## Sear1. Պարզ Google search bar կայքում

**Նպատակ** – կայքումդ դնում ես search field, ու ամեն ինչ ուղղակի տանում է Google.com–ի որոնման engine-ին։  
**Տրամաբանություն**

1. Կառուցում ես սովորական HTML ձև (form)։
2. Երբ օգտատերը գրում է keyword ու սեղմում search, քո form-ը ուղղորդում է Google–ի https://www.google.com/search?q=...։
3. Ոչ մի backend կամ API պետք չի։

👉 Օրինակ՝

<form method="get" action="https://www.google.com/search" target="\_blank">

<input type="text" name="q" placeholder="Search on Google...">

<button type="submit">Search</button>

</form>

## 🔎 2. Օգտագործել **Google Custom Search API**

**Նպատակ** – ծրագրով (JS, Python, Node.js և այլն) հարցում ուղարկել Google-ին և ստանալ արդյունքները JSON ձևաչափով։  
**Տրամաբանություն**

1. Գնում ես Google Cloud Console։
2. Ստեղծում ես **Custom Search Engine (CSE)**։ Կարող ես նշել՝ ամբողջ ինտերնետում, թե կոնկրետ կայքերում որոնելու։
3. Ստանում ես **API Key** և **Search Engine ID**։
4. Backend-ում կամ frontend-ում կանչում ես API-ն։

👉 Օրինակ Node.js (Express) backend-ում.

import express from "express";

import fetch from "node-fetch";

const app = express();

const API\_KEY = "YOUR\_API\_KEY";

const CX = "YOUR\_SEARCH\_ENGINE\_ID";

app.get("/search", async (req, res) => {

const query = req.query.q;

const url = `https://www.googleapis.com/customsearch/v1?key=${API\_KEY}&cx=${CX}&q=${query}`;

const response = await fetch(url);

const data = await response.json();

res.json(data);

});

app.listen(3000, () => console.log("Server running on port 3000"));

Այստեղ /search?q=apple կանչելիս կստանաս Google-ի որոնման արդյունքները JSON-ով։

## 🔎 3. Կառուցել քո սեփական Search Engine (Google-ի նման)

**Նպատակ** – լիարժեք սեփական search engine (crawler + index + ranking)։ Սա արդեն ավելի բարդ է։  
**Տրամաբանություն**

1. **Crawler (scraper)** – ծրագիր, որը շրջում է կայքերով ու հավաքում տվյալներ։
2. **Indexer** – պահում է բառերը ու դրանց կապը փաստաթղթերի հետ (օր. inverted index data structure)։
3. **Ranking / Scoring** – որոնման ժամանակ հաշվում ես՝ որ փաստաթղթերը ավելի համապատասխան են (TF-IDF, BM25 կամ ML մոդելներով)։
4. **Frontend UI** – search bar + արդյունքների ցուցադրում։

Սա արդեն Google-ի նման համակարգի հիմքն է, բայց փոքր մասշտաբով։

## 🔑 Ընդհանուր տրամաբանություն

* Եթե քեզ պետք է **պարզ որոնում**, բավական է տարբերակ 1։
* Եթե ուզում ես **քո համակարգից Google-ից արդյունքներ ստանալ**, ընտրիր տարբերակ 2։
* Եթե ուզում ես **ձեր բանկում/ընկերությունում ներքին search engine կառուցել**, պետք է գնաս տարբերակ 3–ի ճանապարհով։

📌 Քեզ համար ճշտեմ՝ դու ուզում ես **քեզ մոտ (ներքին համակարգում) պատրաստել search համակարգ**, թե՞ ուղղակի օգտագործել Google-ի հնարավորությունները (search bar կամ API)?

##### You said:

3րդ տարբերակն ավելի բացիր

###### ChatGPT said:

Հասկացա 👍 Դու ուզում ես հասկանալ **Google-ի նման search engine կառուցելու տրամաբանությունը**։ Ես հիմա կբացեմ քայլ առ քայլ՝ ինչ բաղադրիչներ են պետք, ինչպիսին է դրանց կապը ու ինչ տեխնոլոգիաներ կարելի է օգտագործել։

# 🚀 Քո սեփական Search Engine կառուցելու քայլեր

## 1. **Crawler (վեբ էջերի հավաքում)**

Crawler-ը (կամ spider-ը) այն ծրագիրն է, որը շրջում է ինտերնետով կամ քո նշած կայքերով և բերում էջերի պարունակությունը։

* Սկսում ես seed URLs-ից (օրինակ՝ www.example.com)։
* Բացում ես էջը, քաղում դրա տեքստը + հղումները։
* Հղումները ավելացնում ես հերթում (queue) ու շարունակում։
* Պահում ես raw HTML-ը կամ միայն տեքստային մասերը։

🛠️ Տեխնոլոգիաներ –

* Python → scrapy, beautifulsoup4
* Node.js → puppeteer, cheerio

## 2. **Parsing & Normalization (տեքստի մաքրում)**

Crawler-ի բերած տվյալները շատ “աղմկոտ” են։ Պետք է մաքրել՝

* Հեռացնել HTML tag-երը, JS-ը, CSS-ը։
* Նորմալացնել բառերը (lowercase, stop words հեռացնել, stemming/lemmatization)։
* Տեքստը բաժանել բառերի/token-ների։

Օրինակ՝ «Հարևանների բակում խաղում էին երեխաները» →  
[հարևան, բակ, խաղալ, երեխա]

## 3. **Indexing (ինդեքսավորում)**

Search engine-ի սիրտը։ Օգտագործում ես **inverted index**։

🔎 Օրինակ

բանկ → [doc1, doc4, doc7]

վարկ → [doc2, doc4]

քարտ → [doc3, doc5]

Այսինքն պահում ես՝ որ բառը որ փաստաթղթում կա։  
Index-ը սովորաբար պահում են հատուկ բազայում՝

* **Elasticsearch**, **Solr**, կամ սեփական տվյալների կառուցվածքով։

## 4. **Ranking (որոնման արդյունքների դասավորում)**

Երբ օգտատերը որոնում է, պետք է ոչ միայն գտնել փաստաթղթերը, որտեղ keyword կա, այլև դասավորել ըստ համապատասխանության։

Հիմնական ալգորիթմներ՝

* **TF-IDF (Term Frequency – Inverse Document Frequency)** – հաշվում է բառի կարևորությունը փաստաթղթում ու ամբողջ կորպուսում։
* **BM25** – ավելի արդիական տարբերակ։
* Machine Learning-based Ranking (ML models, embeddings, semantic search)։

Օրինակ՝

"բանկային քարտ" որոնելիս՝

- doc3 (համընկնում երկու բառով, վերնագրում) → բարձր միավոր

- doc5 (համընկնում մեկ բառով, տեքստի մեջ) → ցածր միավոր

## 5. **Query Processing (հարցման մշակումը)**

* Օգտատերը գրում է հարցում։
* Tokenize → normalize → synonym expansion։
* Որոնում ես index-ում։
* Վերցնում ես համապատասխան փաստաթղթերը։

## 6. **Frontend / API**

* Պետք է ունենաս search box։
* API backend-ը հարցնում է index-ին։
* Վերադարձնում է արդյունքները rank-ով դասավորված։
* Ցուցադրում է (վերնագիր, նկարագրություն, հղում)։

# ⚙️ Օրինակական stack

* **Crawler** – Python Scrapy
* **Storage** – MongoDB/Postgres raw text-ի համար
* **Indexing & Search** – Elasticsearch կամ Whoosh (Python library)
* **Ranking** – Elasticsearch built-in BM25 կամ custom ML
* **Frontend** – React + Node.js/Express API

# 📊 Աշխատանքի տրամաբանություն

1. Crawl → հավաքում ես բովանդակություն։
2. Parse → մաքրում ես, տեքստը բաշխում բառերի։
3. Index → պահում ես բառ → փաստաթուղթ կապերը։
4. Query → օգտատերը գրում է որոնում։
5. Match → որոնում ես index-ում։
6. Rank → դասավորում ես ըստ կարևորության։
7. Display → frontend-ին տալիս ես վերջնական արդյունքներ։

Քեզ օրինակ պետք է ցույց տամ **mini search engine** Python-ով (crawler + index + query)՝ մի քանի էջի վրա աշխատող, թե՞ քեզ պետք է ավելի **արտադրական մակարդակի stack** նկարագրություն (Elasticsearch-ով)?

##### You said:

ինչ ռեսուրսներ է ինձ պետք որպես ընկերություն

###### ChatGPT said:

Շատ լավ հարց ես տալիս 👌, որովհետև եթե ուզում ես որպես **ընկերություն** կառուցել Google-ի նման search engine (թեկուզ փոքր մասշտաբով, օրինակ՝ ներքին corporate search կամ մեծածավալ ինդեքս), պետք է հասկանալ ոչ միայն ծրագրավորման քայլերը, այլ նաև **ռեսուրսները՝ մարդիկ, տեխնոլոգիաներ, ենթակառուցվածք**։

Ես բաժանում եմ ըստ ուղղությունների․

# 🏗️ Ռեսուրսներ Search Engine կառուցելու համար

## 1. 👨‍💻 Մարդկային ռեսուրսներ

Պետք է ունենաս խաչաձև թիմ (cross-functional):

* **Software Engineers (Backend + Data)** – crawler, indexer, API զարգացնելու համար։
* **Data Engineers** – տվյալների հավաքագրում, մաքրում, նորմալացում։
* **Machine Learning Engineers / Data Scientists** – ranking մոդելներ, semantic search, NLP։
* **DevOps / System Administrators** – սերվերների, բազաների, storage-ի կառավարում։
* **Frontend Developers (React, Vue, Angular)** – որոնման UI կառուցելու համար։
* **QA Engineers** – ինդեքսի ճշտության, արագության ու կայունության թեստավորում։
* **Product Manager / Business Analyst** – պահանջների հավաքում, roadmap-ի կառավարում։

Եթե սա միայն ներքին search engine է (օրինակ՝ ընկերության փաստաթղթերի համար), բավական է ~5–7 հոգանոց թիմ։  
Եթե ուզում ես կոմերցիոն search engine (օրինակ՝ Yandex-ի կամ Bing-ի նման), պետք է տասնյակ–հարյուրավոր մասնագետներ։

## 2. 🖥️ Տեխնոլոգիական ենթակառուցվածք

* **Crawler & Scraper Systems** – սերվերներ, որոնք կանոնավոր շրջում են ինտերնետով։
* **Storage** –
  + Raw data (HTML, PDF, Docs) պահելու համար՝ Hadoop/HDFS, AWS S3 կամ on-premise storage։
  + Metadata և index-ի համար՝ ElasticSearch, Solr, կամ self-built inverted index։
* **Databases** – PostgreSQL/MongoDB auxiliary տվյալների համար։
* **Search Cluster** – Elasticsearch կամ OpenSearch (distributed, scalable)։
* **Processing Frameworks** – Spark/Hadoop (մեծածավալ տվյալների մշակման համար)։
* **Caching Layer** – Redis/Memcached (արագացնելու համար)։

## 3. ⚡ Հաշվողական ռեսուրսներ

Կախված է՝ փոքր թե մեծ search ես ուզում։

* **Փոքր search (ներքին)** –
  + 2–3 սերվեր (16–32GB RAM, 8 CPU cores, 1–2TB SSD storage)։
  + Կարող ես տեղակայել on-premise կամ cloud (AWS/GCP/Azure)։
* **Միջին search (տասնյակ միլիոն փաստաթուղթ)** –
  + Distributed ElasticSearch/OpenSearch cluster։
  + 5–10 սերվեր (64GB RAM, NVMe SSD)։
  + GPU ռեսուրսներ semantic search-ի և embeddings-ի համար։
* **Մեծ search (Google/Yandex մակարդակ)** –
  + Հարյուրավոր–հազարավոր սերվերներ։
  + Data centers, իր սեփական CDN, միլիարդավոր փաստաթղթերի crawling։

## 4. 📚 Գիտելիք և տեխնոլոգիական stack

* **NLP & IR (Information Retrieval)** գիտելիքներ։
* **Indexing & Ranking Algorithms** – TF-IDF, BM25, PageRank, embeddings։
* **Distributed Systems** – Elasticsearch, Solr, Cassandra։
* **Cloud Technologies** – AWS/GCP/Azure, Kubernetes, Docker։
* **Security & Compliance** – GDPR, տվյալների գաղտնիություն, access control։

## 5. 💰 Բյուջե

* Փոքր ներքին նախագծի համար (օր. ընկերության փաստաթղթերի որոնում) – կարելի է սկսել $50k–$100k ներդրումով։
* Միջին search (հանրային կամ մասնագիտացված engine) – $500k–$1M։
* Մեծ search (Google/Yandex մակարդակի) – հարյուր միլիոնավոր դոլարներ։

# 📊 Աշխատանքային տրամաբանություն ընկերության մակարդակով

1. Սկզբում սահմանել → **ինչի համար է պետք** search engine-ը։ (ներքին փաստաթղթեր, հաճախորդների տվյալներ, թե համացանց)։
2. Կազմել **MVP** (prototype) փոքր թիմով՝ 1–2 crawl, indexing, simple UI։
3. Ընտրել **stack** (Elasticsearch, OpenSearch, կամ ինքնուրույն համակարգ)։
4. Ապահովել **infrastructure** (servers/cloud)։
5. Հետո արդեն **scale up** անել՝ ավելացնել ML ranking, semantic search, recommendation։

Կուզե՞ս ես քեզ համար կազմեմ **step-by-step roadmap**, օրինակ՝ «ինչ քայլերով պետք է գնա ընկերությունդ search engine կառուցելու համար՝ սկսած MVP-ից մինչև production scale»։

##### You said:

ստեղխիր flow diaramm

###### ChatGPT said:

Ահա flow diagram-ը, որը ցույց է տալիս search engine-ի աշխատանքի տրամաբանությունը․

📌 Ներբեռնել flow diagram-ը

Ուզո՞ւմ ես, ես նաև տեքստային (ASCII կամ Mermaid.js) տարբերակ ստեղծեմ, որ հեշտ ինտեգրվի docs/wiki-ում։

##### You said:

նկար տուր

###### ChatGPT said:

Ահա flow diagram-ի նկարը․

##### You said:

ուր է

###### ChatGPT said:

Հասկացա 🙂 Նկարը ինքս պահել եմ, բայց քեզ պետք է այստեղ անմիջապես երևա։ Կրկին կգծեմ ու կցեմ պատկերի ձևաչափով։

Ահա քո որոնողական համակարգի **flow diagram**–ի նկարը 📊․

Այս սխեման ցույց է տալիս քայլերը՝ Crawl → Parse → Index → Query → Ranking → Results։

##### You said:

պատրաստիր նախագծի տեխնիկական դոկումենտացիան

###### ChatGPT said:

Հիանալի 🙂  
Ես քեզ համար կպատրաստեմ նախագծի **տեխնիկական դոկումենտացիայի skeleton**, որը սովորաբար օգտագործվում է search engine-ի նման համակարգերի համար։ Դու կարող ես հետո լցնել կոնկրետ տվյալներով (stack, պահանջներ, թիմի կազմ)։

# 📑 Տեխնիկական Դոկումենտացիա – Search Engine Project

## 1. Ներածություն

* **Նպատակ**: Կառուցել որոնողական համակարգ, որը հնարավորություն կտա օգտատերերին որոնել փաստաթղթեր/կայքեր և ստանալ դասավորված արդյունքներ։
* **Տարածք (Scope)**:
  + Crawl արտաքին/ներքին կայքերից։
  + Parse և մաքրել տվյալները։
  + Ստեղծել inverted index։
  + Իրականացնել հարցումների մշակումը և արդյունքների դասավորումը։
  + Ապահովել frontend UI և API։
* **Թիրախ օգտատերեր**:
  + Ներքին աշխատակիցներ (corporate search)։
  + Կամ արտաքին հաճախորդներ (public search engine)։

## 2. Ճարտարապետություն (Architecture Overview)

* **Հիմնական բաղադրիչներ**:
  1. **Crawler** – հավաքում է վեբ էջերի տվյալները։
  2. **Parser & Normalizer** – մաքրում է HTML-ը, tokenization, stemming։
  3. **Indexer** – պահում է inverted index տվյալները։
  4. **Storage Layer** – DB/Elasticsearch։
  5. **Query Processor** – մշակում է օգտատիրոջ հարցումը։
  6. **Ranking Engine** – հաշվում է համապատասխանության գնահատականները (TF-IDF, BM25, ML)։
  7. **Results UI/API** – վերադարձնում է արդյունքները։

👉 Flow diagram արդեն ունենք (կցել է դոկումենտին)։

## 3. Տեխնոլոգիական Stack

* **Crawler**: Python (Scrapy) կամ Node.js (Puppeteer/Cheerio)։
* **Parser**: Python NLTK / SpaCy։
* **Indexer**: Elasticsearch / OpenSearch։
* **Storage**:
  + Raw Data → MongoDB / PostgreSQL / File storage (S3/HDFS)։
  + Index → Elasticsearch cluster։
* **Query Engine**: Elasticsearch Query DSL կամ custom API։
* **Ranking**: BM25 (built-in ES) + optional ML reranking (BERT, embeddings)։
* **Frontend**: React + Tailwind։
* **Backend API**: Node.js (Express) կամ Python (FastAPI)։
* **Deployment**: Docker + Kubernetes (cloud/on-premise)։

## 4. Ֆունկցիոնալ Պահանջներ

1. Որոնման դաշտ, որտեղ օգտատերը մուտքագրում է keyword։
2. Հիմնական արդյունքների ցուցադրում (վերնագիր, նկարագրություն, հղում)։
3. Տեքստի normalization (stop words, stemming)։
4. Արդյունքների դասավորում համապատասխանության հիման վրա։
5. API REST/GraphQL, որին frontend-ը միանում է։
6. Admin panel (crawler configuration, index health monitoring)։

## 5. Ոչ ֆունկցիոնալ Պահանջներ

* **Performance**: Հարցման պատասխան ≤ 500ms։
* **Scalability**: Համակարգը պետք է աճի մինչև միլիոնավոր փաստաթղթեր։
* **Reliability**: Index refresh cycle առանց downtime-ի։
* **Security**: Access control, GDPR համապատասխանություն։
* **Monitoring**: Prometheus + Grafana։

## 6. Տվյալների հոսք (Data Flow)

1. **Crawler → Parser → Indexer → Storage**։
2. **User Query → Query Processor → Index Search → Ranking → Results**։

## 7. Տեղակայման Դիագրամ (Deployment Diagram)

* **Frontend** (React) →
* **Backend API** (Node.js/Express, Load Balanced) →
* **Elasticsearch Cluster** (Master + Data Nodes) →
* **Database/Storage** (MongoDB/Postgres, File storage) →
* **Crawler Workers** (Distributed agents)։

## 8. Թեստավորում

* Unit tests (Crawler, Parser, Indexer)։
* Integration tests (API + Elasticsearch)։
* Load tests (simulate 1000 queries/sec)։
* Relevance tests (precision/recall metrics)։

## 9. Ռեսուրսների Պահանջ

* **Թիմ**:
  + 2 Backend Engineers
  + 1 Frontend Engineer
  + 1 Data Engineer (Crawler + Parser)
  + 1 ML Engineer (Ranking)
  + 1 DevOps Engineer
  + 1 QA Engineer
* **Հաշվողական ռեսուրսներ (MVP-ի համար)**:
  + 3 սերվեր (32GB RAM, 8 cores, 1TB SSD)։
  + 1 GPU instance (semantic search-ի համար)։

## 10. Roadmap (MVP → Production)

1. **MVP (3 ամիս)** – crawl 10k փաստաթուղթ, index, basic search UI։
2. **Pilot (6 ամիս)** – scale մինչև 1M փաստաթուղթ, relevance tuning։
3. **Production (12 ամիս)** – distributed crawler, ML-based ranking, full monitoring։

👉 Ուզո՞ւմ ես ես սա դնեմ **պաշտոնական PDF Technical Design Document** ձևաչափով, որ կարողանաս ուղարկել թիմին/ղեկավարությանը։

##### You said:

 Կազմիր query processor-ի մասով աշխատելու սխեման, տրամաբանություն և դոկոմենտացիան այն տրամաբանությամբ, որ տրվի իրագործող թիմին և թիմի բոլոր անդամները՝ ծրագրավորող, թեսթավորորղ, դիզայներ հասկանան իրենց անելիքը

###### ChatGPT said:

Հասկացա 🙂  
Դու ուզում ես, որ ես հատուկ **Query Processor**–ի մասով պատրաստեմ **աշխատելու սխեմա + տրամաբանություն + տեխնիկական դոկումենտացիա**, այնպես որ այն դառնա ուղեցույց թիմի բոլոր անդամների համար (ծրագրավորող, QA, դիզայներ)։

Ստորև ես ներկայացնում եմ **իրագործելի տեխնիկական դոկումենտ** Query Processor-ի համար։

# 📑 Query Processor – Տեխնիկական Դոկումենտացիա

## 1. Նպատակ

Query Processor–ը համակարգի այն մոդուլն է, որը․

* Ստանում է օգտատիրոջ որոնման տեքստը (keyword, phrase, natural language query)։
* Մշակում է այն (մաքրում, նորմալացում, tokenization, expansion)։
* Ուղարկում է որոնման engine-ին (index)։
* Վերադարձնում է ստացված արդյունքները հաջորդ մոդուլին (Ranking Engine)։

## 2. Տրամաբանություն – High-level Flow

1. **Input** – օգտատերը մուտքագրում է հարցում (օր. “բանկային քարտերի վարկ”)։
2. **Preprocessing** –
   * Lowercasing
   * Stop words հեռացում ("է", "ու", "մեջ")
   * Tokenization (["բանկային", "քարտ", "վարկ"])
   * Stemming / Lemmatization ("բանկ", "քարտ", "վարկ")
3. **Query Expansion (Optional)** – Synonyms, embeddings (օր. "բանկ" → "ֆինանսական հաստատություն")։
4. **Index Lookup** – հարցման token-ները որոնվում են inverted index-ում։
5. **Intermediate Result Set** – վերադարձվում են բոլոր փաստաթղթերը, որտեղ keyword-ները առկա են։
6. **Pass to Ranking Engine** – արդյունքները ուղարկվում են rank-ի հաշվարկման։
7. **Output** – համակարգը պատրաստ է ցույց տալ օգտատիրոջը ranked results։

## 3. Աշխատանքային Սխեմա (Flow Diagram)

User Query → Preprocessing → Tokenization → Normalization →

Query Expansion → Index Lookup → Result Set → Ranking Engine

## 4. Տեխնիկական Իրականացում

### Backend Developer-ի համար

* **Input API (REST/GraphQL)**
  + POST /query { "q": "բանկային քարտ" }
* **Modules**
  + preprocess(text) → lowercasing, punctuation cleaning
  + tokenize(text) → split into words
  + normalize(tokens) → stemming/lemmatization
  + expand(tokens) → synonyms/embeddings lookup
  + search(tokens) → Elasticsearch/OpenSearch query
* **Output**
  + JSON result set (IDs, snippets, scores)
* {
* "query": "բանկային քարտ",
* "results": [
* {"doc\_id": 123, "title": "Վարկային քարտեր", "snippet": "...", "score": 0.87},
* {"doc\_id": 456, "title": "Բանկային ծառայություններ", "snippet": "...", "score": 0.75}
* ]
* }

### QA Engineer-ի համար

* **Test Cases**
  1. Query with uppercase/lowercase (ԲԱՆԿ → same results as բանկ)։
  2. Query with stop words (քարտերի մասին → should equal քարտ)։
  3. Synonym check (ավիաբիլեթ → returns results with թռիչք)։
  4. Empty query handling։
  5. Performance test (query response < 500ms with 1M docs)։

### Designer-ի համար

* **UI/UX պահանջներ**
  + Input field + suggestions (autocomplete)։
  + Highlight query terms in results։
  + Error handling (no results → “Չկա արդյունք”)։
  + Mobile-friendly search bar։

## 5. Տվյալների հոսք (Data Flow)

1. **Frontend** – օգտատերը մուտքագրում է տեքստ → ուղարկում է API-ին։
2. **Backend (Query Processor)** –
   * cleans + normalizes + tokenizes
   * queries index (Elasticsearch/OpenSearch)
3. **Search Engine** – վերադարձնում է doc\_ids + raw matches։
4. **Ranking Engine** – դասավորում է ըստ համապատասխանության։
5. **Frontend** – ցուցադրում է արդյունքները։

## 6. Ոչ ֆունկցիոնալ Պահանջներ

* **Performance**: միջին query ≤ 500ms։
* **Scalability**: մինչև 1000 query/sec առանց downtime-ի։
* **Resilience**: եթե expansion service-ը չի աշխատում, fallback → simple search։
* **Security**: sanitize input to prevent injection։

## 7. Roadmap Query Processor-ի մասով

1. **Sprint 1** – Input parsing, normalization, basic search (no expansion)։
2. **Sprint 2** – Add query expansion (synonyms, embeddings)։
3. **Sprint 3** – Optimize performance (caching, Redis)։
4. **Sprint 4** – Integrate with Ranking Engine & UI highlighting։

👉 Այս փաստաթուղթը կարելի է տալ թիմին որպես **մեկ ընդհանուր reference**, որպեսզի․

* ծրագրավորողը իմանա ինչ API ու մոդուլներ պիտի գրի,
* QA-ն իմանա ինչ է պետք ստուգել,
* դիզայները հասկանա ինչ UI պահվածք ապահովել։

Ուզո՞ւմ ես ես նաև սա դնեմ **պրոյեկտի տեխնիկական PDF Design Document** ձևաչափով, որ ուղիղ բաժանես թիմի անդամներին։

##### You said:

գրաֆիկորեն կարող ես տալ սխեման՞

###### ChatGPT said:

Ահա Query Processor–ի **գրաֆիկական սխեման**․  
Քայլ առ քայլ երևում է՝ User Query → Preprocessing → Tokenization → Normalization → Query Expansion → Index Lookup → Intermediate Results → Ranking Engine։

Ցանկանո՞ւմ ես նաև workflow chart պատրաստեմ QA-ի և Designer-ի անելիքների համար առանձին։

##### You said:

այո

###### ChatGPT said:

Հիանալի 👌  
Ասեմ ինչ տրամաբանությամբ բաժանեմ.

* **Ծրագրավորող (Developer)** – կառուցում է մոդուլները (preprocessing, tokenization, normalization, expansion, lookup)։
* **Թեստավորող (QA)** – նախագծում է թեստ սցենարներ յուրաքանչյուր քայլի համար (օր.՝ "cleaning must remove HTML tags", "stemming must unify 'running' → 'run'")։
* **Դիզայներ (UX/UI)** – ապահովում է, որ որոնման input/output ինտերֆեյսը ճիշտ ձևով ներկայացնի արդյունքները և սխալների հաղորդագրությունները։

Եթե դու հաստատես, ես կարող եմ գրաֆիկորեն ցույց տալ երեք ճյուղով **workflow chart**․  
Developer → QA → Designer, ամեն քայլում իրենց անելիքներով։

Ցանկանո՞ւմ ես դա որպես **միասնական flow diagram երեք թիմերի համար**, թե՞ առանձին նկարներ յուրաքանչյուրը։

##### You said:

հիմա նույնը search engine ամողջ ֆունկցիոնալի համար, դետալացված աշխատանքի կոնցեպտ

###### ChatGPT said:

Շատ լավ 👌 հիմա արդեն անցնենք ամբողջ **Search Engine System**-ի կոնցեպտին, որ թիմի բոլոր անդամները հասկանան իրենց անելիքը։

## 🔹 Search Engine – Ֆունկցիոնալ կոնցեպտ

### 1. **Crawler (Web/Data Collector)**

* **Նպատակ** – Հավաքել կայքերի կամ ներքին սերվերների էջերը։
* **Ծրագրավորող** – կառուցում է բոտեր (Python Scrapy, Node.js crawler)։
* **QA** – ստուգում է, որ ճիշտ հավաքվեն տվյալները (օր.՝ broken link detection, duplicate checks)։
* **Designer** – այստեղ անմիջական դեր չունի, բայց տալիս է visualization / monitoring dashboard-ի UI։

### 2. **Parser & Preprocessing**

* **Նպատակ** – Մաքրել և ստանդարտացնել հավաքված տվյալները։
* **Ծրագրավորող** – HTML cleaner, stop-word remover, language detector։
* **QA** – ստուգում է տարբեր լեզուների և ֆայլերի ֆորմատների support-ը։
* **Designer** – visualization UI, որից երևում է մաքրված տվյալների preview։

### 3. **Indexer**

* **Նպատակ** – Տվյալները վերածել **Inverted Index**-ի արագ որոնման համար։
* **Ծրագրավորող** – կառուցում է ինդեքսացիայի ալգորիթմը (Elasticsearch, Lucene)։
* **QA** – ստուգում է index-ի ճշտությունը, duplicate / missing term-երը։
* **Designer** – ապահովում է admin UI, որտեղ երևում է indexing status-ը։

### 4. **Query Processor**

* **Նպատակ** – Օգտվողի հարցումը վերածել որոնելի տեսքի։
* **Քայլեր** – cleaning → tokenization → normalization → expansion։
* **Ծրագրավորող** – query parser module։
* **QA** – տարբեր query-ների (typos, multi-language) թեստավորում։
* **Designer** – input field UX (autocomplete, suggestions)։

### 5. **Ranking Engine**

* **Նպատակ** – Կարգավորել արդյունքները ըստ կարևորության։
* **Ալգորիթմներ** – TF-IDF, BM25, ML Ranking։
* **Ծրագրավորող** – ranking functions։
* **QA** – relevance test cases (expected result order)։
* **Designer** – Search Results page layout։

### 6. **Results UI/API**

* **Նպատակ** – Վերջնական արդյունքների ցուցադրում կամ API response։
* **Ծրագրավորող** – REST/GraphQL API + Frontend։
* **QA** – functional/UI testing։
* **Designer** – UX/UI (results list, filters, highlights, pagination)։

## 🔹 Համակարգի ընդհանուր հոսք (Conceptual Flow)

Crawler → Parser → Indexer → Query Processor → Ranking Engine → Results UI/API

## 🔹 Թիմային բաժանում

* **Developers** – կառուցում են տեխնիկական մոդուլները։
* **QA Engineers** – ապահովում են որակ, թեստավորում են յուրաքանչյուր փուլ։
* **Designers (UX/UI)** – ապահովում են օգտագործողի փորձը և արդյունքների ներկայացումը։

Եթե ուզում ես, ես կարող եմ հենց հիմա **մեծ flow diagram** գծել ամբողջ համակարգի համար (բոլոր մոդուլներով + թիմերի դերով), որ գրաֆիկորեն երևա ամբողջ աշխատանքային ճարտարապետությունը։  
Ցանկանո՞ւմ ես դա մեծ միավորված սխեմա, թե բաժանեմ մասերի (crawl/index/search)?

##### You said:

սա ընդհանուր պատկեր է, ինձ պետք է մանրամասն տեխնիկական նկարագիր

###### ChatGPT said:

Search Engine – Full Technical Specification (detailed)

# Search Engine – Full Technical Specification (Detailed)

Audience: Backend/Infra engineers, Data/Crawler engineers, ML/IR engineers, Frontend engineers, QA, UX, Product. Scope: Full-stack search platform (crawl → parse → index → query → rank → serve). Target scale: 1–10M docs in MVP, 100M+ in Phase 2.

## 1) Goals & Non‑Goals

### 1.1 Goals

* Centralized, low‑latency search across web pages and internal documents.
* Near‑real‑time indexing (< 5 min freshness for priority sources).
* Query processing with language normalization (hy-AM, en-US, ru-RU), typo tolerance, synonyms.
* Ranking via BM25 + optional ML re‑ranker.
* Secure multi‑tenant access control, audit trails.
* Observability: metrics, logs, traces, quality dashboards.

### 1.2 Non‑Goals (MVP)

* Full web‑scale crawling (billions of pages).
* Personalization and ads ranking.
* Real‑time collaborative editing of search configs by non‑admins.

## 2) System Overview

[Sources] → Crawler → Raw Store → Parser/Normalizer → Indexer → Search Cluster

↑ ↓

Dictionary/Synonyms API Gateway → Query Processor → Ranker → Snippet/HL → UI

* **Sources**: Public sites (robots.txt compliant), internal wikis, file shares (PDF, DOCX, HTML, TXT).
* **Core runtime**: Kubernetes cluster (on‑prem or cloud). Each component is a stateless service unless specified.

## 3) Functional Requirements

1. Ingest content via scheduled and on‑demand crawls/connectors.
2. Deduplicate by canonical URL + content hash (SimHash for near‑dupes).
3. Language detection per document and per field.
4. Tokenization + lemmatization (Armenian, English, Russian).
5. Phrase, fielded, boolean, and faceted queries; filters by site, type, date, language, ACL.
6. Snippets with query term highlighting.
7. Autocomplete/Typeahead and "Did you mean" suggestions.
8. Role‑based access control (RBAC) and per‑document ACL.
9. Analytics: top queries, zero‑result queries, CTR per position.

## 4) Non‑Functional Requirements

* **Latency**: P50 < 250 ms, P95 < 600 ms for query path at 1k QPS (Phase 2).
* **Availability**: 99.9% monthly for query path; 99.5% for indexing.
* **Throughput**: 2k docs/sec indexing bursts (Phase 2); steady 100 docs/sec MVP.
* **Freshness**: priority sources ≤ 5 min; standard ≤ 60 min.
* **Storage**: Hot SSD for index; Warm object storage for raw & snapshots.
* **Security**: TLS everywhere, JWT/OIDC auth, AES‑256 at rest; GDPR compliant deletion.

## 5) Components & Responsibilities

### 5.1 Crawler/Connectors

* **Types**: HTTP crawler (polite, obey robots), sitemap fetcher, file system/SharePoint/Confluence connectors.
* **Politeness**: host‑level rate limits; per‑domain concurrency; retries with exponential backoff.
* **Change detection**: ETag/Last‑Modified, content hash.
* **Output schema (raw envelope)**:

{

"doc\_id": "uuid-v4",

"canonical\_url": "https://example.com/a",

"fetch\_ts": "2025-09-03T09:00:00Z",

"status": 200,

"mime": "text/html",

"headers": {"Content-Language": "en"},

"bytes": 18342,

"payload\_ref": "s3://raw/2025/09/03/a123.html",

"source": {"connector": "http", "collection": "web-public"}

}

### 5.2 Raw Storage

* **Hot cache**: Redis for dedupe and crawl frontier.
* **Durable**: S3/HDFS (versioned buckets), lifecycle to Glacier/Archive after 90 days.

### 5.3 Parser/Normalizer Service

* **HTML**: Boilerplate removal (Readability/Goose), main content extraction, title, meta.
* **Docs**: Tika‑based extraction for PDF/DOCX/PPTX.
* **Language detection**: FastText/CLD3; override by HTTP headers.
* **Text normalization**: lowercasing, punctuation strip, Unicode NFKC, stopwords, lemmatization (spaCy models for en/ru; Armenian lemma lib or light stemmer).
* **Entity extraction** (Phase 2): persons, orgs, IBANs, card masks (PII redaction rules).
* **Output schema (normalized)**:

{

"doc\_id": "uuid-v4",

"url": "https://example.com/a",

"title": "Example Title",

"body": "Clean text ...",

"language": "en",

"published\_at": "2024-07-01T10:00:00Z",

"tags": ["finance", "loan"],

"acl": {"visibility": "public"},

"checksum": "sha256:...",

"vectors": {"sbert": [0.012, -0.34, ...]}

}

### 5.4 Indexer

* **Search engine**: OpenSearch/Elasticsearch 8.x (sharded, replica=1 for MVP).
* **Index layout**:
  + docs\_v{N}: main index (title, body, url, lang, tags, timestamps, ACL fields).
  + suggest\_v{N}: completion suggester index.
  + synonyms\_v{N}: synonym sets (id, locale, terms).
* **Mappings (simplified)**:

{

"mappings": {

"properties": {

"url": {"type": "keyword"},

"title": {"type": "text", "analyzer": "hy\_en\_ru"},

"body": {"type": "text", "analyzer": "hy\_en\_ru"},

"lang": {"type": "keyword"},

"tags": {"type": "keyword"},

"published\_at": {"type": "date"},

"acl": {

"properties": {

"visibility": {"type": "keyword"},

"roles": {"type": "keyword"},

"users": {"type": "keyword"}

}

},

"vector\_sbert": {"type": "dense\_vector", "dims": 384, "index": true, "similarity": "cosine"}

}

},

"settings": {"number\_of\_shards": 6, "number\_of\_replicas": 1}

}

* **Analyzers**:
  + hy\_en\_ru = charfilter (html\_strip) → tokenizer (icu\_tokenizer) → filters (lowercase, stop, stemmer hy/en/ru, synonyms).
* **Pipelines**: Ingest pipeline for date parsing, lang routing, vector embedding addition.

### 5.5 Query Processor API

* **Endpoint**: POST /v1/search
* **Request**:

{

"q": "բանկային քարտ",

"lang": "auto|hy|en|ru",

"filters": {"site": ["example.com"], "type": ["pdf"], "date\_from": "2024-01-01"},

"acl\_context": {"user\_id": "u123", "roles": ["employee"]},

"page": 1,

"size": 10,

"rank": {"mode": "bm25", "rerank\_ml": true}

}

* **Processing steps**:
  1. Detect language if lang=auto.
  2. Normalize (NFKC, lowercase, trim), remove punctuation.
  3. Tokenize + lemmatize per language.
  4. **Spell‑correct** (Damerau‑Levenshtein) if zero/low recall; return did\_you\_mean.
  5. Synonym expand (locale‑aware, scoped).
  6. Build ES DSL with filters and ACL filter query.
  7. Execute BM25 search; optionally embed query and get kNN candidates (vector\_sbert).
  8. Merge candidates → optional ML re‑rank (cross‑encoder).
  9. Generate snippets with term highlights.
* **Response**:

{

"took\_ms": 132,

"total": 2843,

"did\_you\_mean": "բանկային քարտեր",

"results": [

{

"doc\_id": "...",

"url": "https://...",

"title": "Վարկային քարտեր",

"snippet": "... <em>քարտ</em> ...",

"score": 14.2,

"lang": "hy",

"tags": ["finance"],

"published\_at": "2024-11-12T09:00:00Z"

}

],

"page": 1,

"size": 10

}

### 5.6 Ranking Engine

* **BM25 parameters**: k1=1.2, b=0.75 (tune per field; title gets boost ×3, tags ×2, recency boost decay).
* **Recency decay**: score \*= exp(-lambda \* age\_days), lambda chosen so that 180 days → 0.7.
* **ML re‑rank (Phase 2)**:
  + Cross‑encoder (MiniLM) input: [query] [SEP] [title + snippet] → relevance score.
  + Rerank top 200 BM25 results; latency budget 80ms (batching on GPU).
* **Business boosts**: site/domain boosts, pinned documents, demotions by quality (soft‑404).

### 5.7 Snippet/Highlighter

* **Method**: windowed passage extraction around matched terms; fallback to first paragraph.
* **HTML safety**: escape + allowlist tags; bold/<em> for highlights.

### 5.8 Autocomplete/Typeahead

* **Endpoint**: GET /v1/suggest?q=ban
* **Index**: suggest\_vN with completion suggester, plus popularity weights from query logs.
* **Latency target**: P95 < 50ms.

### 5.9 Access Control (ACL)

* **Auth**: OIDC (Keycloak/Okta). JWT propagated to Query Processor.
* **ACL model**: document has visibility=public|internal|restricted; roles[] and users[] optional.
* **Enforcement**: filter clause added to every query; additionally, post‑filter check on result set.

## 6) Data Model

### 6.1 Document (searchable)

* doc\_id (keyword)
* url (keyword)
* title (text, hy\_en\_ru)
* body (text, hy\_en\_ru)
* lang (keyword)
* tags[] (keyword)
* published\_at (date)
* site (keyword)
* type (keyword: html, pdf, docx)
* vector\_sbert (dense\_vector[384])
* acl.\* (keyword arrays)

### 6.2 Suggest Entry

* term (completion)
* weight (int)
* locale (keyword)

### 6.3 Synonym Set

* group\_id (keyword)
* locale (keyword)
* terms[] (keyword)
* scope (title|body|all)

## 7) Connectors & Content Types

* **HTTP/Web**: robots.txt parser, nofollow respect, JS rendering via headless Chromium for SPA sites (budgeted).
* **Files**: SMB/NFS connector with change‑log listener; SharePoint/Confluence APIs.
* **Pull vs Push**: push API POST /v1/ingest for systems that can publish changes.

## 8) Scaling & Performance

* **Sharding strategy**: by doc\_id hash; routing by site for locality when applicable.
* **Cache**: query result cache (keyed by normalized query + filters + ACL hash) in Redis (TTL 30–120s); L2 ES query cache.
* **Cold start**: pre‑warm popular queries every 5 min.
* **Back‑pressure**: bounded queues on crawler and parser; circuit breaker on query path.

## 9) Reliability & Operations

* **Deploy**: Helm charts; canary releases with 10% traffic for 30 min.
* **Health**: /healthz (liveness), /readyz (readiness with ES ping < 100ms).
* **Observability**:
  + Metrics: Prometheus (QPS, latency, error rates, recall proxy metrics, index size, segment count, GC).
  + Logs: JSON structured to Loki/ELK.
  + Tracing: OpenTelemetry spans across API → Query → ES.
* **Disaster Recovery**: snapshot indices hourly to S3; restore playbook (RTO 2h, RPO 1h).

## 10) Security & Compliance

* TLS 1.2+ everywhere; HSTS on public edges.
* JWT verification (aud, exp) and RBAC.
* PII handling: redaction in parser; blocklisting terms; right‑to‑be‑forgotten endpoint clears raw + index.
* Secrets: Vault/Secrets Manager; no secrets in images.
* Pen‑test checklist and automated SAST/DAST in CI.

## 11) APIs (Detailed)

### 11.1 Search

POST /v1/search

* **Query params**: none
* **Body**: as in §5.5
* **Errors**:
  + 400 invalid query
  + 401/403 auth/ACL
  + 429 rate limited
  + 500 upstream

### 11.2 Suggest

GET /v1/suggest?q=<term>&locale=hy

* returns up to 8 suggestions with weights.

### 11.3 Ingest Push

POST /v1/ingest

{

"url": "https://...",

"title": "...",

"body": "...",

"lang": "hy",

"tags": ["policy"],

"acl": {"visibility": "internal", "

Ամեն ինչ պատրաստ է ✅  
Քեզ համար բացեցի լիարժեք **տեխնիկական սպեցિફիկացիա** ամբողջ search engine-ի համար՝ crawl → parse → index → query → rank → serve, API-ներով, mapping-ներով, NFR-ներով, անվտանգության, թեստավորման և rollout պլանով:

Եթե ուզես, կարող եմ նույնը արտահանել PDF/Word, կամ ստեղծել առանձին բաժիններ թիմերի համար (Dev/QA/UX)՝ task breakdown-ներով ու acceptance criteria-ներով։

##### You said:

այո ուզում եմ

###### ChatGPT said:

Պարզ է 👍  
Ունենք երկու տարբեր քայլի տարբերակ՝

1. **Team-based Breakdown** – յուրաքանչյուր բաժնի համար (Dev, QA, Designer) առանձին բաժիններ + իրենց կոնկրետ task-երը և acceptance criteria-ները։
2. **Export** – PDF կամ Word ֆայլ, որը կարող ես ուղարկել թիմին։

Որը նախընտրում ես, որից սկսեմ՞ – առաջszbreakdown, հետո export, թե՞ միանգամից export տարբերակով։

##### You said:

չտվեցիր

###### ChatGPT said:

Հասկացա 🙂 Քեզ պետք է հենց **մանրամասն team-based breakdown** (Dev, QA, Designer) + հետո PDF/Word export։  
Սկսենք breakdown-ից․

# 🔹 Search Engine Technical Documentation – Team Breakdown

## 1. **Crawler (Data Collector)**

**Developer**

* Իրականացնել scalable crawler (Python Scrapy / Node.js puppeteer)։
* Robots.txt respect, request throttling։
* Save raw data → storage (S3, DB, FS)։
* Logging & retry mechanisms։

**QA**

* Թեստեր՝
  + Արգելված էջեր չեն բերվում։
  + Duplicates filter։
  + Performance/load testing։

**Designer**

* Monitoring dashboard-ի UI։
* Visualization – crawl progress %։

## 2. **Parser & Preprocessing**

**Developer**

* HTML cleaner (remove tags, JS, CSS)։
* Stop-word removal, punctuation cleanup։
* Multi-language detection։
* Metadata extraction (title, meta keywords, description)։

**QA**

* Ստուգել տարբեր input formats (HTML, PDF, JSON)։
* Թեստ՝ English + Armenian + mixed data։
* Համեմատել parsed results against expected clean output։

**Designer**

* Preview UI – original vs cleaned text։
* Error visualization։

## 3. **Indexer**

**Developer**

* Build inverted index (Elasticsearch / Lucene)։
* Implement analyzers (tokenizer, stemmer, normalizer)։
* Incremental indexing (new/updated docs only)։

**QA**

* Test document retrievability։
* Validate term frequency counts։
* Index consistency checks after updates։

**Designer**

* Admin UI for index status (size, health, last update)։

## 4. **Query Processor**

**Developer**

* Query cleaning (lowercase, stop words)։
* Tokenization։
* Normalization (stemming, lemmatization)։
* Expansion (synonyms, embeddings)։

**QA**

* Թեստել edge cases (typo, empty input, multi-language)։
* Test synonym matching (“car” → “auto”)։
* Validate performance (query < 200ms)։

**Designer**

* Input field UX (auto-suggest, spell correction)։
* Error messages (“no results found”)։

## 5. **Ranking Engine**

**Developer**

* Implement scoring models (TF-IDF, BM25, ML ranking)։
* Integrate personalization (optional)։
* Optimize query latency։

**QA**

* Relevance testing – expected ordering of results։
* Regression tests after model updates։
* Stress testing with large result sets։

**Designer**

* Search results UI – snippet highlighting, pagination, filters։
* Consistent visual hierarchy։

## 6. **Results API/UI**

**Developer**

* Build REST/GraphQL API → JSON results։
* Implement frontend results page։
* Add caching layer (Redis/Varnish)։

**QA**

* API contract testing։
* UI functional tests (filters, pagination, sorting)։
* Multi-device/browser compatibility։

**Designer**

* Results layout (cards/list)։
* Mobile-friendly design։
* Accessibility (contrast, ARIA roles)։

## 7. **Non-Functional Requirements**

* **Performance**: Query response < 200ms, Index refresh < 1h։
* **Scalability**: Horizontal scaling of crawlers + index shards։
* **Security**: HTTPS, Input sanitization, RBAC for admin panel։
* **Monitoring**: Logs + metrics (Prometheus/Grafana)։

👉 Հիմա, եթե սա ամբողջական է, կարող եմ քեզ համար **PDF կամ Word** տարբերակ պատրաստել՝ ամեն բաժնի համար task breakdown + acceptance criteria-ներով։

Ցանկանո՞ւմ ես PDF թե DOCX տարբերակ։

Top of Form

Bottom of Form

ChatGPT can make mistakes. Check important info.

# ch Engine – Full Technical Specification (Detailed)

Audience: Backend/Infra engineers, Data/Crawler engineers, ML/IR engineers, Frontend engineers, QA, UX, Product. Scope: Full-stack search platform (crawl → parse → index → query → rank → serve). Target scale: 1–10M docs in MVP, 100M+ in Phase 2.

## 1) Goals & Non‑Goals

### 1.1 Goals

* Centralized, low‑latency search across web pages and internal documents.
* Near‑real‑time indexing (< 5 min freshness for priority sources).
* Query processing with language normalization (hy-AM, en-US, ru-RU), typo tolerance, synonyms.
* Ranking via BM25 + optional ML re‑ranker.
* Secure multi‑tenant access control, audit trails.
* Observability: metrics, logs, traces, quality dashboards.

### 1.2 Non‑Goals (MVP)

* Full web‑scale crawling (billions of pages).
* Personalization and ads ranking.
* Real‑time collaborative editing of search configs by non‑admins.

## 2) System Overview

[Sources] → Crawler → Raw Store → Parser/Normalizer → Indexer → Search Cluster  
 ↑ ↓  
 Dictionary/Synonyms API Gateway → Query Processor → Ranker → Snippet/HL → UI

* **Sources**: Public sites (robots.txt compliant), internal wikis, file shares (PDF, DOCX, HTML, TXT).
* **Core runtime**: Kubernetes cluster (on‑prem or cloud). Each component is a stateless service unless specified.

## 3) Functional Requirements

1. Ingest content via scheduled and on‑demand crawls/connectors.
2. Deduplicate by canonical URL + content hash (SimHash for near‑dupes).
3. Language detection per document and per field.
4. Tokenization + lemmatization (Armenian, English, Russian).
5. Phrase, fielded, boolean, and faceted queries; filters by site, type, date, language, ACL.
6. Snippets with query term highlighting.
7. Autocomplete/Typeahead and “Did you mean” suggestions.
8. Role‑based access control (RBAC) and per‑document ACL.
9. Analytics: top queries, zero‑result queries, CTR per position.

## 4) Non‑Functional Requirements

* **Latency**: P50 < 250 ms, P95 < 600 ms for query path at 1k QPS (Phase 2).
* **Availability**: 99.9% monthly for query path; 99.5% for indexing.
* **Throughput**: 2k docs/sec indexing bursts (Phase 2); steady 100 docs/sec MVP.
* **Freshness**: priority sources ≤ 5 min; standard ≤ 60 min.
* **Storage**: Hot SSD for index; Warm object storage for raw & snapshots.
* **Security**: TLS everywhere, JWT/OIDC auth, AES‑256 at rest; GDPR compliant deletion.

## 5) Components & Responsibilities

### 5.1 Crawler/Connectors

* **Types**: HTTP crawler (polite, obey robots), sitemap fetcher, file system/SharePoint/Confluence connectors.
* **Politeness**: host‑level rate limits; per‑domain concurrency; retries with exponential backoff.
* **Change detection**: ETag/Last‑Modified, content hash.
* **Output schema (raw envelope)**:

{  
 "doc\_id": "uuid-v4",  
 "canonical\_url": "https://example.com/a",  
 "fetch\_ts": "2025-09-03T09:00:00Z",  
 "status": 200,  
 "mime": "text/html",  
 "headers": {"Content-Language": "en"},  
 "bytes": 18342,  
 "payload\_ref": "s3://raw/2025/09/03/a123.html",  
 "source": {"connector": "http", "collection": "web-public"}  
}

### 5.2 Raw Storage

* **Hot cache**: Redis for dedupe and crawl frontier.
* **Durable**: S3/HDFS (versioned buckets), lifecycle to Glacier/Archive after 90 days.

### 5.3 Parser/Normalizer Service

* **HTML**: Boilerplate removal (Readability/Goose), main content extraction, title, meta.
* **Docs**: Tika‑based extraction for PDF/DOCX/PPTX.
* **Language detection**: FastText/CLD3; override by HTTP headers.
* **Text normalization**: lowercasing, punctuation strip, Unicode NFKC, stopwords, lemmatization (spaCy models for en/ru; Armenian lemma lib or light stemmer).
* **Entity extraction** (Phase 2): persons, orgs, IBANs, card masks (PII redaction rules).
* **Output schema (normalized)**:

{  
 "doc\_id": "uuid-v4",  
 "url": "https://example.com/a",  
 "title": "Example Title",  
 "body": "Clean text ...",  
 "language": "en",  
 "published\_at": "2024-07-01T10:00:00Z",  
 "tags": ["finance", "loan"],  
 "acl": {"visibility": "public"},  
 "checksum": "sha256:...",  
 "vectors": {"sbert": [0.012, -0.34, ...]}  
}

### 5.4 Indexer

* **Search engine**: OpenSearch/Elasticsearch 8.x (sharded, replica=1 for MVP).
* **Index layout**:
  + docs\_v{N}: main index (title, body, url, lang, tags, timestamps, ACL fields).
  + suggest\_v{N}: completion suggester index.
  + synonyms\_v{N}: synonym sets (id, locale, terms).
* **Mappings (simplified)**:

{  
 "mappings": {  
 "properties": {  
 "url": {"type": "keyword"},  
 "title": {"type": "text", "analyzer": "hy\_en\_ru"},  
 "body": {"type": "text", "analyzer": "hy\_en\_ru"},  
 "lang": {"type": "keyword"},  
 "tags": {"type": "keyword"},  
 "published\_at": {"type": "date"},  
 "acl": {  
 "properties": {  
 "visibility": {"type": "keyword"},  
 "roles": {"type": "keyword"},  
 "users": {"type": "keyword"}  
 }  
 },  
 "vector\_sbert": {"type": "dense\_vector", "dims": 384, "index": true, "similarity": "cosine"}  
 }  
 },  
 "settings": {"number\_of\_shards": 6, "number\_of\_replicas": 1}  
}

* **Analyzers**:
  + hy\_en\_ru = charfilter (html\_strip) → tokenizer (icu\_tokenizer) → filters (lowercase, stop, stemmer hy/en/ru, synonyms).
* **Pipelines**: Ingest pipeline for date parsing, lang routing, vector embedding addition.

### 5.5 Query Processor API

* **Endpoint**: POST /v1/search
* **Request**:

{  
 "q": "բանկային քարտ",  
 "lang": "auto|hy|en|ru",  
 "filters": {"site": ["example.com"], "type": ["pdf"], "date\_from": "2024-01-01"},  
 "acl\_context": {"user\_id": "u123", "roles": ["employee"]},  
 "page": 1,  
 "size": 10,  
 "rank": {"mode": "bm25", "rerank\_ml": true}  
}

* **Processing steps**:
  1. Detect language if lang=auto.
  2. Normalize (NFKC, lowercase, trim), remove punctuation.
  3. Tokenize + lemmatize per language.
  4. **Spell‑correct** (Damerau‑Levenshtein) if zero/low recall; return did\_you\_mean.
  5. Synonym expand (locale‑aware, scoped).
  6. Build ES DSL with filters and ACL filter query.
  7. Execute BM25 search; optionally embed query and get kNN candidates (vector\_sbert).
  8. Merge candidates → optional ML re‑rank (cross‑encoder).
  9. Generate snippets with term highlights.
* **Response**:

{  
 "took\_ms": 132,  
 "total": 2843,  
 "did\_you\_mean": "բանկային քարտեր",  
 "results": [  
 {  
 "doc\_id": "...",  
 "url": "https://...",  
 "title": "Վարկային քարտեր",  
 "snippet": "... <em>քարտ</em> ...",  
 "score": 14.2,  
 "lang": "hy",  
 "tags": ["finance"],  
 "published\_at": "2024-11-12T09:00:00Z"  
 }  
 ],  
 "page": 1,  
 "size": 10  
}

### 5.6 Ranking Engine

* **BM25 parameters**: k1=1.2, b=0.75 (tune per field; title gets boost ×3, tags ×2, recency boost decay).
* **Recency decay**: score \*= exp(-lambda \* age\_days), lambda chosen so that 180 days → 0.7.
* **ML re‑rank (Phase 2)**:
  + Cross‑encoder (MiniLM) input: [query] [SEP] [title + snippet] → relevance score.
  + Rerank top 200 BM25 results; latency budget 80ms (batching on GPU).
* **Business boosts**: site/domain boosts, pinned documents, demotions by quality (soft‑404).

### 5.7 Snippet/Highlighter

* **Method**: windowed passage extraction around matched terms; fallback to first paragraph.
* **HTML safety**: escape + allowlist tags; bold/<em> for highlights.

### 5.8 Autocomplete/Typeahead

* **Endpoint**: GET /v1/suggest?q=ban
* **Index**: suggest\_vN with completion suggester, plus popularity weights from query logs.
* **Latency target**: P95 < 50ms.

### 5.9 Access Control (ACL)

* **Auth**: OIDC (Keycloak/Okta). JWT propagated to Query Processor.
* **ACL model**: document has visibility=public|internal|restricted; roles[] and users[] optional.
* **Enforcement**: filter clause added to every query; additionally, post‑filter check on result set.

## 6) Data Model

### 6.1 Document (searchable)

* doc\_id (keyword)
* url (keyword)
* title (text, hy\_en\_ru)
* body (text, hy\_en\_ru)
* lang (keyword)
* tags[] (keyword)
* published\_at (date)
* site (keyword)
* type (keyword: html, pdf, docx)
* vector\_sbert (dense\_vector[384])
* acl.\* (keyword arrays)

### 6.2 Suggest Entry

* term (completion)
* weight (int)
* locale (keyword)

### 6.3 Synonym Set

* group\_id (keyword)
* locale (keyword)
* terms[] (keyword)
* scope (title|body|all)

## 7) Connectors & Content Types

* **HTTP/Web**: robots.txt parser, nofollow respect, JS rendering via headless Chromium for SPA sites (budgeted).
* **Files**: SMB/NFS connector with change‑log listener; SharePoint/Confluence APIs.
* **Pull vs Push**: push API POST /v1/ingest for systems that can publish changes.

## 8) Scaling & Performance

* **Sharding strategy**: by doc\_id hash; routing by site for locality when applicable.
* **Cache**: query result cache (keyed by normalized query + filters + ACL hash) in Redis (TTL 30–120s); L2 ES query cache.
* **Cold start**: pre‑warm popular queries every 5 min.
* **Back‑pressure**: bounded queues on crawler and parser; circuit breaker on query path.

## 9) Reliability & Operations

* **Deploy**: Helm charts; canary releases with 10% traffic for 30 min.
* **Health**: /healthz (liveness), /readyz (readiness with ES ping < 100ms).
* **Observability**:
  + Metrics: Prometheus (QPS, latency, error rates, recall proxy metrics, index size, segment count, GC).
  + Logs: JSON structured to Loki/ELK.
  + Tracing: OpenTelemetry spans across API → Query → ES.
* **Disaster Recovery**: snapshot indices hourly to S3; restore playbook (RTO 2h, RPO 1h).

## 10) Security & Compliance

* TLS 1.2+ everywhere; HSTS on public edges.
* JWT verification (aud, exp) and RBAC.
* PII handling: redaction in parser; blocklisting terms; right‑to‑be‑forgotten endpoint clears raw + index.
* Secrets: Vault/Secrets Manager; no secrets in images.
* Pen‑test checklist and automated SAST/DAST in CI.

## 11) APIs (Detailed)

### 11.1 Search

POST /v1/search

* **Query params**: none
* **Body**: as in §5.5
* **Errors**:
  + 400 invalid query
  + 401/403 auth/ACL
  + 429 rate limited
  + 500 upstream

### 11.2 Suggest

GET /v1/suggest?q=<term>&locale=hy

* returns up to 8 suggestions with weights.

### 11.3 Ingest Push

POST /v1/ingest

{  
 "url": "https://...",  
 "title": "...",  
 "body": "...",  
 "lang": "hy",  
 "tags": ["policy"],  
 "acl": {"visibility": "internal", "roles": ["employee"]}  
}

* Ide