Web IR

Tsinghua

Job Finder

Design & Implementation

A web-first WebIR system that turns natural-language internship searches into verified French company leads.

Theodore Billotte - 2025400572 | Louis Rollet - 2025400573

Web Information Retrieval Workshop - May 19, 2026

Presentation roadmap

1. Frame the problem
Students need structured, trustworthy company leads, not another list of pages to inspect manually.

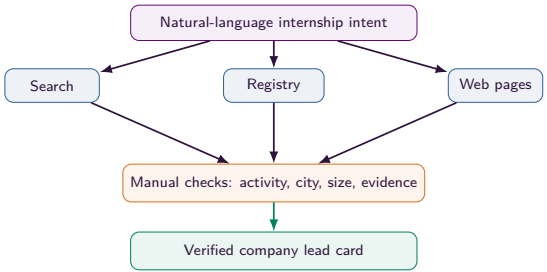
2. Explain retrieval
Natural-language intent is converted into NAF, location, size, evidence, and relevance decisions.

3. Show implementation
A React/FastAPI application makes the long-running IR pipeline reviewable and resumable.

Problem: company discovery is fragmented

User need
A student asks for: **"video game studios in Montpellier with more than 50 employees".**

- sector labels are broad and inconsistent;
- registry records need location, status, and size checks;
- websites and social pages contain useful evidence, but not in a table.

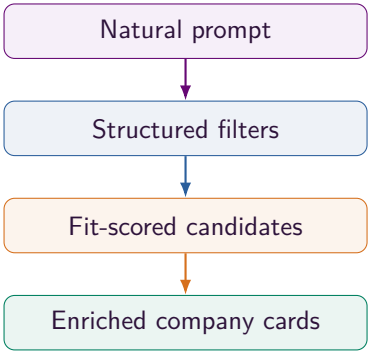


Point
The task is retrieval + normalization + relevance validation, not one search box.

What the website serves

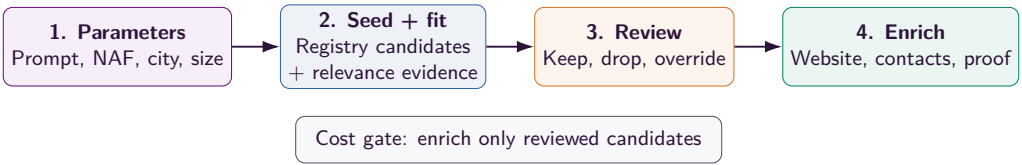
Job Finder is a resumable research console.

- the user writes a plain-English search prompt;
- the system proposes structured filters: NAF codes, city, size, thresholds;
- companies are retrieved from official French registry data;
- AI and web evidence filter out false positives;
- kept companies become enriched cards: website, emails, phone, socials, evidence.



The implemented product is a FastAPI + React web app with a four-step wizard, MongoDB persistence, and live progress over Server-Sent Events.

System workflow: four user-facing steps



Design principle
Retrieve broadly first, filter aggressively, then spend web-search and LLM budget only on companies that are likely useful.

Why this is a Web Information Retrieval problem

The query is semantic, but the official database is structured.

- users describe sectors in natural language;
- French companies are indexed with NAF activity codes;
- NAF codes are vague, overlapping, and sometimes too broad;
- the system must bridge intent, taxonomy, location, and web evidence.

Example difficulty
"Video game company" can mean game publishing, software development, multimedia production, animation, or post-production depending on the company.

IR strategy
High-recall seed retrieval, then semantic relevance checking and enrichment for precision.

Data sources and retrieval signals

Source / signal	Role in the system	Stage
French company registry	Official company and establishment data: address, identifiers, activity code, status	Seed retrieval
Curated NAF taxonomy	Maps business domains to possible activity classes	Query translation
NAF crosswalk	Converts selected NAF 2025 codes to older rev.2 codes required by the API	API filtering
Web search	Finds official sites and external evidence for relevance	Fit + enrichment
Company pages	Extracted text, contacts, social links, and evidence snippets	Enrichment

The project combines structured public records with web evidence instead of relying on only one source.

Retrieval challenge 1: NAF codes are necessary but ambiguous

NAF codes describe company economic activity. They are essential for registry search, but they are not a perfect semantic taxonomy.

- one activity can require several NAF codes;
- one code can contain several real-world domains;
- the project keeps a curated NAF 2025 taxonomy but the official endpoint still filters with older NAF rev.2 values.

Domain code	Meaning	API filter
58.21Y	Game publishing	58.21Z
62.10Y	Programming	62.01Z
59.12Y	Video post-production	59.12Z

Implementation point

The NAF subset and crosswalk make retrieval reproducible and compatible with the official registry API.

Retrieval challenge 2: prompt understanding must be structured

Input prompt
"Find video game studios near Montpellier, preferably small enough for an internship."

Expected by the backend:

- relevant NAF codes;
- city and postal-code candidates;
- employee range;
- fit threshold;
- optional exclusions.

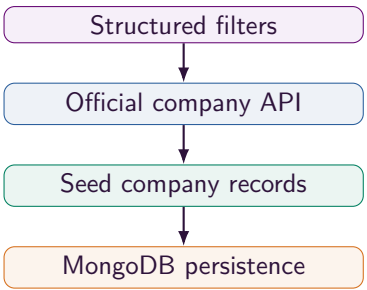
```
{
  "domain": "video game studios",
  "city": "Montpellier",
  "postalCodes": ["34000", "34070"],
  "nafCodes": ["58.21Z", "62.01Z"],
  "maxEmployees": 50,
  "minFitScore": 0.65
}
```

Design decision
The LLM is not the data source. It transforms vague intent into controllable filters and auditable decisions.

Seed phase: official registry retrieval

Data source. The seed phase queries the French official company-search API and stores seed records in MongoDB.

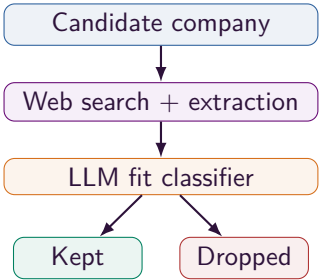
- filters: NAF, city/postal code, employee range, active status;
- establishment-aware: local establishments matter, not only headquarters;
- stored fields: headquarters SIRET, local establishment SIRET, address, commune, legal status, size, raw evidence.



Fitting phase: remove false positives with web evidence

Why fitting is required. Registry retrieval is high recall, but not high precision. A company can match an IT-related code while not being a video-game studio.

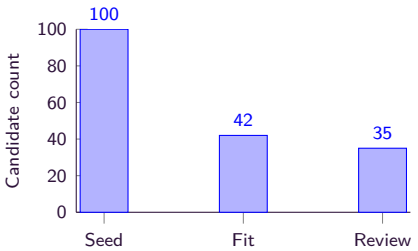
- search web snippets and pages about each company;
- give company context + target domain to a relevance checker;
- produce a confidence score and kept/dropped decision;
- keep evidence so users can understand the decision.



Human review before enrichment

Why review matters
Enrichment costs time and quota, model relevance is not perfect, and users may know the exact internship target better than the classifier.

- Pending, Kept, and Dropped columns make decisions visible;
- users can override borderline companies;
- overrides are stored in MongoDB;
- the final enrichment set is auditable.



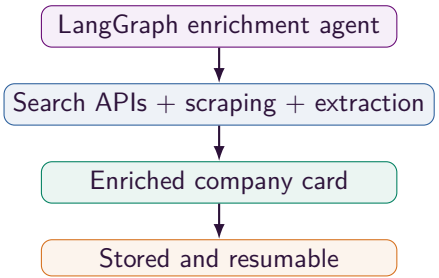
Illustrative: broad retrieval over-generates; fit and review improve precision.

Enrichment phase: turn matches into actionable records

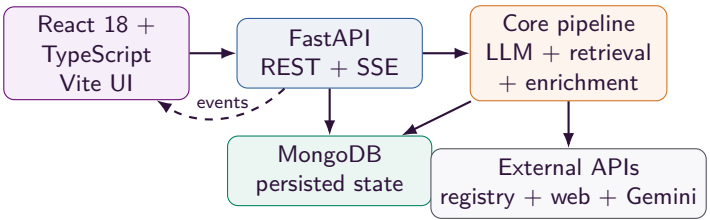
What enrichment adds

- official website and source confidence;
- generic or personal email signals, classified by type;
- phone numbers and social links;
- short evidence snippets supporting the domain match;
- traceability for later review or removal.

Key principle
The final result is not only a name list. It is a lead card with provenance: what we found, where we found it, and why it matches.



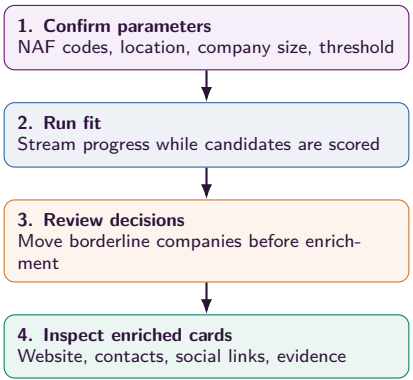
Implementation architecture



Implementation focus

The frontend stays responsive while the backend runs long retrieval, fitting, and enrichment jobs. Progress and final results are persisted so a search can be resumed later.

Frontend: wizard turns pipeline state into decisions



Why this UI shape fits IR

Each page corresponds to one irreversible or expensive decision: choose filters, launch retrieval, approve candidates, then enrich only the useful set.

State model

React Router exposes resumable search URLs. MongoDB stores stage outputs. Server-Sent Events keep long-running backend jobs visible.

Main API surface used by the website

Endpoint family	Purpose
`/api/searches`	create, list, fetch, update, pause, resume, or delete searches
`/api/searches/{id}/seed`	start official company seed retrieval
`/api/searches/{id}/fit`	start candidate relevance fitting
`/api/searches/{id}/decisions`	inspect and override keep/drop decisions
`/api/searches/{id}/enrich`	start full web enrichment for kept companies
`/api/searches/{id}/events`	stream live progress events to the frontend
`/api/reference/*`	lookup NAF codes, postal codes, and geocoding helpers

Backend design: stateful long-running jobs

Pipeline execution

- each phase can be launched separately;
- background runners avoid blocking API requests;
- progress state is streamed and persisted;
- pause/resume is possible because intermediate outputs are stored.

Why persistence is important

- web retrieval can be slow or quota-limited;
- a demo can fall back to a previous completed search;
- users can compare decisions and evidence later;
- debugging is easier because each stage has stored artifacts.

Practical takeaway
The system is designed as a web application around an IR pipeline, not as a one-shot API call.

Difficult implementation points

Ambiguous classification
One sector can map to many NAF codes; one NAF code can contain many unrelated activities.

LLM output reliability
The backend needs strict JSON outputs for filters and repeatable confidence decisions.

Noisy candidates
Closed companies, micro-enterprises, multiple addresses, and broad IT firms create false positives.

Long-running UX
Registry search, web search, scraping, and enrichment need progress streaming, cancellation, and resumability.

The design is therefore retrieval plus validation, evidence, operational control, and user review.

Technical safeguards and project boundaries

Precision and reliability controls

- curated NAF taxonomy and NAF crosswalk;
- Pydantic schemas for structured state;
- relevance score before enrichment;
- manual review for kept/dropped decisions;
- bounded concurrency, retries, and rate limits.

Ethics and privacy boundaries

- public company-registry data first;
- web pages fetched only when useful for fit/enrichment;
- robots.txt and quota-aware crawling;
- personal emails are flagged separately;
- evidence is stored for traceability and removal requests.

Expected results and limitations

What works well

- natural-language entry point;
- official-data-first retrieval;
- domain-fit filtering improves precision;
- evidence-backed enrichment makes results usable;
- resumable searches support long-running IR tasks.

Current limitations

- LLM JSON output requires validation and fallbacks;
- NAF categories remain imperfect even after curation;
- web evidence can be missing or outdated;
- contacts may be generic rather than internship-specific;
- external APIs introduce latency and quota limits.

Demo

From prompt to enriched company leads

Q&A

Questions