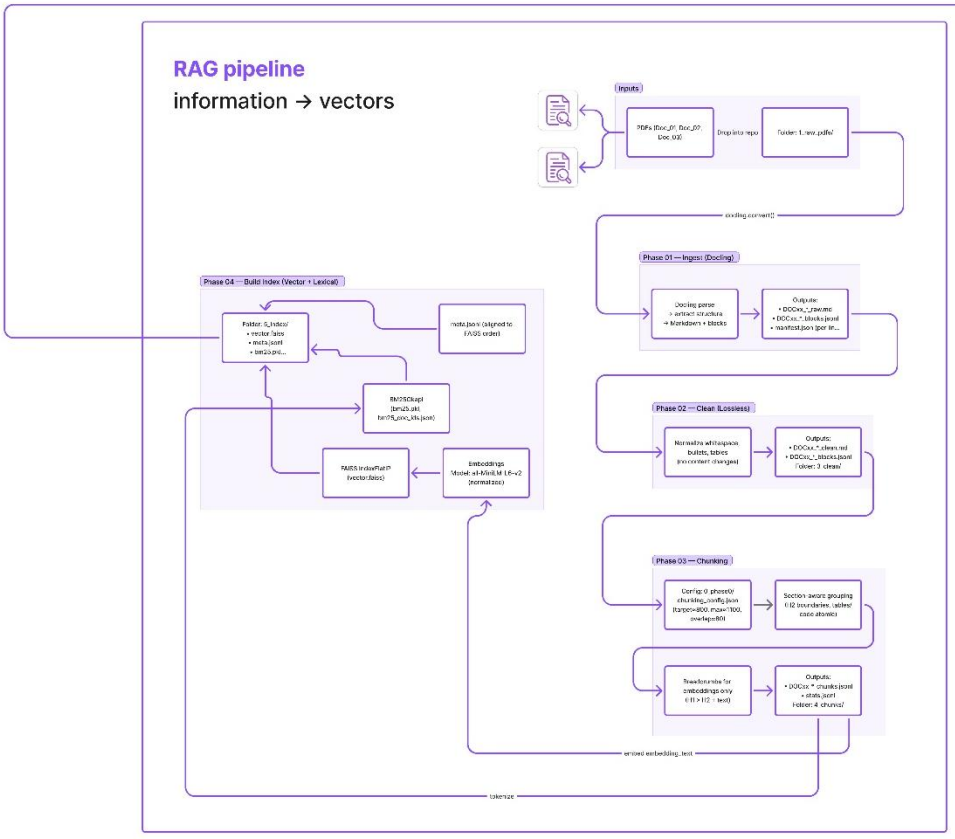
Main technologies & tools used: python, react, RAG kb, tailwind CSS, docling, MongoDB.

This section is composed of components which will be explained in detail later under the main system architecture diagram. These components were implemented periodically under defined time periods.

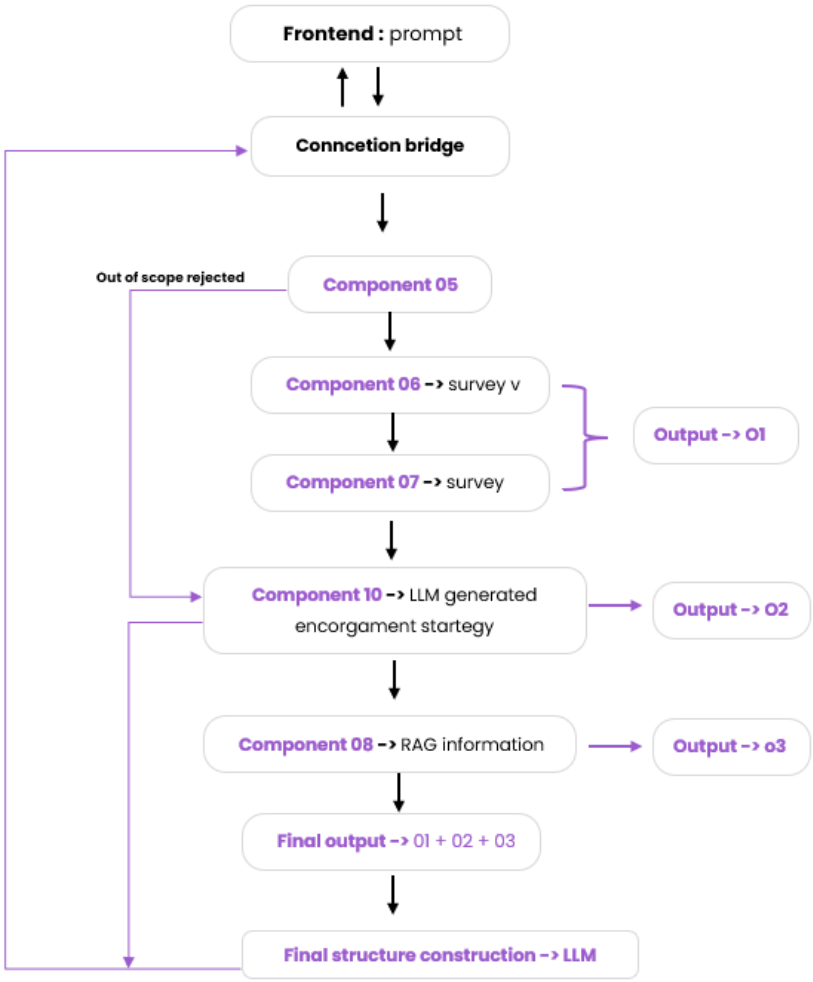
System Design Architecture (#2)

communication flow + system design





During the project development we designed a comprehensive architecture that could turn the idea into pipeline and into working model. Below is the simplest architecture diagram we first designed before drawing the main framework as shown above.



To make sure the requirements outlined which the final agent should perform happens as required, we divided the requirements into **components** and applied in the pipeline. Each main components will do a specific specialised function, and each component has decision gate which will decide whether to run the component based on what the used has given in their prompt. In this method we will orchestrate the 2 agentic system – UAA and UIA with each having a set of components that together delivers the function which the agents are responsible.

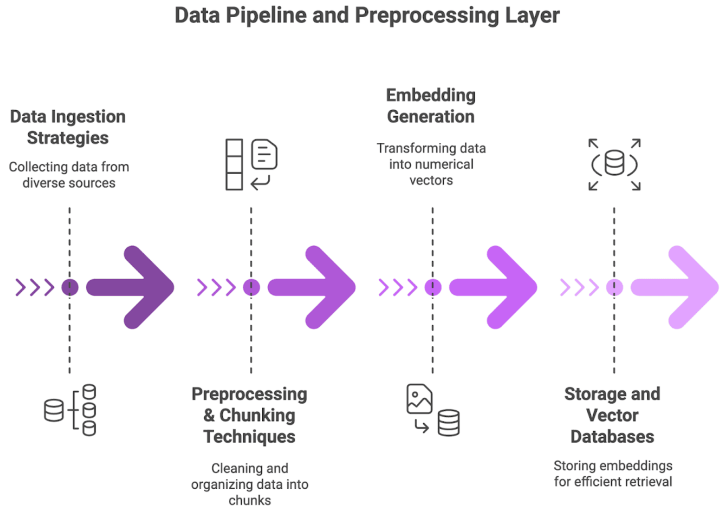
The system was divided into 2 main parts a agentic ai pipeline and RAG (retravel augmented generation) pipeline.

1) **Agentic AI pipeline**
   The main framework that defines the system from the time user gives an input prompt to until the response is produced.

2) **RAG – retrieval augmented generation.**
   The main knowledge base that will support users prompt with accurate up to date information.

Justification of the system -> RAG pipeline

Here we have described the process behind creating the knowledge base that takes the information needed later by the components 8 in the agentic AI pipeline. Consider the following diagram,



We carried out a research step that allowed us to collect information about possible question that users could raise in out platform. Based on this we created the knowledge corpus. Turned a set of PDFs into a **searchable knowledge corpus** that a downstream LLM can query. The corpus is stored as:

- **Structured text blocks** (paragraphs/lists/tables/code) with stable metadata (section paths, block indices).
- **Retrieval chunks** that preserve content (no rewriting) and keep tables/code atomic.
- **Hybrid indexes**:
    - a **vector index** (FAISS, cosine) built over each chunk's embedding_text (breadcrumb H1 > H2 + content), and
    - a **lexical index** (BM25) over the same text.

Everything is file-based and reproducible:

- 1_raw_pdfs/ → input PDFs
- 2_docling/ → Docling markdown + block JSON + manifest.json
- 3_clean/ → lossless-clean markdown + normalized blocks
- 4_chunks/ → chunk JSONL with text (clean), embedding_text (breadcrumbs+text), section_path, section_group_id, deterministic chunk_id
- 5_index/ → vector.faiss, meta.jsonl (aligned to FAISS row order), bm25.pkl, bm25_doc_ids.json, index_config.json

This corpus is what your Phase-05 "RAG runtime" uses to plan queries, retrieve, rerank, check relevance/coverage, and compose a **clean, structured** answer (no inline bracket IDs; citations are shown separately).

1) Data Ingestion Strategies

Goal: get reliable, versioned text from PDFs.

- **Inputs**: your PDFs (e.g., Doc_01 Data Science, Doc_02 Insights, Doc_03 Agentic system …).

- **Loader**: **Docling** parses each PDF into:
    - Markdown (structure preserved),
    - *_blocks.jsonl (one block per line with block_type, heading_level, section_path, block_index, text),
    - manifest.json (one line per doc: doc_id, filename, page_count, checksum, timestamps).
- **Versioning & naming**:
    - Documents are named DOCxx and carry a date/version in filenames (e.g., DOC01_20251014_*).
    - Checksums recorded for integrity.
- **Outputs** (folder): 2_docling/

Quality gates

- If Docling fails or page_count=0, the doc is flagged before any downstream step.
- The ingest is idempotent: reruns produce the same outputs for the same file.

2) Preprocessing & Chunking Techniques

Goal: normalize text without altering content; build retrieval-friendly chunks.

Preprocessing (Phase-02: "lossless clean")

- Normalize whitespace, bullets, page breaks; unify tables' fence markers.
- **No semantic edits** — the markdown body remains byte-for-byte equivalent in meaning.
- Outputs: 3_clean/DOCxx/*_clean.md and updated *_blocks.jsonl (same block indices).

Chunking (Phase-03: section-aware, content-preserving)

- **Strategy**:
    - Group only **whole blocks** (paragraphs/lists) into chunks; **never split tables or code** (each is atomic).
    - Use **H2 boundaries** to keep subtopics coherent (break_on_heading_level=2).
    - Keep headings out of visible chunk text to avoid content mutation; preserve them in metadata (section_path).
    - Build **embedding_text** = breadcrumb (H1 > H2) + original text — used only for embeddings; the visible text stays pristine.
    - Small **overlap** (e.g., 80 tokens) between adjacent text chunks to avoid mid-thought cuts.
- **IDs & traceability**:
    - Every chunk has a deterministic chunk_id that encodes doc_id, version, block span, and a short hash.
    - section_group_id lets retrieval pull neighbors from the same subheading when needed.
- **Outputs**: 4_chunks/DOCxx/*_chunks.jsonl + 4_chunks/stats.jsonl

Why this works: you retain exact source text for quoting/citations, but embeddings "see" the semantic breadcrumb so retrieval understands where a paragraph sits in the document hierarchy.

3) Embedding Generation

Goal: turn each chunk into a dense vector for semantic search.

- **Source field:** embedding_text (breadcrumb + text) — improves recall without changing what users see.

- **Model:** Sentence-Transformers all-MiniLM-L6-v2 by default (configurable later; you can rebuild with OpenAI text-embedding-3-large if desired).
- **Normalization:** vectors are L2-normalized so **cosine similarity = inner product**.
- **Batching:** batched encoding; dimension recorded in index_config.json.

Outputs: in memory as numpy arrays → then written to FAISS (see next section).

4) Storage and Vector Databases

Goal: persist hybrid indexes that are fast, portable, and aligned to metadata.

- **Vector store: FAISS IndexFlatIP** (5_index/vector.faiss)
  - Stores normalized embeddings; simple and portable (cosine via inner product).
- **Lexical store: BM25Okapi** (5_index/bm25.pkl) over the same texts; bm25_doc_ids.json maps positions → chunk_id.
- **Metadata sidecar:** 5_index/meta.jsonl (one line per FAISS row)
  - Fields: chunk_id, doc_id, version, section_path, breadcrumb, section_group_id, chunk_type, token_count.
  - **Order matches FAISS add order** so results can be resolved back to chunks.
- **Config:** 5_index/index_config.json (model name, dim, normalization, build time).

Why hybrid?
Lexical (BM25) excels on exact terms, acronyms, and rare tokens; vectors catch paraphrases and semantics. You fuse them (RRF) in retrieval for robust performance.

Justification of the components -> Agentic AI pipeline

component 05 : Decision gate

What it does (purpose)

C05 is the **first checkpoint** in /send. It decides whether the incoming user turn is **in-scope** for the User Analysis Agent (UAA) or **out-of-scope**. If it's in-scope, the pipeline continues (C06 → C07 → C10, etc.). If it's out-of-scope, the gate returns a **friendly boundary message** (and optionally re-asks the last encouragement), and the rest of the components are skipped for that turn.

Key inputs

- **User message** (the new turn).
- **Context from prior assistant turns** (fetched from MESSAGES):
  - the **last actionable assistant prompt** (an earlier **encouragement question** and/or an **active survey**),
  - excluding any messages labeled scope_label="out_of_scope" (so an earlier encouragement isn't hidden by a later boundary message).
- **Heuristics & LLM prompt**:

- lightweight keyword checks (e.g., to treat general DS questions as in-scope, and code/ops requests as out-of-scope),
- the **Decision Gate** LLM prompt (JSON-only contract) with:
  - IN-SCOPE / OUT-OF-SCOPE criteria,
  - a **priority rule** (execution/coding wins → out-of-scope),
  - **special handling** for "about the system" questions (returns a short explainer),
  - instructions to **re-ask the prior encouragement** when the user goes off-track.

Decision pipeline (ladder)

1. **Recover in-flight prompt**
   Find the most recent assistant message that is **not** out_of_scope and **has either** an encouragement question or a surveyType.
   This is the "in-flight" ask the user might be answering.
2. **Short-circuit rules (no LLM)**
   - If a **survey** is currently shown → **proceed = true** (user is interacting with our survey).
   - If an **encouragement question** exists and the user **appears to be answering it** (e.g., mentions one of the canonical options) → **proceed = true**.
   - If an encouragement exists **but** the user asks for **execution/coding/ops** → **proceed = false**, return a **boundary message + re-ask** the exact encouragement (with canonical option labels), and label this message as out_of_scope.
3. **Pre-LLM safety net**
   - If the message clearly looks like **Data-Science knowledge** (e.g., "why is learning data science important?", "explain k-means") and **doesn't** request execution → **proceed = true**.
4. **LLM classification (JSON-only)**
   - Provide the **last assistant text** and the **last encouragement question** in context.
   - The LLM outputs:
     - {"proceed": true} (in-scope), or
     - {"proceed": false, "message": "<friendly boundary …>"} (out-of-scope).
   - For out-of-scope, the message:
     - states the boundary and what the agent *does*,
     - **re-asks** the last encouragement (if one exists) with **canonical labels** preserved,

- for **"about the system"** queries, returns a **richer explainer** (2–4 sentences) rather than a bare boundary.

Outputs

- **Gate decision**: proceed: true/false.
- **If false**: a **single friendly message** (optionally with the **re-ask** appended).
  (On storage, it's labeled scope_label="out_of_scope" so it doesn't mask the next actionable prompt.)

How it shapes the rest of the pipeline

- **proceed = true** → continue to C06 (intake), then C07 (insights), then C10 (encouragement).
- **proceed = false** → only the gate's message is returned to the frontend for that turn (no surveys, no new encouragement).

Edge-case handling

- **User goes off-topic after we asked something** → out-of-scope + **polite boundary** + **re-ask** the exact encouragement (keeps the user on track).
- **User returns with a short answer later** (after an OOS bubble) → gate looks past the OOS bubble to the last **actionable** prompt and lets it through.
- **Ambiguous "about" vs DS knowledge** → precedence to **in-scope DS knowledge** (e.g., "why is learning data science important?" is in-scope).

---

component 06 : 2-stage segment collection. (UIA part 01)

What it does?
Goal: identify the user's **employment category (EC)** and (optionally) their **skill focus** for this chat, using a shared **Segment Vault** and two lightweight surveys. It writes only to **per-chat state** and returns either a survey to the UI or a short acknowledgement text. It does **not** generate the final domain answer—that happens in later components.

Core responsibilities

- **Read the Segment Vault** (authoritative list of employment categories and their skills).
  → Only categories defined in the vault are accepted. Any category not in the vault is ignored.
- **Classify the user message** with an LLM:
  - employment_intent — true if the message names/implies a job/role **or** discusses a professional field (e.g., "tell me about data science").
  - skills_intent — true when the user asks to choose/improve skills or requests a learning plan/roadmap.
  - ec_hit — a strict "exact role hit" (e.g., only set to "ec_ds" when "data scientist" as a title is explicitly mentioned).
- **Capture employment category**:

- If the message explicitly names a vault category (exact hit), record it **once** in the per-chat state.
- If employment intent is present but category is unclear, **ask the Employment Survey** (single-select across vault ECs).

- **Capture skills** (only if EC is already known):
  - If the message asks about skills, **ask the Skills Survey** for that EC (pick **1-4** skills or **let system decide**), recorded **once** per chat.
- **Guardrails**:
  - **Never overwrite** an employment category already stored for this chat.
  - **Never show** a skills survey again if the chat already has skills recorded (manual or system-decide).
  - If a survey message was already **submitted**, the UI shows that state on reload (thanks to writing the submission back into the same message document).
- **Output**:
  - If a survey is needed: return an assistant **survey message** (and stop the rest of the turn).
  - Otherwise: return a small assistant **text** acknowledgement (e.g., "Noted your employment category: **ec_ds**.").

Pipeline for component 06

1) Vault and intent recognition

Load the active Segment Vault

- Retrieve the currently active vault version and indexes of categories and skills.
- Emit progress (e.g., "Loading active vault").

Run intent detection on the user message

- Extract three signals:
  - **employment_intent** — true if the message names or discusses a job/role **or** a professional field/industry.
  - **skills_intent** — true if the user asks to choose/learn/prioritize/improve skills or requests a skill plan/roadmap/course.
  - **ec_hit** — a strict match to a **specific employment category** in the vault (e.g., "data scientist" → ec_ds).
- Emit progress (e.g., "Detecting intent (LLM)").
- Optionally emit a dynamic line if a category was explicitly recognized (e.g., "Captured employment category = ec_ds").

2) Read current per-chat UIA state

Fetch per-chat state

- Look up whether an employment category is already recorded.
- Check whether skills are already recorded (either a chosen list or "let system decide").
- Emit progress (e.g., "Checking chat state").

Guardrails enforced from here on:

- Never overwrite an existing employment category for this chat.
- Never show the skills survey again if skills were already recorded.

3) Decide the employment path

If an employment category is not yet recorded:

- **Case A — Explicit category detected:**
  If ec_hit maps to a vault category, record that category once in per-chat state.
  Emit a dynamic progress line (e.g., "Recorded EC = ec_ds").
  Continue to the skills decision (do not ask an employment survey).
- **Case B — Employment intent but category unclear:**
  Build an **Employment Survey** (single-select across all vault categories, labeled and described).
  - Persist a single **assistant survey message** containing:
    - A clear prompt (e.g., "Please select your employment category").
    - Survey metadata (vault version, list of {id, label, description}).
  - Stop the turn here and return this survey message. (Later components are skipped for this turn.)

4) Decide the skills path

If employment category is recorded and skills are not recorded yet:

- **If skills_intent is true:**
  Build a **Skills Survey** tailored to the chosen category:
  - Multi-select with limit (1–4) and an optional "let system decide" choice.
  - Persist a single **assistant survey message** with that payload.
  - Stop the turn here and return this survey message.
- **If skills_intent is false:**
  No skills survey is needed now; continue to acknowledgement.

5) No survey needed → produce acknowledgement

Prepare and save a short acknowledgement

- If you just recorded the EC in this turn: reply like "Noted your employment category: **ec_xxx**."
- Otherwise: reply politely that no identification action was needed.
- Persist a single **assistant text message** with that acknowledgement.

Emit completion

- Send a final progress line (e.g., "Preparing response") and then a "done" event on the SSE stream.
- Return the assistant message.

6) Survey submission lifecycle (separate requests from the UI)

Employment Survey submit

- Frontend posts: chat id, the message id of the survey, chosen category, and vault version.
- Backend actions:
  - Validate against the vault and **write once** to per-chat state (if already set, reject or no-op).
  - Reflect the submission back into the same survey message, marking:
    - survey.isSubmitted = true
    - survey.submitted = { employment_category_id, vault_version, submittedAt }
  - Respond with a small status object (e.g., action: recorded_ec).
- Effect: On page reload, the survey appears "Submitted," with the selection visible and inputs disabled.

Skills Survey submit

- Frontend posts: chat id, the message id of the survey, vault version, employment category id, and either:
  - let_system_decide = true, or
  - a list of up to 4 selected skills.
- Backend actions:
  - Validate and **write once** to per-chat state (record chosen skills or the system-decide flag).
  - Reflect into the same survey message:
    - survey.isSubmitted = true

        - survey.submitted = { mode: "manual" | "system_decide", skills_selected?, employment_category_id, vault_version, submittedAt }
      - Respond with a small status object (e.g., action: recorded_skills, mode, skills_count).
    - Effect: On reload, the skills survey shows submitted choices (or "system decide") and is locked.

7) Persistence: what is written where

- **Messages collection**
  - User message (always).
  - Assistant survey or text (one message per turn).
  - On survey submit: the **same survey message** is updated to include the submission data and isSubmitted = true (so UI can hydrate on reload).
- **Per-chat UIA state**
  - employment_category_id — set once.
  - skills_selected (1–4) or let_system_decide — set once.
  - vault_version — stored with the state for audit/validation.

---

component 7 : UIA for insights (UIA part 02)

What it does (purpose)
From a **free-text user prompt**, the component figures out which predefined **insight questions** (and, when clear, which **answers**) apply to this chat. It writes those decisions to the **user insight states in mongo DB** and then builds **targeted micro-surveys** to finish anything still unanswered.

Key behaviors (guarantees):

- **LLM-only inference:** No retrieval stack needed. You pass the complete (small) **Vault Pack** (your 1–2 batches, each with 1–5 insights) into a single LLM call; the model returns **only matched insights** with a confidence and short evidence quote(s).
- **Strict thresholds:**
  - Auto-take (mark answered) when confidence ≥ **0.75**.
  - Question-only when confidence ≥ **0.60**.
- **Batch Expansion (MUST):** If **any** insight in a batch is touched in Stage-01, the component pre-creates pending rows for **every other active insight in that batch** (except already taken). This ensures Stage-02 will ask **the entire batch** (minus already-taken items).
- **Surveys are per touched batch:** Each touched batch with pending items gets **one** micro-survey containing **all pending** insights in that batch (both "question_only" and "batch_fill").

- **Survey wins on conflict:** If a user's survey answer disagrees with Stage-01 auto-inference, the survey answer **overwrites** the auto decision for this chat.
- **Vault is read-only;** all state is written **per chat** (PCCH), idempotent upserts keyed by {chatId, insightId}.

Stage 01 — Auto-Inference (LLM-only)

Goal: read a messy user prompt and decide, for each insight in the vault, whether the prompt:

- clearly states the question **and** one listed answer (**QUESTION_AND_ANSWER**),
- directly states one listed answer or alias without restating the question (**ANSWER_ONLY**),
- talks about the question's topic but no listed answer is clear (**QUESTION_ONLY**).

We do **not** return non-matches.

Step 1 — Initialize this chat's state

- Ensure there's a uia insight state shell for the chat.
- Note which insights are already answered for this chat and which are already pending (from previous turns).

Step 2 — One LLM pass over the whole vault

- Provide the user's prompt together with the full vault (both small batches).
- The LLM returns **only the insights that actually match**, each with:
  - the match type (from the three above),
  - the matched answer id (or "none" for question-only),
  - a confidence score (0–1),
  - short evidence quotes from the prompt.

Rules enforced in the prompt: use only the vault's options/aliases; be strict; prefer **QUESTION_ONLY** over guessing; ignore negated/obsolete statements like "not a problem anymore".

Step 3 — Write decisions into the uia insight state

- For each matched insight:
  - **Auto-take** (mark answered) when the match is **QUESTION_AND_ANSWER** or **ANSWER_ONLY** and the confidence is **≥ 0.75**.
    Record taken = true, the chosen answer(s), confidence, evidence, and source = "auto-inference".
  - **Question-only** when confidence is **≥ 0.60**.
    Record taken = null, pendingReason = question_only, with confidence and evidence.
- Any batch that had at least one such decision is added to the chat's **touched batches** list.

Step 4 — Batch Expansion (must)

- For every **touched batch**, pre-create pending rows for **all other active insights in that batch** that are not already answered or pending:
  - mark them taken = null with pendingReason = batch_fill,
  - source = "batch-expansion" and confidence = 1.0 (it's a structural pending, not a guess).
- Result: Stage 02 will ask **the entire batch** for this chat, except items already answered.

Step 5 — Handoff summary

- For this chat, prepare a simple summary:
    - the set of **touched batches**,
    - for each touched batch, the list of **pending insights** (question_only + batch_fill).

Stage 02 — Targeted Micro-Surveys (per touched batch)

Goal: for each touched batch that still has pending insights, show the user a short survey to collect answers, and write those answers back to the uia insight state.

Step 1 — Employment category gate

- If this chat doesn't have an employment category recorded, pause surveys and capture it (infer with high confidence or ask once).
- If it's present, continue.

Step 2 — Select which batches to survey

- Start from the **touched batches**.
- A batch is selected if it has **any** pending insights (taken = null) for this chat.
- Batches with no pending items are skipped.

Step 3 — Build the survey for each selected batch

- For that batch, include **all** pending insights (both question_only and batch_fill).
- Use the **vault's** canonical wording and options. You may lightly rephrase the question text for clarity, but do not change IDs or option labels.
- **Order inside the survey:**
    1. all question_only items first (they're closest to resolution),
    2. then batch_fill items,
    3. break ties by higher confidence and, if needed, the vault's display order.
- Include an optional "Other (write-in)" text box if you want to collect a note; this does not create a new option.

Step 4 — Collect answers

- Single-select insights are shown with radio buttons; multi-select with checkboxes.
- The user submits one batch at a time.

Step 5 — Write survey results to the uia insight state

- For each answered insight in that submission:
    - validate the answer is a valid option for that insight,
    - set taken = true and store the chosen answer id(s),
    - clear any pending reason,
    - record source = "survey", confidence = 1.0.

- **Conflict policy:** if a Stage-01 auto-take previously set a different answer for this chat, the **survey answer overrides it**.

Step 6 — Completion bookkeeping

- Recompute simple stats for this chat: how many insights are answered and how many remain pending.
- Optionally mark a batch "complete for this chat" when it has no pending items left.
- Return a short success summary to the UI.

---

component 8 : RAG retrieval of Knowledge base (UAA part 01)

What it does (purpose)
C8 is mainly responsible for retrieving the information form the vector DB we created in faiss which are needed as information for the users given prompt.

1) Query intake & planning

Goal: convert one user prompt into a small plan that optimizes retrieval and presentation.

- **Input:** user question.
- **Action (LLM):** generate 2–4 focused **sub-queries** and a **style plan** (style/tone/format/audience).
  (Function: llm_plan_queries())
- **Doc routing:** if you keep doc_filters, sanitize them (or set to "all docs" as you decided).
- **Output:** {queries: [...], style/tone/format/audience, doc_filters?}.

2) Hybrid retrieval (for each sub-query)

Goal: get broad, high-recall candidates.

- **Vector search:** FAISS on **embedding_text** (cosine via normalized vectors).
- **Lexical search:** BM25 on the same text.
- **Fusion:** Reciprocal Rank Fusion (RRF) per sub-query → then **pool** across all sub-queries.
  (Function: hybrid_search_multi(..., allow_docs=...))

Output: ranked list of candidate chunk_ids (broad recall).

3) LLM re-ranking

Goal: keep only the best-matching candidates for the question.

- **Input:** top candidates (breadcrumb + 400-char snippet each).
- **Action (LLM):** choose the best ~8–12 chunks.
  (Function: llm_rerank())
- **Output:** curated chunk_ids.

4) Relevance gate (strict filter)

Goal: throw out tangents/duplicates before context building.

- **Input:** question + short snippets for the curated set.
- **Action (LLM):** return {keep, drop}; keep only directly useful chunks.
  (Function: llm_relevance_filter())
- **Output:** relevance-clean chunk_ids.

5) Coverage check (sufficiency gate)

Goal: decide whether RAG alone is enough.

- **Input:** question + brief summaries of kept chunks.
- **Action (LLM):** score sufficiency (0–1) and list missing_aspects.
  (Function: llm_sufficiency_gate())
- **Policy:**
  - sufficiency ≥ 0.7 → answer **strictly from RAG**.
  - else, if RAG_ALLOW_GENERAL_KNOWLEDGE=true → allow a **small** "Background (general)" supplement capped by RAG_MAX_GENERAL_PERCENT.
  - else → answer with what's available and state gaps.

6) Context packing

Goal: assemble a single context string under budget with provenance.

- **Input:** final chunk_id list.
- **Action:** concatenate [chunk_id] breadcrumb\n<text> blocks with separators; keep under token budget.
  (Function: pack_context())
- **Output:** context_str + the list of included chunk records (for citations).

7) Answer composition (style-aware, clean)

Goal: produce a grounded, well-structured answer that matches the user's requested format.

- **Input:** question, style plan, context_str, sufficiency decision.
- **Action (LLM):** compose **Markdown** with house style:
  TL;DR → Key Points → Details (+ steps/table/examples as requested).
  No inline bracketed IDs; if general info is allowed/needed, add a final **"Background (general)"** sub-section and keep it under the % cap.
  (Function: llm_answer())
- **Output:** draft answer.

8) Validation (optional safety pass)

Goal: catch off-topic or contradiction before showing the user.

- **Input:** question + evidence summaries + draft.
- **Action (LLM):** return on_topic/contradiction and an optional revision.
  (Function: llm_validate())
- **Output:** final answer.

10) Presentation

Goal: render clearly with traceability.

- **Answer:** print/render the clean Markdown body.
- **Citations:** list [chunk_id] breadcrumb from the included chunks (no noisy inline markers). *(Your CLI/UI already does this.)*

---

component 10 : User encouragement strategy (UAA part 02)

What it does (purpose)
C10 is the **gentle nudge** that keeps users moving through the user-analysis funnel when no survey is being shown on the current turn. It asks **one creative, persuasive question** that helps the system collect the next piece of needed data in the pipeline:

Employment Category → Skills → Insights

Key inputs

- **C06 (intake) result**: whether an **EC** or **skills** survey is available/just shown; uia_action values (show_ec_survey, show_skills_survey, …).
- **C07 (insights) result**: whether an **insight survey** was prepared this turn (surveysPrepared, touchedBatchIds, pendingByBatch).
- **Chat state**: employment_category_id (EC current), skills_selected or let_system_decide (skills done).
- **Vault options**:
    - list_ec_options(version) → valid EC list,
    - list_skill_options_for_ec(version, ec_id) → canonical **skill categories** for that EC,
    - InsightVault.list_batches_in_order() → ordered **insight batches** and each **insight** with **canonical answers** (A,B,C…) and isMultiSelect.
- **User's current message**: to tune tone and relevance of the nudge.

Decision pipeline

C10 emits **exactly one** question per turn (or none), depending on what's missing and what other components already surfaced.

Stage 1 — Employment Category (EC)

Goal: If we don't know the EC and there's no EC survey being shown, nudge the user to pick their EC.

Skip if:

- ec_current exists **or**
- C06's action is show_ec_survey (survey will gather EC).

If not skipped:

- Pull the **canonical EC options**.
- Reflect any cues from the user's message (e.g., "I'm into NLP") but **do not** invent options.
- Ask a **creative, low-friction** question that **invites a choice**; keep it short and engaging.

Output example (shape):

- stage: "employment_category"
- question: "<creative one-liner inviting them to pick EC (options hinted naturally)>"

Stage 2 — Skills

Goal: If EC is known (or an EC survey just ran), and **skills** aren't recorded and there's no Skills survey this turn, nudge for **1–4 skill categories**.

Skip if:

- skills already recorded (skills_selected not empty or let_system_decide true) **or**
- C06's action is show_skills_survey.

If not skipped:

- Pull **canonical skill categories** for the **current EC**.
- Ask a **persuasive, focused** one-liner inviting the user to pick **a few**; **preserve labels** so downstream detection is reliable.

Output example (shape):

- stage: "skills"
- question: "<creative one-liner inviting 1–4 categories; canonical labels woven in naturally>"

Stage 3 — Insights

Goal: Once EC & skills are set, gather **pain points/weaknesses** by nudging toward the **next eligible insight batch**—but **only** if no insight survey is shown this turn.

Skip if:

- C07 prepared an insight survey (surveysPrepared > 0) **or**
- The next batch is already **touched** this turn, **or** fully **completed** (in complete_batches).

If not skipped:

- Iterate batches in **defined order** (Batch 1 → 2 → …); pick the **first** batch that:
    - is **not** in touchedBatchIds, and
    - is **not** fully completed.

- Within that batch, choose one **insight** to nudge on.
- Build a **creative, natural** question that:
    - **mentions the insight plainly**,
    - **includes the canonical answer labels** in a subtle way (so C07 can auto-detect on the next turn),
    - respects isMultiSelect (e.g., “choose any that apply”).

Output example (shape):

- stage: "insights"
- question: "<creative one-liner about the selected insight; canonical answers included>"
    - (e.g., “When you’re stuck, what helps first — Search docs/examples, Restate/simplify, Sketch a diagram, Compare to a prior pattern, or Ask for a targeted hint?”)

Output (to the UI)

- **Only one** encouragement per turn, with:
    - stage ∈ {employment_category, skills, insights}
    - question ∈ creative, concise one-liner
- It’s surfaced as a **separate bubble** (the UI shows up to three bubbles: content, survey, encouragement).

When C10 does nothing

- If **any** survey is being shown this turn (EC or Skills via C06, or Insight via C07), C10 is typically **suppressed** for that turn to avoid cognitive overload.
- If all three stages are already satisfied (EC known, skills recorded, insight batches either in progress via survey or completed), C10 **stays quiet**.

Working pipeline: prompt -> answer

0) User action → request enters /send

- The user types a prompt in the chat box and hits send.
- The frontend immediately:
    - Adds a **user bubble** with that text.
    - Creates a single **assistant “progress” bubble** (loader) and opens an SSE stream to receive step updates (e.g., “Queuing request”, “Decision Gate”, “RAG: composing”…).

1) Server receives the turn (setup & persistence)

- The backend verifies chat ownership.

- **Persist user message** into MESSAGES with basic fields (role=user, content, timestamps).
- A request-id ties the turn to SSE progress updates.

2) Component 5 — Decision Gate (scope check)

Purpose: decide whether this turn is "in scope" for the User Analysis Agent.

Inputs it gathers

- The **current user message**.
- The **last actionable assistant message** (not labeled out_of_scope), which may carry:
  - an **encouragement question** (enc_question), and/or
  - a currently visible **surveyType**.
- Light **heuristics**:
  - If the user is clearly answering our last encouragement (mentions canonical options), treat as in-scope.
  - If the user asks for coding/execution/ops, lean out-of-scope.
  - General DS knowledge questions (e.g., "why is learning DS important?") are treated as in-scope.

LLM classification

- If needed (i.e., heuristics didn't settle it), the gate calls its JSON-only prompt with:
  - agent purpose,
  - in-/out-of-scope criteria & priority rule,
  - the last assistant message and last encouragement (for context),
  - special handling for **"about the system"** questions (returns a friendly, multi-sentence explainer).

Outcomes

1. **Proceed = true** → pipeline continues.
2. **Proceed = false** → the gate generates one friendly boundary message.
   - If we had a pending encouragement, it **re-asks** it concisely (canonical options preserved).
   - The assistant message is labeled internally as **out_of_scope** so it won't block future "last actionable" lookups.
   - The server **stops** the rest of the components for this turn and returns the single content bubble.

3) Component 6 — Intake (Employment Category & Skills)

Purpose: decide whether to emit an **EC survey** or a **Skills survey**, or do nothing this turn.

What it looks at

- The user's message (intent detection).
- The per-chat **UIA state**: employment_category_id, skills_selected, let_system_decide.

Decisions

- If EC isn't set and conditions match: **build an EC survey** (canonical EC options).
- Else if EC is set but skills aren't and conditions match: **build a Skills survey** (canonical skill categories for that EC).
- Otherwise: no survey from C06 this turn.

Frontend effect

- If C06 produced a survey, it will appear as its own **survey bubble** (either EC or Skills). Only one survey is ever shown at a time.

4) Component 7 — Insight Engine (Auto-inference & Surveys)

Purpose: gather **insights** (pain points/weaknesses) via auto-inference and, when allowed, via an Insight survey.

Stage-01 (always runs): Auto-inference

- Parses the user message for high-confidence matches to known insights/answers.
- Records:
  - **auto-taken** answers,
  - **question-only** hints,
  - **touchedBatchIds** (which insight batches were implicated),
  - **pendingByBatch** (which insights remain within those batches).

Survey gating

- **Prerequisite:** only attempt building an Insight survey if **EC exists** and **skills are recorded**.
- If prereqs fail → it returns surveysPrepared = 0 and a clear skip reason.
- If prereqs pass → builds an **Insight survey**:
  - Honors **batch order** (Batch 1 → 2 → …).
  - Includes questions and **canonical answer labels** (plus "Other" where defined).
  - Respects single vs. multi-select.

Frontend effect

- If an Insight survey is built this turn, it's rendered as a **full-width survey bubble** (distinct from EC/Skills).
- If not, C07 still updates internal state (auto-taken, touched batches) for future steps.

5) Component 8 — RAG (knowledge answer)

Purpose: produce a grounded answer (Markdown) to the user's question, using your local retrieval index.

What it does

- **Plans** focused sub-queries (LLM).
- Runs **hybrid retrieval** (FAISS vectors + BM25), **RRF** fusion.
- **LLM reranks** candidates; **LLM relevance filter** trims off-topic.
- **Packs context**; runs a **sufficiency gate** (limits general knowledge if evidence is thin).
- **Composes** a clean **Markdown answer** (TL;DR, Key Points, Details).
- **Validates** on-topic/contradictions (LLM).
- Assembles a **sources list** (chunk_id + breadcrumb).

Outputs

- answer_md → becomes the assistant **content** bubble.
- sources → an array of origins for the Sources strip.

Frontend effect

- A standard **assistant content bubble** (Markdown), plus a **horizontal sources strip**:
  - Each source shown as a **small dark-ash chip**, max **20 chars** with …, horizontally scrollable, full title on hover.

6) Component 10 — Encouragement Strategy (when no survey shown this turn)

Purpose: keep users moving through the analysis funnel with **one** creative nudge when appropriate.

Pipeline

- **Stage 1: Employment Category**
  If EC is missing **and** C06 didn't produce an EC survey, ask a short, persuasive EC question (hints options naturally).
- **Stage 2: Skills**
  If EC exists but skills aren't recorded **and** C06 didn't produce a Skills survey, ask for **1–4** skill categories (canonical labels woven in).
- **Stage 3: Insights**
  If EC & skills exist **and** C07 didn't build an Insight survey this turn, select the **next eligible batch** and nudge on a specific insight; include **canonical answer labels** in the wording (so C07 can auto-capture on the next turn).

Frontend effect

- The nudge appears as a separate **encouragement bubble**.
- If any survey was shown this turn, C10 is typically **suppressed** to avoid overload.

7) Compose the assistant response (one payload, up to three bubbles)

The server now has everything it might show this turn:

- **Content** (Markdown) from C08 (or sometimes no content).
- **Survey** (from C06 or C07; **only one** survey per turn).
- **Encouragement question** (from C10; typically only when no survey is shown).

It composes one assistant "message" object with these fields:

- content (string; may be empty)
- surveyType (ec_survey | skills_survey | insight_survey | null)
- survey (structured data if a survey is present)
- enc_question (string; may be empty)
- sources (list from RAG; optional, shown under content)

Persistence

- Inserts this assistant message into MESSAGES, storing all present fields (including sources and optional scope_label for out-of-scope cases).

8) Frontend renders the final answer

- The SSE "progress" bubble is marked done and removed.
- The returned assistant message is **split into up to three visible bubbles**:
  1. **Content bubble** (Markdown from RAG).
     - If sources exist: a **scrollable chips strip** appears beneath it (dark ash chips; 20-char names; tooltip shows full).
  2. **Survey bubble** (if surveyType+survey present).
     - EC or Skills or Insight — never more than one in a turn.
  3. **Encouragement bubble** (if enc_question present).
- Bubble widths adapt to their own contents. Insight surveys render full-width to fit controls comfortably.

9) Next turn readiness (state continuity)

- UI survey submissions (EC/Skills/Insight) call their dedicated endpoints and produce small confirmation bubbles.
- The next user message will:
    - Re-run the **Decision Gate** (respecting in-flight prompts),
    - Continue surveys where left off,
    - Auto-infer insights from free text,
    - Possibly produce a new RAG content bubble,
    - And, if no survey is shown, **C10** will nudge the next missing piece.

2. Commercialisation plan

1) Who we are targeting (for the MVP)

Audience: People in the **data-science job market only**, in two employment states:

- **Unemployed/Job-seeking** (fresh graduates, career shifters).
- **Employed** (already working in a related role—e.g., analyst, intern, trainee—who want to move into data science).

Level segmentation (full product vision):

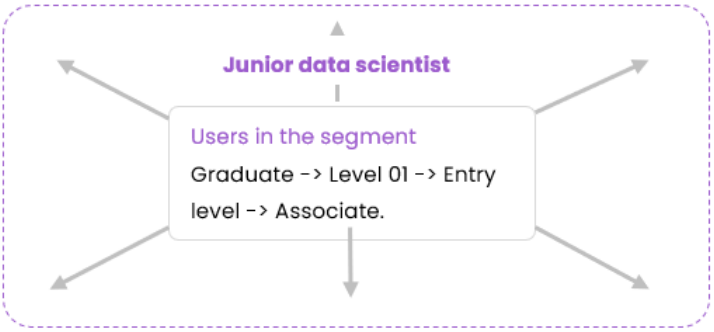
- **Junior**, **Average (mid)**, **Senior**.
- **MVP focus: Junior** category **only**.

This means the MVP is built to **take a person from Graduate → Level 01 → Entry → Associate → Junior**. Reaching **Junior** is the final milestone for this MVP.

2) Why this market now

The data-science/AI job market is competitive and fast-moving. Many newcomers have a **skills/experience gap** and struggle to prove job-ready ability. Companies want **practical skills + portfolio + interview readiness**. Our MVP solves exactly this for **Junior-level** readiness.

We have broken down this into 3 categories namely **junior, average and senior** levels. So basically, Since this is the MVP -> (minimum viable product) we will be focussing on junior category ONLY. For the data scientist field,



This is strictly related to data science job market ONLY. This will help to reach a person from Graduate - Level 01 - Entry level - Associate to get qualified to Junior level which is the final stage to look forward to jumping into average level by upgrading their skills in the field

Primary users

1. **Fresh Graduate (Unemployed)**
    - Needs: skill-gap map, guided path, ready-to-show projects, interview coaching.
2. **Career-Shifter (Unemployed)**
    - Needs: focused curriculum (no fluff), project portfolio relevant to target role, confidence-building practice.
3. **Working Analyst / Trainee (Employed)**
    - Needs: a clear upskilling plan around work hours, mentor hints, fast feedback loops, promotion/transition outcomes.

Pricing model

Following shows how we can distribute the cost the UAA + UIA platform,

Per-prompt cost structure (used for both plans)

Component	Cost per prompt (USD)	% of total cost
LLM API	$0.015	65.2%
Infra (hosting/vector/storage)	$0.005	21.7%
Support/overhead	$0.003	13.0%
Total	**$0.023**	**100%**

Free (Beta)

- **Included:** 25 prompts/month
- **Monthly cost at cap: $0.58**
    - LLM **$0.38** (65.2%) · Infra **$0.13** (21.7%) · Support **$0.08** (13.0%)

  Price: $0

Starter

- **Price: $9/month**
- **Included:** ~**150** prompts/month
- **Monthly cost at included usage: $3.45**
    - LLM **$2.25** (65.2%) · Infra **$0.75** (21.7%) · Support **$0.45** (13.0%)

  Estimated gross margin at included usage: $5.55

3. Responsible AI practises

Fairness

- Task-scope gating (Component 5):
  The Decision Gate blocks requests that push the agent into coding/ops or other unsafe roles, and treats short follow-ups to the assistant's own questions as inscope. This avoids uneven treatment driven by user verbosity and keeps assistance consistent across users.

- Neutral, user-led data collection (Components 6, 7, 10):
  1. EC/skills/insights are gathered via explicit surveys or encouragement questions, not inference alone.
  2. Surveys and nudges use canonical option labels (e.g., "Weekly plan", "Reading") and avoid suggestive phrasing, reducing leading bias.
  3. Insight batches proceed in a fixed order (Batch 1 → 2 → …), so different users don't get different questions due to hidden heuristics.

- No demographic profiling & minimal assumptions: The pipeline never asks for protected attributes or infers identity. It focuses on role/skills/pain points—factors relevant to learning—so guidance doesn't vary by sensitive attributes.

- Consistent error/edge handling: If a user goes off-track, the system shows the same friendly boundary and re-asks the prior question (preserving options), so users aren't penalized for one mistaken step.

Transparency

- **Explainable boundaries (Component 5):**
  Out-of-scope replies clearly state what the agent does (analysis & coaching) and what it won't do (write/run code). For "about" questions, it returns a **multi-sentence explainer** of purpose and scope.

- **UI separation of concerns:**
  Each assistant turn can show up to **three distinct bubbles**:

  1. **Content** (e.g., RAG answer)
  2. **Survey** (EC/Skills/Insights—only one at a time)
  3. **Encouragement question** (a friendly nudge)
     This makes it obvious *what* the system is doing and *what* it's asking from the user.

- **Provenance for knowledge answers (Component 8 RAG):**
  You expose a **scrollable strip of source chips** with tooltips (full titles on hover). That gives users line-of-sight into where information comes from.

- **Step-by-step progress (SSE):**
  The backend emits progress events ("Decision gate (LLM)", "Insights: Stage-01", "RAG:

composing", etc.), which can be surfaced to users—helping them understand the flow, not just the output.

- **Explicit state & gating in Insights (Component 7):**
  Insight surveys are built **only after** EC and skills are recorded; skipped with a clear "prereqs_not_met" reason. Batches mark **touched** and **completed**, so the user knows what's next.

Ethical handling of data

- **Access control & tenant isolation:**
  Every message is saved with user_id + chat_id, and routes verify **chat ownership** before reading or writing. That prevents cross-user leakage.
- **Data minimization by design:**
  You collect only what's needed—EC, skills, and insight responses—progressively, when relevant. No hidden scraping or background profiling.
- **Clear labeling of non-advisory content:**
  Out-of-scope messages are stored with scope_label="out_of_scope", ensuring they don't interfere with later question-answer collection and keeping the auditing trail clear.
- **Local retrieval corpus for RAG:**
  The RAG uses a **local FAISS/BM25 index** (no uncontrolled web calls during retrieval), reducing the risk of pulling in unvetted or privacy-sensitive external content.
- **Hallucination controls in RAG:**
  - **Sufficiency gate** estimates evidence coverage and limits "general knowledge" add-ons.
  - **Validation pass** checks on-topic/contradiction before finalizing the answer.
    These safeguards aim to keep answers faithful to sources.

4. Evaluation and results

Project evaluations, metrics and insights

This section shows how the requirements we have outlined for the agent UAA and UIA has been implemented in the main system. Refer to the following screen shots provided. Evaluations has been done component wise starting from,

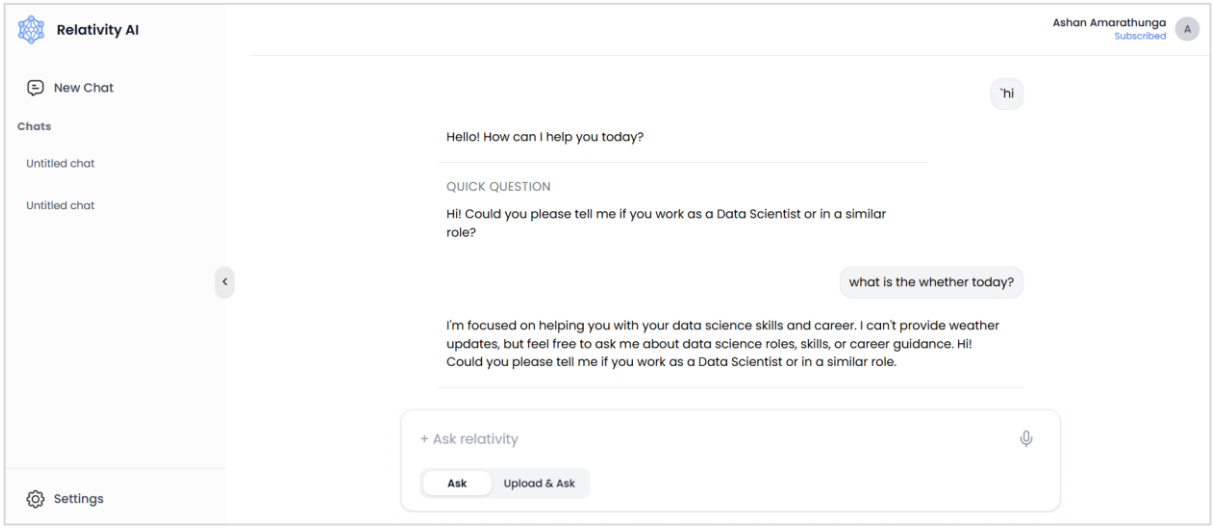
component 05 -> component 06 -> component 07 -> component 08 -> component 10. where comp. 06 and 07 deals with User Insight agent (UIA) and comp 08 and comp.10 deals with User Analysis agent (UAA). For each component we have outlined *what it should do? , result and evidence as success metrics* obtained from agentic system we have built.

Component 05 : Decision gate

what should it do?

C05 (Decision Gate) is the scope keeper: its goal is to decide if a user's message belongs in the User Analysis flow. It checks the message (and the last actionable assistant prompt) to see if the user is answering our question or asking DS-related guidance, then lets the pipeline continue, or, if it's off-scope (e.g., coding/execution), it returns a friendly boundary and briefly re-asks the prior encouragement to guide the user back.

Results



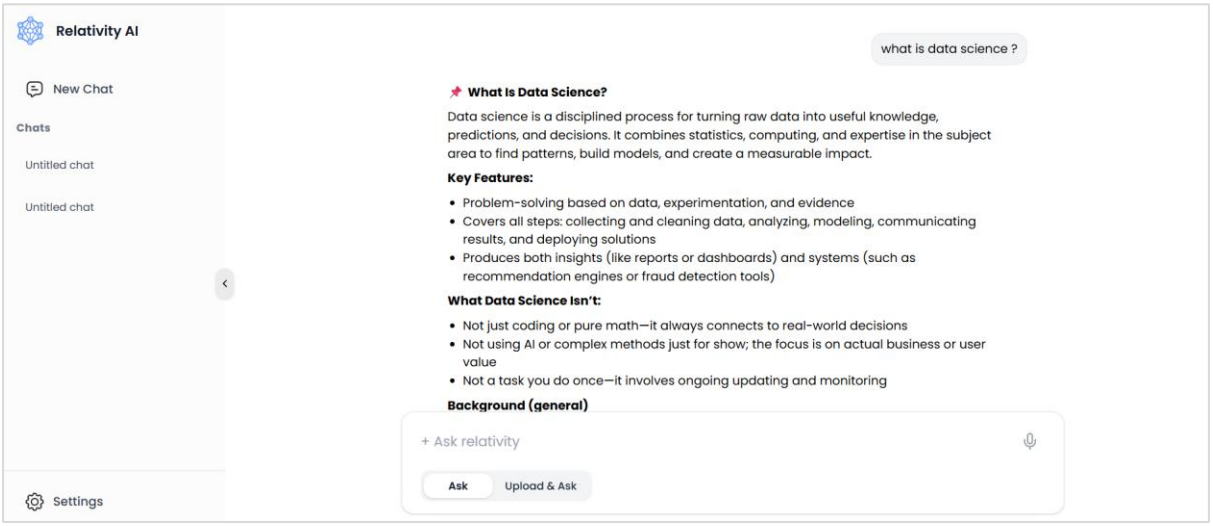
Success metrics : for the given prompt we can clearly see that for a question regarding whether has not been taken into account. Which means the decision gate will decide if an prompt should be taken into account or pass it which saves the rest of the pipeline from spending unnecessary computational + LLM calls costs.

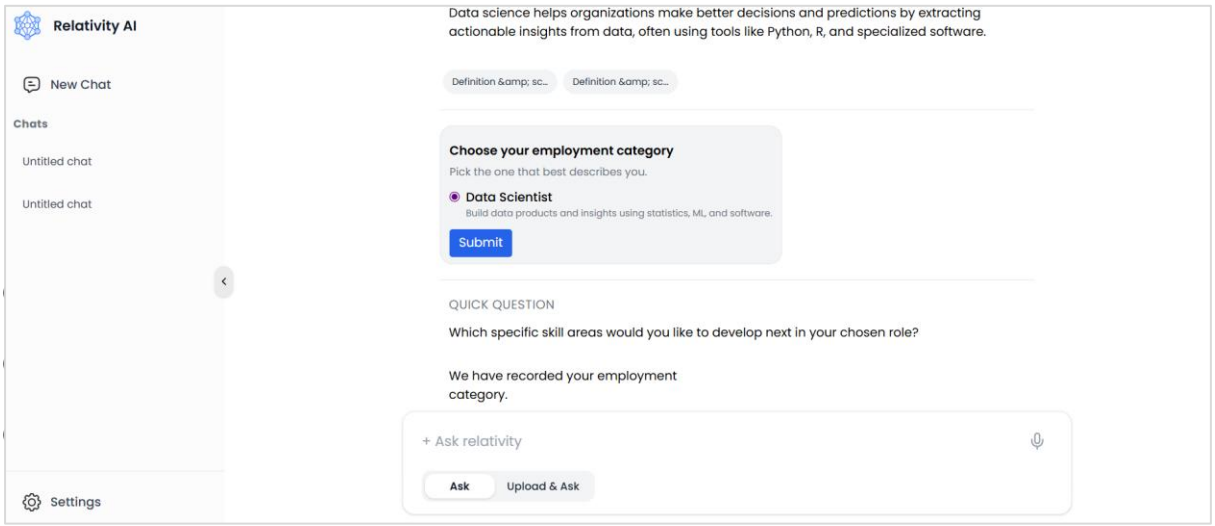
Component 06 : UIA segment extraction

what should it do?

C06 (Intake) is the funnel starter: its goal is to decide what the user should fill in next for **Employment Category** and **Skills**. It reads the new message and the chat's UIA state to detect intent, then either emits an **EC survey** (if EC isn't set), a **Skills survey** tailored to the chosen EC (if skills aren't set), or does nothing if both are already captured. It uses canonical option lists (so answers are machine-detectable), avoids re-asking what's already saved, and returns a clear action (e.g., show_ec_survey, show_skills_survey, recorded_ec, or none) to drive the rest of the pipeline.

Results : employment category

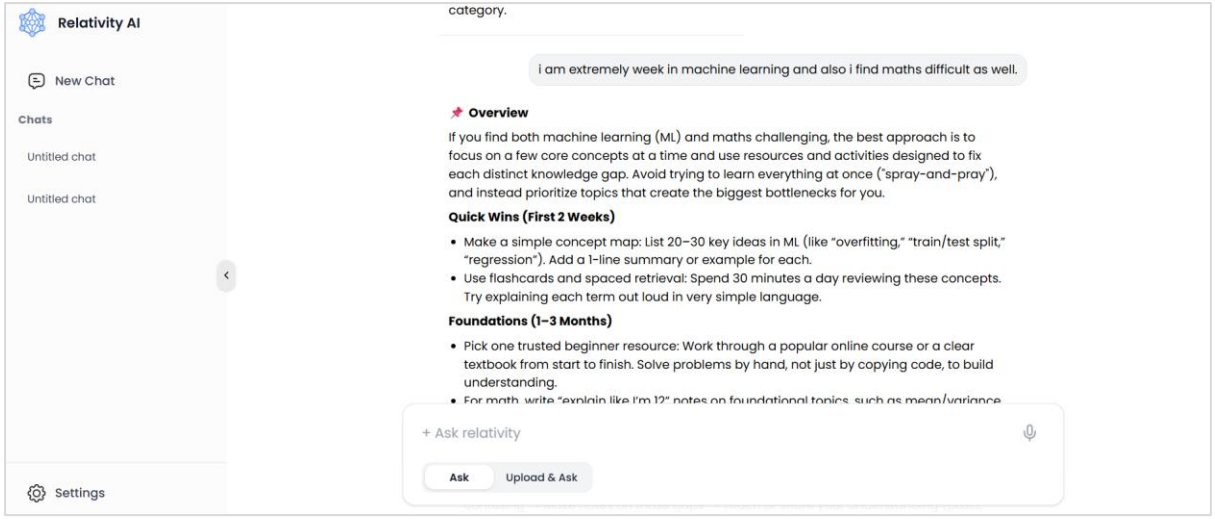


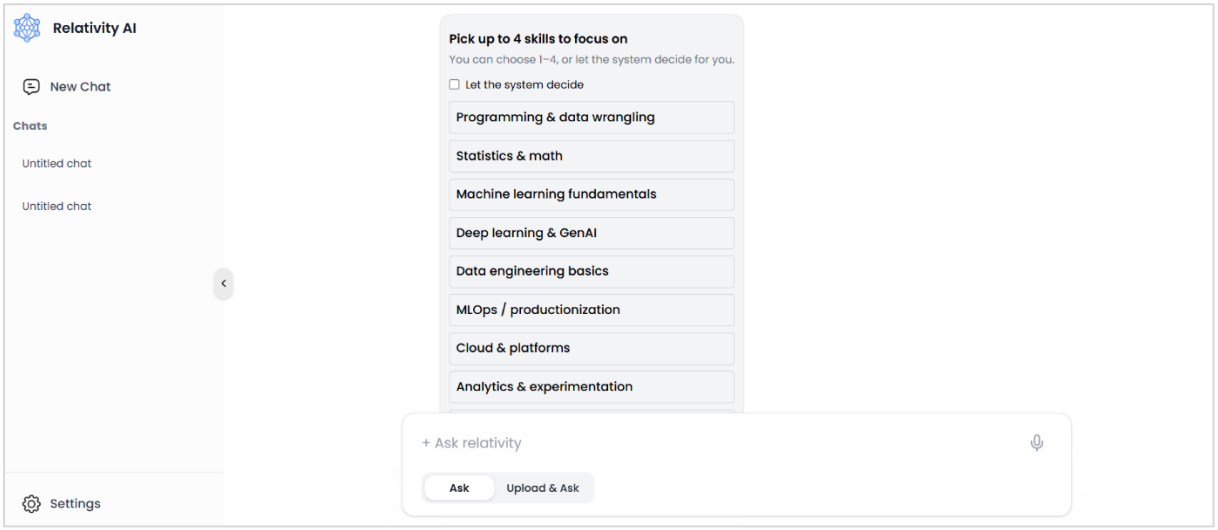


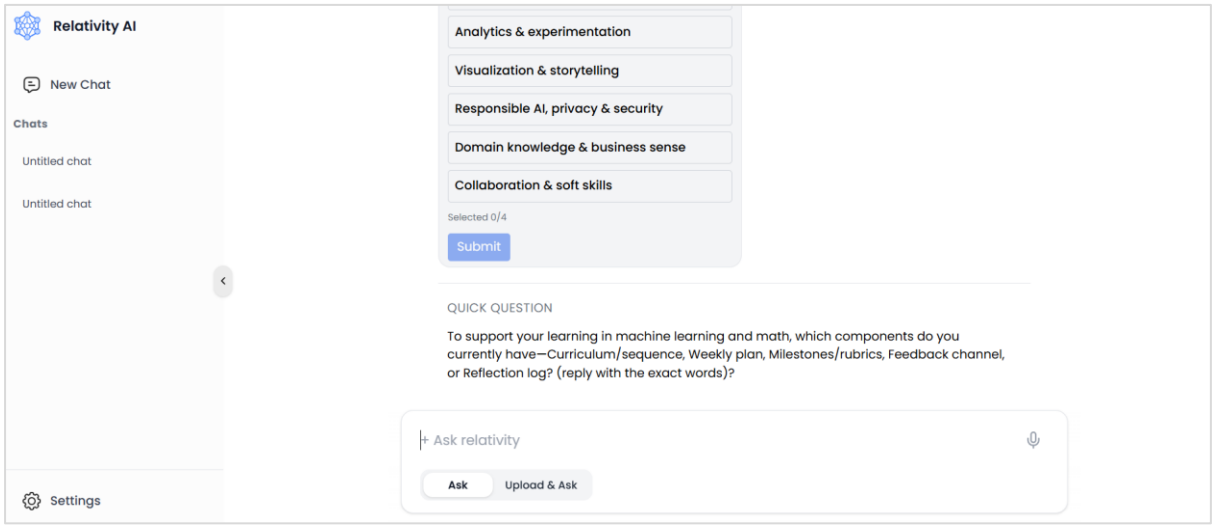
Success metrics

Based on the prompt the user has given we have given a survey for the user to answer to select if their preferred employment category is data science or not. This was done using the auto inference method. By this method we collect the employment category that we need.

Results : skills category for the selected employment category







Success metrics

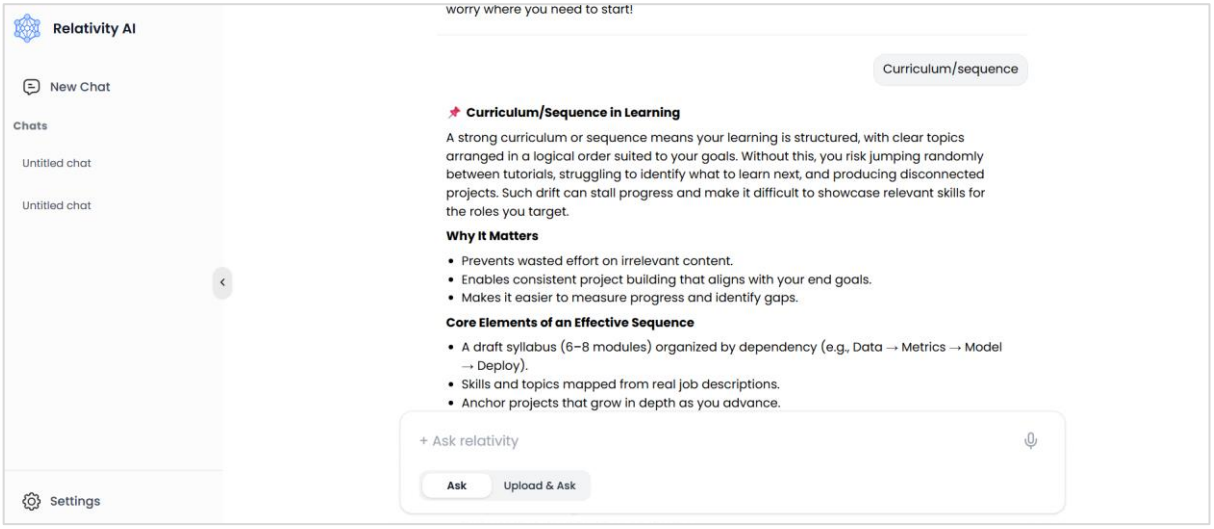
Now that we must collect the skills category user need to we need to emit the skills if the prompt has any inferences made on skills. So a survey would be triggered at this situation for the user to select the maximum of 4 skills or let the system decide. Since user has referred to

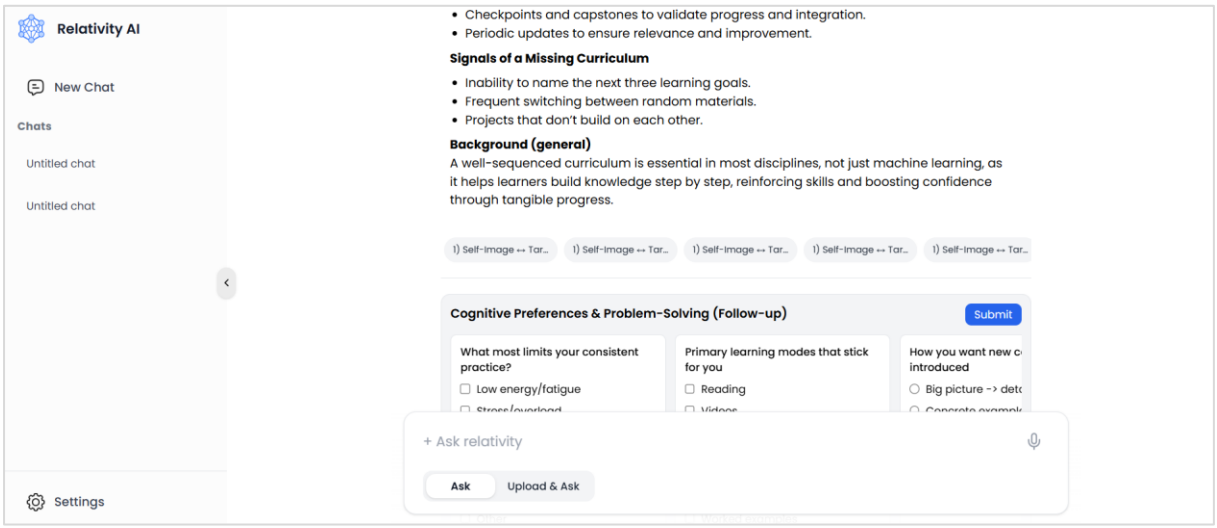
machine leaning and maths skills weakness we trigger the user to select the skills they need to proceed with.

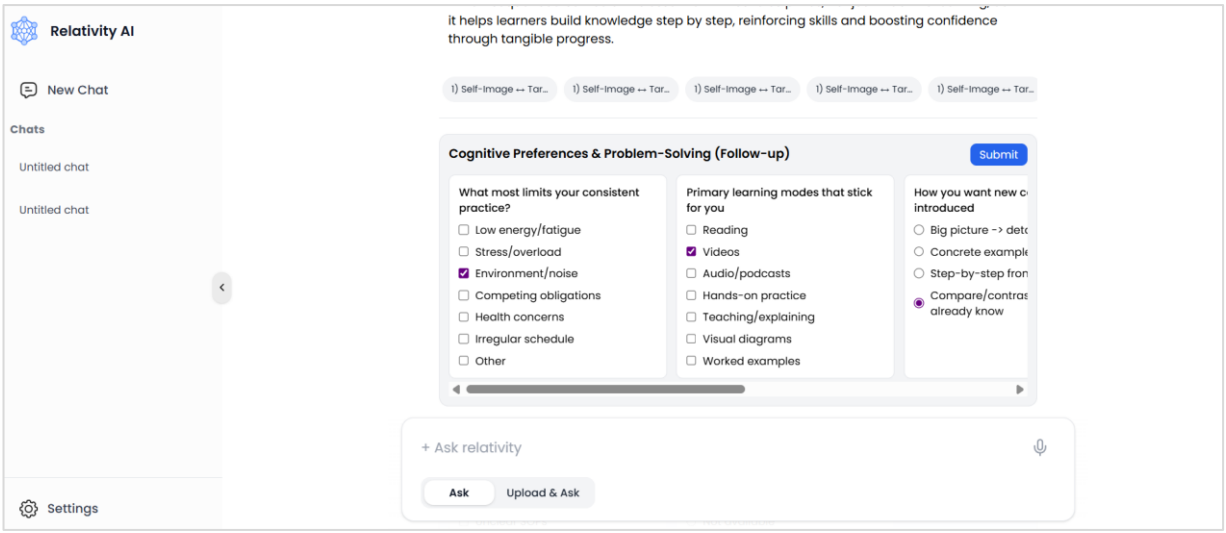
Component 07 : UIA insight extraction

what should it do?

C07 (Insight Engine) is the pain-points collector: its goal is to capture the user's bottlenecks and preferences as structured **insights**. Each turn it auto-infers any answers from the user's text and tracks progress by ordered batches; when both Employment Category and Skills are set, it builds a focused **Insight survey** with canonical options for the next unanswered items (Batch 1 → 2 → …). If prerequisites aren't met or nothing is pending, it skips survey creation and simply records touched/completed status so later turns can pick up cleanly. Following results demonstrate how both stages in insight are used as UIA stage 01 and UIA stage 02.







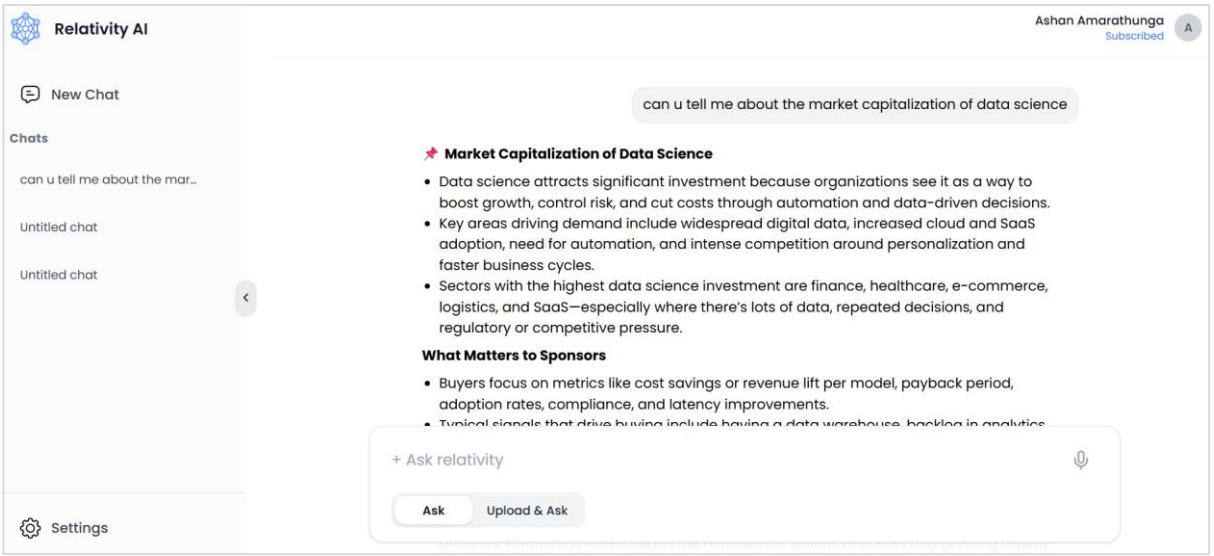
Success metrics

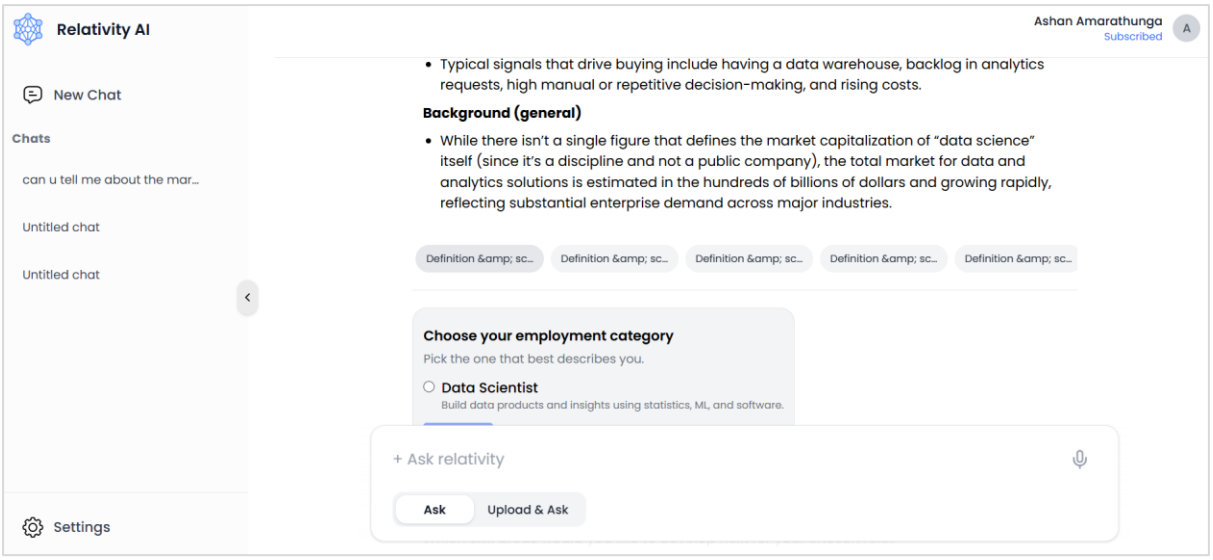
The users prompts indicates a response to a an insight which need to be collected. Stage 01 uses auto inference to get this insight and along with this insight the batch that contains the insight is triggered as a survey.

Component 08 : UAA RAG knowledge base

what should it do?

C08 (RAG) is the knowledge answerer: its goal is to generate a grounded, readable Markdown reply to the user's question using your curated corpus. It plans focused sub-queries, retrieves evidence via hybrid search (vectors + BM25), reranks and filters with the LLM, then composes and validates an answer (TL;DR, key points, details). Alongside the content, it returns source metadata so the UI can show a horizontal strip of "source" chips, keeping the response transparent and verifiable.





Success metrics

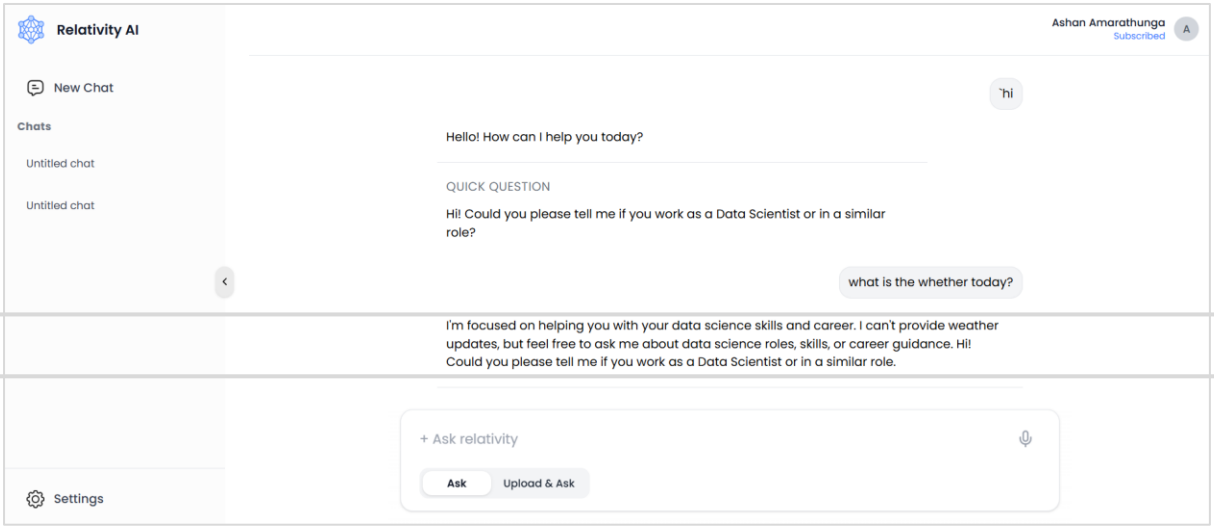
agentic ai retrieve the knowledge needed to answer the response from vector DB which is stored in FAISS. The rectangular ash boxes show the from which sections the sources were taken in the pdf. The sources from PDF are as follows. The following result in cmd is related to component 08 and it shows how chunks has been received as information precises from necessary documents in order to complete user's request.

Component 10 : UAA Encouragement strategy

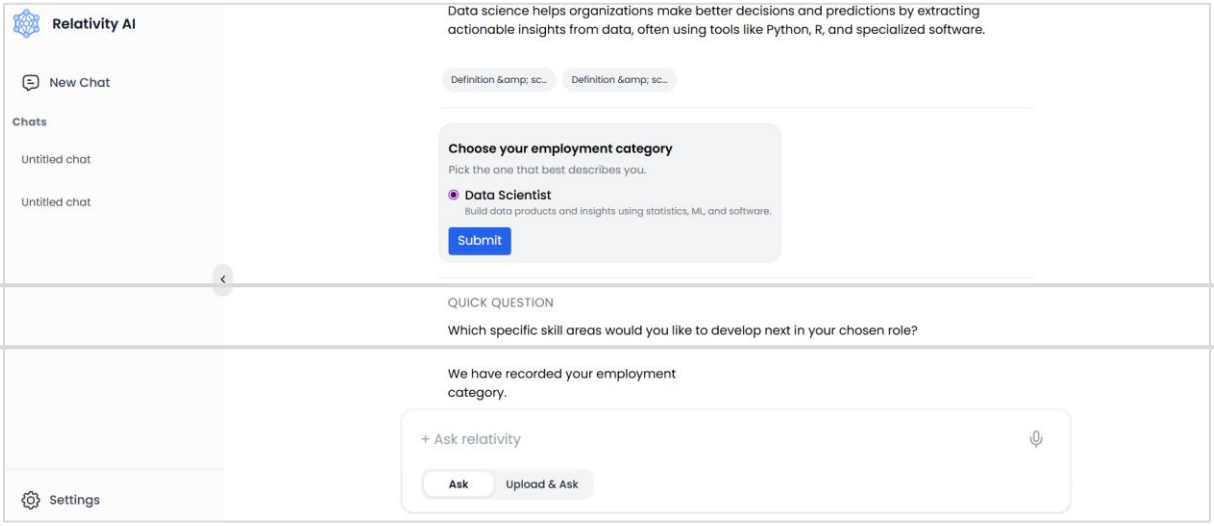
what should it do?

C10 (Encouragement) is the momentum keeper: its goal is to nudge the user toward the **next missing step** in the analysis funnel—**Employment Category → Skills → Insights**—whenever a survey isn't already on screen. It looks at chat state and recent actions, then asks one short, creative, **canonically worded** question (so answers are auto-detectable): first to choose an EC if unknown, otherwise to pick priority skill categories for that EC, and finally to surface a specific insight from the next eligible batch.

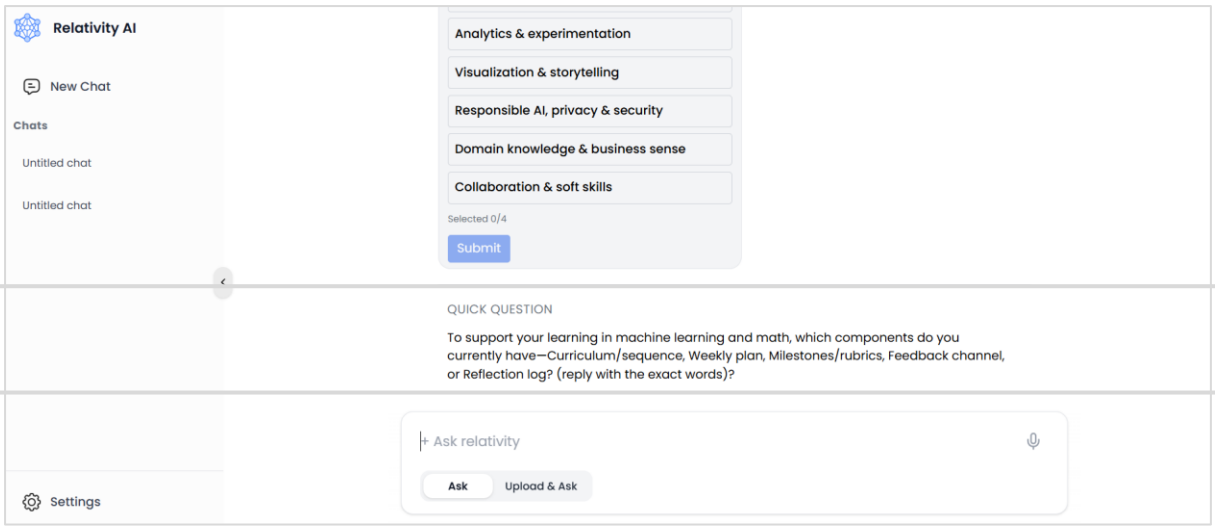
Employment Category : encouragement strategy 01



Skills : encouragement strategy 02



Insights : encouragement strategy 03



Conclusion

This project demonstrates how a focused, agentic AI approach can turn vague career pain points into structured, evidence-backed guidance. We framed the solution as a multi-step problem solver and deliberately scoped the MVP to two collaborating agents—User Analysis (UAA) and User Identification (UIA)—so the system first understands the user before attempting any plan or execution, aligning effort with the university's requirement and our team-of-four capacity.

Technically and methodologically, we split the runtime into an Agentic AI pipeline (from prompt to response) and a Retrieval-Augmented Generation (RAG) pipeline. This separation made the conversation flow controllable (via Decision Gate and staged insight collection) while keeping answers grounded in curated knowledge. The RAG corpus was engineered for traceability and freshness, with hybrid FAISS vectors and BM25 lexical search fused for robust retrieval—an intentional design to reduce hallucinations, maintain currency, and provide transparent provenance.

Ethical and responsible AI practices were embedded throughout: fairness via task-scope gating and neutral, user-led data collection; transparency through clear boundaries, UI separation of content/survey/nudge bubbles, and visible source chips; and privacy by relying on a local retrieval index rather than uncontrolled web calls.

On the product side, we delivered a coherent stack—Python, React, Tailwind, MongoDB, Docling-based RAG KB—and a Figma-driven UI that clarifies the agent's intent each turn. Looking ahead, the roadmap naturally extends to adding the Planning and Execution/Verification agents, broadening the domain corpus, and stress-testing the pipelines with more users. Finally, our commercialization focus on the data-science market—starting with junior roles—keeps the impact tangible and measurable.

In sum, the team delivered a principled MVP that pairs disciplined conversation control with auditable knowledge retrieval—building user trust today and laying a practical foundation for richer agent capabilities tomorrow.

Roles and responsibilities

#	Name	ID	Responsibilities
1	A V Amarathunga	IT23192782	**Project leader.** 1. Strategic planning and foreseeing project development from start to end. 2. Supported every single stage of the development pipeline with the necessary guidance by solving problems, building system architectures needed and advising the team. 3. Developed + Enhanced Backend development for components agentic pipeline + RAG pipeline.
2	M S R V Amararathna	IT23264366	**Backend lead** 1. Main backend developer of the components in the agentic ai pipeline. 2. lead the main integration of frontend + backend. 3. Lead the systems backend development by solving conflicts, problems and potential risks.
3	S A R U Amarasinghe	IT23216778	**Frontend + Backend developer** 1. Developed the frontend for the agentic ai system. 2. Carried out research to collect the data and information needed to build the RAG knowledge base.. 3. Carried out the RAG kb development from facts in PDF to vector DB in FAISS. 4. Carried out the Video + documentation work.
4	A A Gunawardena	IT23140752	**Frontend + Backend developer** 1. Developed the frontend for the agentic ai system. 2. Carried out research to collect the data and information needed to build the RAG knowledge base. 3. Carried out the RAG kb development from facts in PDF to vector DB in FAISS. 4. Carried out the Video + documentation work.