ENTERPRISE SEARCH PLATFORM

TURING

Intelligent Search & AI

DOCUMENTATION

v 2026.1

March 21, 2026

viglet viglet.com

VIGLET PRODUCT DOCUMENTATION

Turing ES

Enterprise Search Platform

PRODUCT	Turing ES — Enterprise Search
VERSION	2026.1
GENERATED	March 21, 2026
WEBSITE	viglet.com
SOURCE	github.com/openturing
LICENSE	Apache License 2.0

Turing ES is an enterprise search platform that combines powerful indexing, natural language processing, and AI-powered retrieval to deliver intelligent search experiences. This document covers architecture, installation, configuration, connectors, and the complete API reference.

Table of Contents

What is Turing ES?

Viglet Turing ES is an open-source enterprise search platform. It helps organizations make their content findable, understandable, and interactive — through keyword search, faceted navigation, and generative AI conversations.

Whether you have thousands of documents on a file server, pages on a CMS, records in a database, or a combination of all of them, Turing ES indexes everything into a single search experience that your users can explore naturally.

What can you do with Turing ES?

Search your content

Index content from multiple sources and expose a unified, faceted search experience. Users can filter results by category, date, author, or any attribute you define — and always find what they are looking for.

Ask questions, get answers

Enable generative AI on your search site and let users ask questions in natural language. Turing ES retrieves the most relevant documents and uses an LLM to generate a grounded, accurate response — not a hallucination.

Build AI Agents

Compose AI Agents that combine a language model with a curated set of tools: search your content, browse the web, run code, query financial data, call external systems via MCP, and more. Each agent appears as a chat tab ready to assist users.

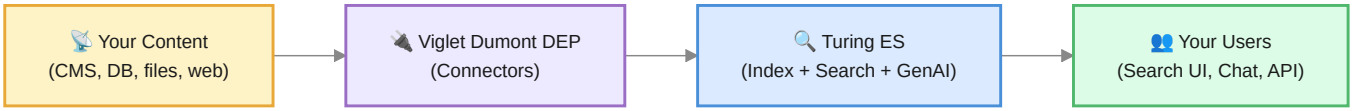
Connect any content source

Turing ES receives content from **Viglet Dumont DEP**, a companion application that provides connectors for web crawlers, databases, file systems, AEM, and WordPress. Connectors run independently and push content to Turing ES through its REST API.

Integrate with any stack

Consume Turing ES from your application via **REST API**, **GraphQL**, the **Java SDK** (available on Maven Central), or the **JavaScript / TypeScript SDK** `@viglet/turing-sdk` (available on npm).

How it works at a glance



Content flows from its sources through Dumont DEP connectors into Turing ES, where it is indexed and made available to users through search interfaces, chat, and APIs.

Key concepts

These are the main building blocks you will work with in Turing ES. You do not need to understand all of them before getting started — come back to each one as you need it.

CONCEPT	WHAT IT IS	LEARN MORE
Semantic Navigation Site	The central configuration object. Defines what content is indexed, how it is searched, and how results are presented.	Core Concepts
Connector	A component in Dumont DEP that extracts content from a source and sends it to Turing ES.	Core Concepts
Facets	Filterable attributes shown alongside search results (e.g., category, date, author).	Core Concepts
Spotlight	Curated results pinned to specific search terms.	Semantic Navigation
Targeting Rules	Rules that show different results to different users based on their profile.	Semantic Navigation
Merge Providers	Rules that combine documents from two different connectors into one enriched result.	Semantic Navigation
RAG	Retrieval-Augmented Generation — finding relevant documents and using them to ground an LLM's response.	GenAI & LLM
AI Agent	A named assistant that combines an LLM with a set of tools and knowledge sources.	GenAI & LLM
Assets	A file manager backed by MinIO where you upload documents to feed the Knowledge Base — files are automatically indexed as vector embeddings.	Assets
Chat	The conversational AI interface — three tabs for direct LLM chat, Semantic Navigation search, and each configured AI Agent.	Chat
Token Usage	A dashboard showing LLM token consumption by model, day, and month — useful for monitoring AI costs.	Token Usage

Where to go next

Not sure where to start? Here is a suggested path depending on what you want to do:

I want to understand how Turing ES works → Read Core Concepts first, then Architecture Overview.

I want to set up Turing ES → Go to the Installation Guide.

I want to configure search for my content → Start with Core Concepts then go to Semantic Navigation.

I want to add generative AI to my search → Read GenAI & LLM Configuration.

I want to use the AI chat interface → Go to Chat for direct LLM, Semantic Navigation, and AI Agent tabs.

I want to upload documents to the Knowledge Base → Go to Assets to manage files and train the RAG index.

I want to monitor LLM usage and costs → Go to Token Usage.

I want to secure Turing ES with SSO → Go directly to Security & Keycloak.

Core Concepts

This page explains the fundamental concepts of Turing ES in plain terms. No configuration files, no code — just the mental model you need before diving into the technical documentation.

Semantic Navigation Site

The **Semantic Navigation Site** (or SN Site) is the central object in Turing ES. Everything revolves around it.

Think of an SN Site as a configured search experience. It defines:

- **What content is indexed** — which fields each document has (title, text, URL, date, image, and custom fields)
- **How content is searched** — which fields are searchable, how results are ranked, how many results per page
- **How results are navigated** — which fields become filterable facets (e.g., category, author, date range)
- **How results are presented** — field mappings for display, highlighting of matched terms
- **Advanced behaviors** — Spotlights for curated results, Targeting Rules for personalization, Merge Providers for combining content from multiple sources
- **AI capabilities** — whether GenAI and RAG are enabled, and which language model to use

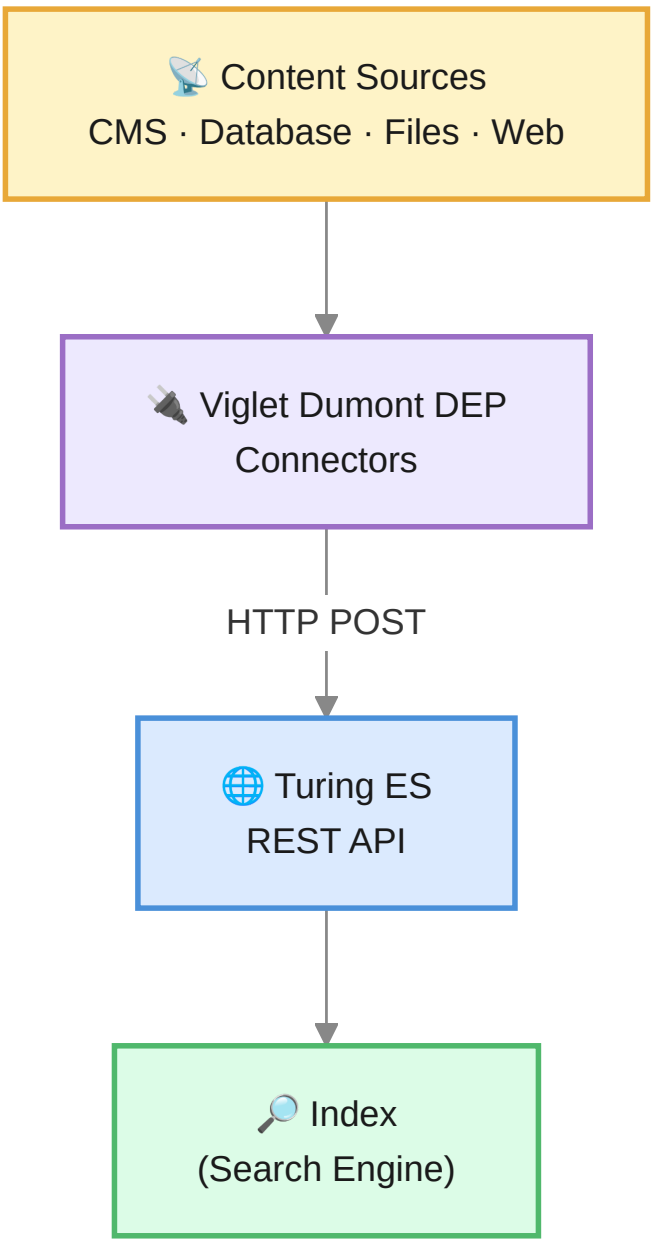
A single Turing ES instance can host multiple SN Sites, each independently configured. For example, you could have one site for your public website, another for your internal knowledge base, and a third for your product documentation — all on the same server.

Content Ingestion

Content does not flow into Turing ES on its own. It needs connectors.

Viglet Dumont DEP

Viglet Dumont DEP is a separate application that manages connectors — the components responsible for extracting content from its original source and sending it to Turing ES. Dumont DEP and Turing ES are separate projects that work together: Dumont handles *getting* content, Turing handles *indexing and searching* it.



How a connector works

Each connector in Dumont DEP:

- Connects to a content source (a website, a database table, a file folder, a CMS)
- Extracts documents according to its configuration (which pages to crawl, which SQL query to run, which folder to scan)
- Sends each document to the Turing ES REST API, targeting a specific SN Site

Turing ES receives the document, validates it against the SN Site configuration, and creates an indexing job. The job processes the document and writes it to the configured search engine (Apache Solr, Elasticsearch, or Apache Lucene) — making it immediately searchable. Apache Solr is the primary and

default backend, but Turing ES also supports Elasticsearch and Lucene as alternative search engine backends.

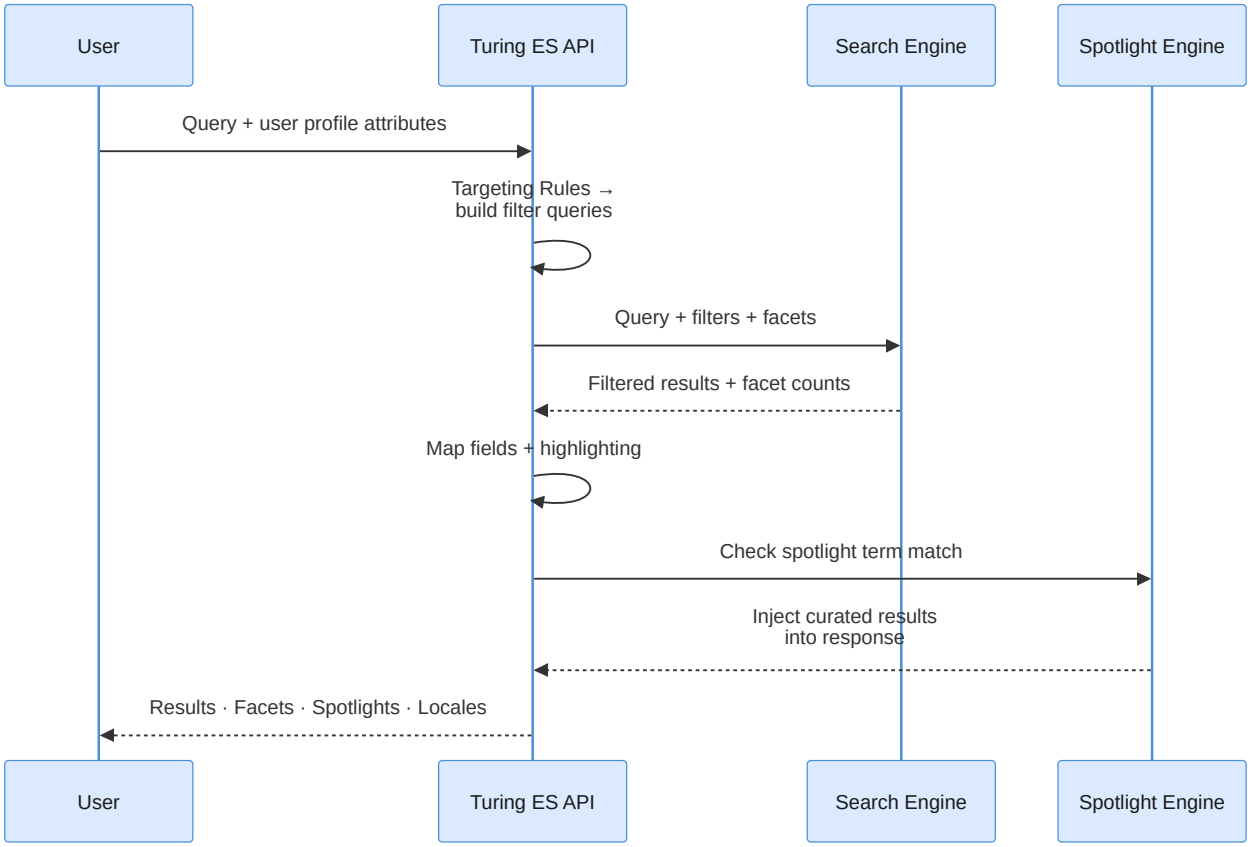
What happens when two connectors index the same content

Sometimes the same real-world document exists in two systems with complementary information. For example, your CMS has structured metadata (author, tags, content type) while your web crawler has the full rendered text of the same page. Neither connector alone gives you a complete document.

Merge Providers solve this by instructing Turing ES to detect when two connectors have indexed the same document — using a shared field as a join key — and merge them into one enriched result. See Semantic Navigation to learn how to configure this.

Search

When a user searches on an SN Site, this is what happens — in simple terms:



The query arrives at the Turing ES API along with the user's profile attributes (if Targeting Rules are configured)

Targeting Rules run first — they translate the user's profile attributes into additional filter queries (e.g., Solr `fq` parameters when using the default Solr backend), so only content relevant to that user is retrieved

Turing ES executes the search engine query with those filters, facet counts, and result ranking applied together

Results come back from the search engine already filtered; Turing ES maps fields and applies highlighting

Spotlights are checked last — if the query matches a spotlight term, curated documents are injected at their configured positions in the response

The final response — results, facets, spotlights, locales, spell check, and similar documents — goes back to the client

Facets

Facets are filterable dimensions shown alongside search results. A user searching for "annual report" might see facets for *Year*, *Department*, and *Document Type*, and click to narrow results to "Finance" documents from "2024".

Facets are configured per SN Site, at the field level. Any indexed field can become a facet — the configuration defines how it is labeled, sorted, and counted.

Spotlights

A Spotlight is a curated result pinned to a specific search term. When someone searches for "benefits", you can ensure your HR benefits page always appears at the top — regardless of how it ranks organically.

Spotlights are configured per SN Site and managed through the admin console. See Spotlights for details.

Targeting Rules

Targeting Rules let you show different results to different users from the same index. A document tagged as `audience: internal` only appears for users whose profile includes that attribute. Users without a matching profile always see untagged content.

See Targeting Rules for a full explanation.

Generative AI

Turing ES integrates generative AI in two places: directly on **SN Sites** (search + summarization) and through standalone **AI Agents** (open-ended chat with tools).

RAG on an SN Site

When GenAI is enabled for an SN Site, indexed documents are also stored as **vector embeddings** — a mathematical representation of their meaning, not just their words. When a user asks a question, Turing ES:

- Finds the most semantically relevant documents (not just keyword matches) using the vector embeddings
- Passes those documents as context to a language model
- Returns a natural-language answer grounded in your actual content

This is called **Retrieval-Augmented Generation (RAG)**. The LLM does not make things up — it answers based on what is in your index.

Knowledge Base

Beyond SN Sites, Turing ES provides a **Knowledge Base** — a file and folder interface in the admin console (backed by MinIO) where you can upload documents directly. These files are also indexed as vector embeddings and become available to AI Agents as a searchable knowledge source, independent of any connector or SN Site.

The Knowledge Base is managed through the **Assets** page: drag-and-drop upload, folder navigation, inline preview, and batch AI training with real-time progress. Uploaded files are indexed automatically on upload and unindexed on deletion.

AI Agents

An **AI Agent** is a named assistant that you compose from three ingredients:

- A **language model** (Anthropic Claude, OpenAI, Gemini, Ollama, or others)
- A set of **tools** it can call (search your SN Sites, query the Knowledge Base, browse the web, run Python code, get financial data, and more)
- Optionally, one or more **MCP Servers** — external services that provide additional tools via the Model Context Protocol

Each AI Agent appears as its own tab in the **Chat** interface. You can have a "Research Assistant" that combines SN Site search with web browsing, a "Data Analyst" that can run Python code and query your knowledge base, and a "Support Agent" that only sees your product documentation — all on the same platform.

See AI Agents for configuration, Tool Calling for the full tool reference, and MCP Servers for connecting external tools.

The Admin Console

Everything described above — SN Sites, connectors, Spotlights, Targeting Rules, Merge Providers, AI Agents, MCP Servers, and the Knowledge Base — is configured and managed through the **Turing ES Admin Console**: a browser-based React application served by Turing ES itself.

The admin console also exposes application logs (when MongoDB is configured), search metrics, and top search terms, giving you operational visibility without needing access to the server.

Consuming Turing ES from Your Application

Turing ES exposes its search, chat, and indexing capabilities through four integration options. Choose the one that best fits your technology stack.

REST API

The primary integration method. All search, indexing, and administration operations are available as HTTP endpoints. The search response is a self-describing JSON object with pre-built navigation links — no URL construction needed on the client side.

```
GET https://<your-host>/api/sn/{siteName}/search?q=your+query
```

Suitable for any language or platform that can make HTTP requests.

GraphQL

Turing ES also exposes a GraphQL endpoint for clients that prefer a graph-based query model. Useful when you want to request only specific fields or combine multiple queries in a single request.

```
POST https://<your-host>/graphql
```

Java SDK

An official Java client library is available on **Maven Central**. It provides a typed API for performing searches, submitting documents for indexing, and interacting with SN Sites — without dealing with raw HTTP or JSON.

```
<dependency>
    <groupId>com.viglet.turing</groupId>
    <artifactId>turing-java-sdk</artifactId>
    <version>${turing.version}</version>
</dependency>
```

JavaScript / TypeScript SDK

An official JavaScript SDK is available on **npm**. It is TypeScript-ready and covers the same search and indexing operations as the Java SDK, suited for web applications and Node.js backends.

```
npm install @viglet/turing-sdk
```

Ready to go deeper?

I WANT TO...	GO TO
Understand the full system architecture	Architecture Overview
Configure Spotlights, Targeting Rules, or Merge Providers	Semantic Navigation
Set up GenAI and RAG	GenAI & LLM Configuration
Configure AI Agents and tools	AI Agents, Tool Calling
Use the AI chat interface	Chat
Upload documents to the Knowledge Base	Assets
Monitor LLM token consumption	Token Usage
Authenticate via API Key	Authentication
Secure Turing ES with Keycloak	Security & Keycloak
Install Turing ES	Installation Guide

Turing ES — Architecture Overview

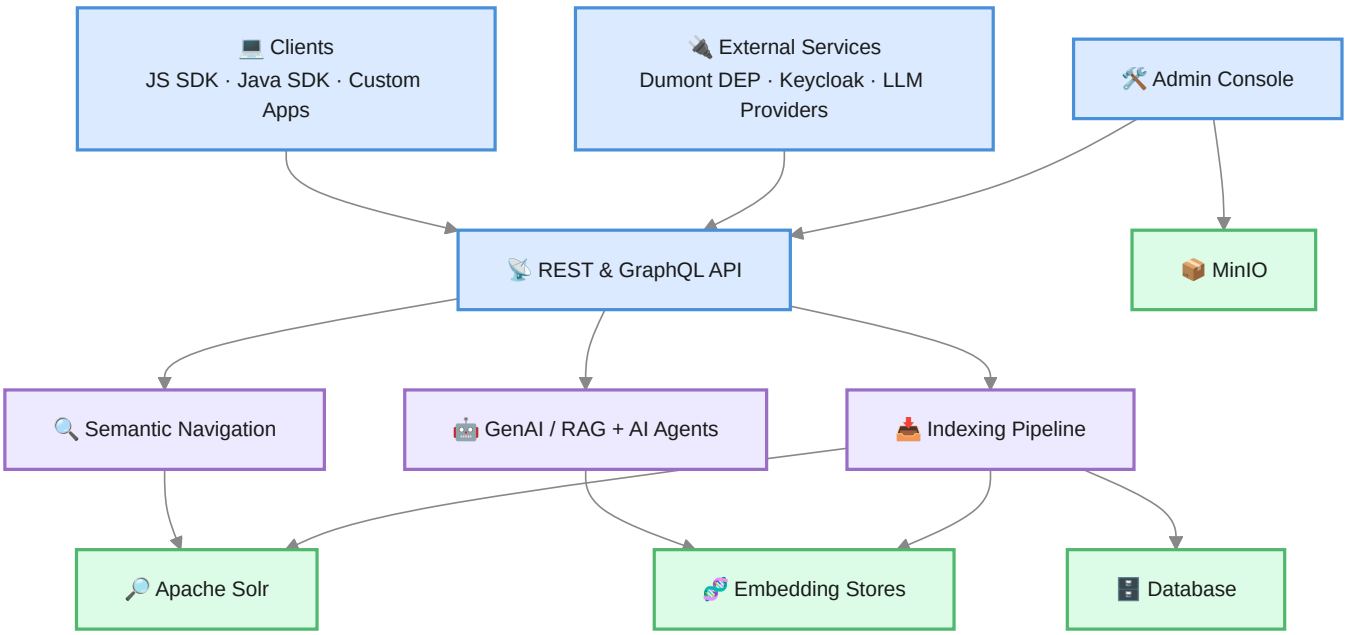
Introduction

Viglet Turing ES is an open-source enterprise search platform that combines semantic navigation, generative AI, tool calling, and AI Agents. It allows organizations to index content from multiple sources, apply Retrieval-Augmented Generation (RAG) over that content, and expose rich search experiences through REST and GraphQL APIs.

Content ingestion is handled by **Viglet Dumont DEP**, a separate project that runs connectors independently and delivers indexed documents to Turing ES via an asynchronous message queue.

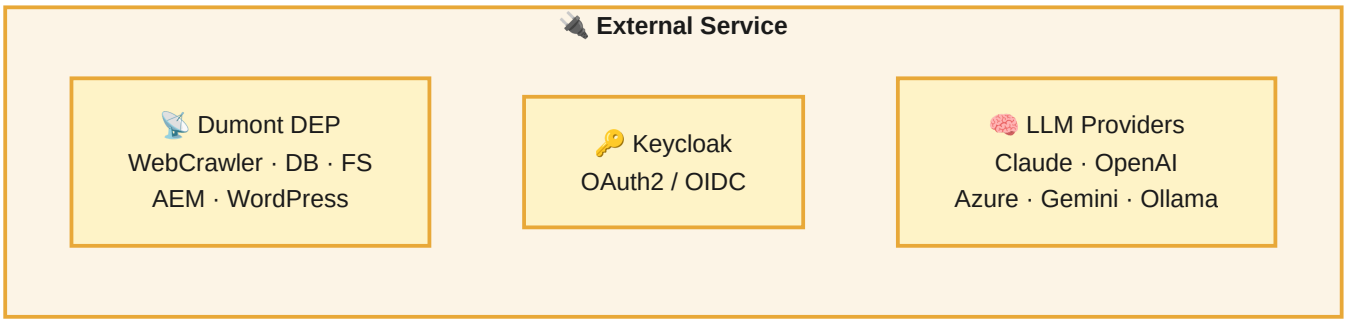
This document describes the system's components, internal modules, and the two core data flows: **indexing** and **search**.

High-Level Component Diagram



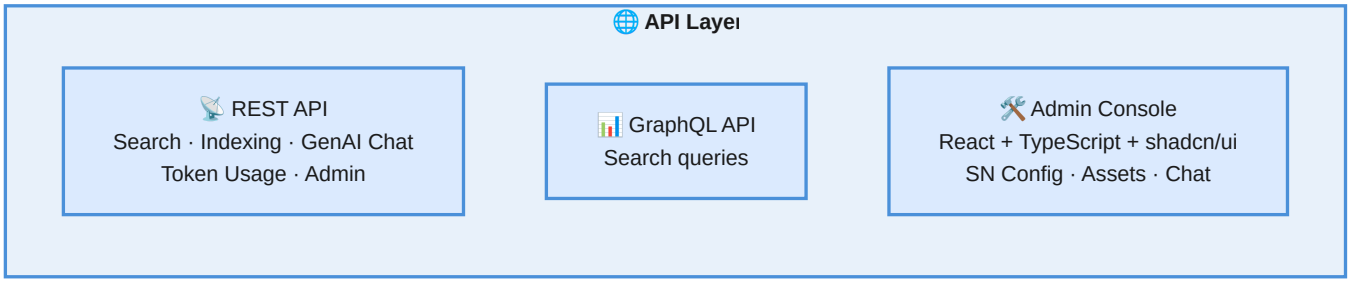
The architecture is organized into **four layers**: Clients & External Services at the top, an API gateway in the middle, Core Modules for business logic, and Backends & Storage at the bottom. Each layer is detailed below.

Clients & External Services



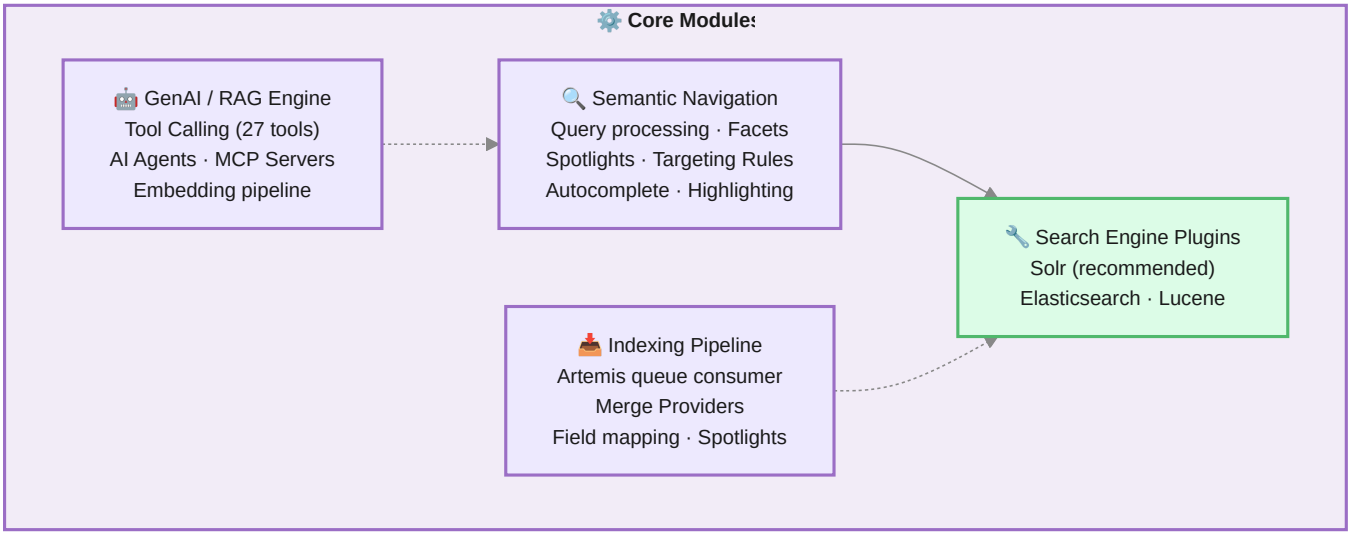
COMPONENT	DESCRIPTION
JS SDK / Java SDK	Typed clients for search and indexing — available on npm and Maven Central
Dumont DEP	Separate application that runs connectors and sends documents to Turing ES via REST API
Keycloak	Optional OAuth2 / OIDC provider for production SSO
LLM Providers	Six supported vendors: Anthropic Claude, OpenAI, Azure OpenAI, Google Gemini, Gemini (OpenAI-compatible), Ollama

API Layer



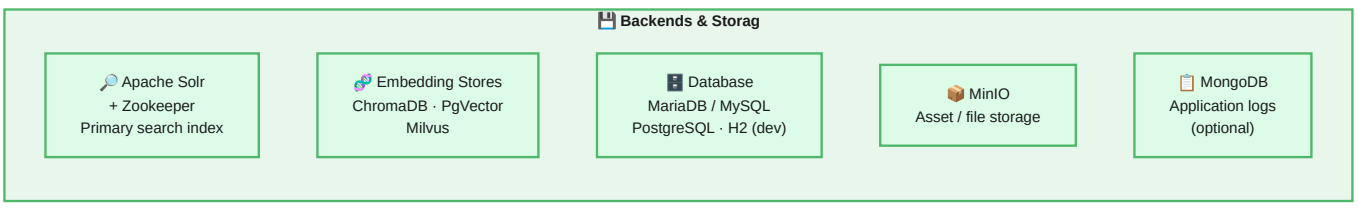
COMPONENT	PACKAGE	DESCRIPTION
REST API	`api`	Controllers for search, indexing, GenAI chat, agents, token usage, and administration
GraphQL API	`api`	Search query resolvers as an alternative to REST
Admin Console	`turing-react`	SN Site configuration, Assets file manager (MinIO), Chat interface, Token Usage dashboard
Security	`spring/security`	Session-based auth (console) + API Key header (REST) + optional Keycloak OAuth2/OIDC

Core Modules



MODULE	PACKAGE	RESPONSIBILITY
Semantic Navigation	`sn`	Core search orchestration: query processing, facets, spotlights, targeting rules, autocomplete
Search Engine Plugins	`se` / `plugins/se`	Abstraction layer over Solr (recommended), Elasticsearch, and Lucene
GenAI / RAG	`genai`	RAG over SN content and MinIO assets; LLM context building and invocation
Tool Calling	`genai/tool`	27 native tools: SN search (15), RAG/KB (4), web crawler (2), finance (2), weather (1), image search (1), datetime (1), code interpreter (1); plus MCP servers
LLM Providers	`genai/provider/llm`	Pluggable provider factory: Anthropic, OpenAI, Azure OpenAI, Gemini, Gemini-OpenAI, Ollama
AI Agents	`agent`	Composition of LLM Instance + Tools + MCP Servers into deployable assistants
Indexing Pipeline	`indexer`	Receives documents via Artemis, applies Merge Providers, writes to Solr and embedding stores
Message Queue	`artemis`	Apache Artemis — async communication between Dumont DEP and the indexing pipeline
OCR	`ocr`	Text extraction from PDFs, Word documents, and images via Apache Tika
Persistence	`persistence`	JPA entities, repositories, DTOs, and MapStruct mappers for all domain objects

Backends & Storage



BACKEND	PURPOSE	NOTES
Apache Solr	Primary search index	SolrCloud mode with Zookeeper in production
Embedding Stores	Vector storage for RAG	One active per deployment — details
Database	Configuration, metadata, spotlights	H2 for dev; MariaDB/MySQL for production
MinIO	Asset/file object storage	Managed via admin console Assets file manager
MongoDB	Application log persistence	Optional — custom Logback appender, browsable in admin console

Indexing Flow

Content ingestion is handled externally by **Viglet Dumont DEP**. Each connector runs as an independent process and sends documents to Turing ES via its **REST API**. The API receives the request, validates it against the target Semantic Navigation Site configuration, and creates an indexing job that is queued internally via Apache Artemis for asynchronous processing.

The **Semantic Navigation Site** is the central configuration artifact that drives the entire indexing behavior: it defines which Solr instance to use, which fields the documents carry, how those fields are mapped and used (title, text, URL, date, image, facets, etc.), how search will behave, and which spotlights are configured. The indexing pipeline reads this configuration to know exactly what to do with each incoming document.

Key indexing concepts

REST API as entry point: Dumont DEP connectors send documents to Turing ES via REST API, not by writing directly to Solr or the queue. The API is the single integration point — it validates the request, loads the target SN Site configuration, and enqueues an indexing job via Apache Artemis for asynchronous processing.

SN Site as the indexing blueprint: Every indexing job is bound to a Semantic Navigation Site. The site configuration defines the complete indexing contract: which Solr collection to write to, which document fields exist and how they are typed, which fields become facets, how highlighting and autocomplete will work, and which spotlights are active. The pipeline reads this configuration at job execution time to determine field mappings, facet assignments, and spotlight handling.

Spotlight persistence: When an incoming document matches a configured Spotlight — for example, its URL or ID matches a spotlight term — the pipeline indexes the document in Solr normally and also persists the spotlight content (title, description, URL, position) in the relational database. This ensures the spotlight data remains available for injection even if the document is later removed from the Solr index.

Viglet Dumont DEP: The connector system that feeds Turing ES. It runs as a separate application and manages its own connector lifecycle (schedules, credentials, field mappings). Connectors currently available in Dumont DEP include WebCrawler (Nutch-based), Database, FileSystem, AEM, and WordPress. Refer to the Dumont DEP documentation for connector configuration.

Merge Providers: When two Dumont DEP connectors independently index different representations of the same real-world document — for example, AEM indexing structured metadata from `model.json` and WebCrawler indexing the rendered HTML of the same page — the Merge Provider identifies them as the

same document using a configured join key and merges their fields before writing to Solr. See Semantic Navigation for a detailed explanation.

Embedding stores: If Generative AI is enabled for an SN Site, a vector embedding is generated for each indexed document and written to the configured embedding store. Turing ES supports three embedding backends via Spring AI: **ChromaDB**, **PgVector** (PostgreSQL extension), and **Milvus**. Only one is active per deployment. The default embedding store and embedding model are defined globally in **Administration → Settings**.

MinIO asset indexing: Turing ES includes an **Assets** file manager in the admin console, backed by MinIO as the object storage layer. Files are uploaded via drag-and-drop, organized into folders, and automatically indexed as vector embeddings on upload (and unindexed on deletion). A batch "Train AI with Assets" operation processes all files using Apache Tika for text extraction, chunking at 1,024 characters, and storing embeddings in the active vector store.

Application logs in MongoDB: When MongoDB is configured, Turing ES ships with a custom Logback appender that extends `ch.qos.logback`. Every log entry generated by the application — including indexing events, search requests, errors, and system events — is persisted to MongoDB in addition to standard output. These logs are exposed in the admin console, giving administrators full visibility into application behavior without requiring access to the server file system or a separate log management tool.

Search Flow

The search flow is synchronous and request-driven. Every request goes through a structured pipeline inside `TurSNSearchProcess` before a response is returned to the client.

Key search concepts

Site-scoped execution: Every search request is scoped to a named Semantic Navigation Site. The site configuration defines which search engine backend to use, how many results per page, facet definitions, field mappings, whether Spotlights and Targeting Rules are active, and whether GenAI is enabled.

Plugin abstraction: The orchestrator does not query the search backend directly. An intermediate plugin layer translates the abstract search context into backend-specific queries, supporting **Apache Solr**, **Elasticsearch**, and **Lucene** as backends. Solr is the recommended production backend as it provides the most complete feature set, including full support for facets, spotlights, targeting rules, highlighting, and autocomplete. Elasticsearch and Lucene are available as alternatives with a reduced feature set.

Spotlight injection: After retrieving organic results, the system checks whether any configured Spotlights match the current query. Matching Spotlights insert curated documents at specific positions in the result list. See Semantic Navigation for details.

Targeting Rules: Results are filtered based on the requesting user's profile attributes, passed in the request context. Targeting Rules translate those attributes into additional Solr filter queries. See Semantic Navigation for details.

Field mapping and highlighting: Before returning results, each document's fields are remapped to a canonical set of display fields configured per site (title, description, text, date, image, URL). Highlighting wraps matched terms with configurable HTML tags (default: `<mark>`).

Metrics: After assembling the response, query metrics (search term, result count, site, timestamp) are logged asynchronously to avoid adding latency to the response.

Technology Stack

LAYER	TECHNOLOGY	NOTES
Runtime	Java 21	Minimum supported version
Framework	Spring Boot + Spring AI	Application container; Spring AI powers LLM and embedding integrations
Search Engine	Apache Solr (recommended), Elasticsearch, Lucene	Solr is the primary production backend with the most complete feature set; Elasticsearch and Lucene are supported as alternatives
Solr Coordination	Apache Zookeeper	Required for Solr in production (SolrCloud mode)
Message Broker	Apache Artemis	Asynchronous indexing queue (embedded in Turing ES)
Database	H2 / MariaDB / MySQL / PostgreSQL	H2 for development; MariaDB or MySQL recommended for production
Embedding Stores	ChromaDB / PgVector / Milvus	Via Spring AI; one backend active per deployment
Asset Store	MinIO	External service; configured in `application.yaml` (host, user, password); object storage for files managed via the admin console folder UI
Log Store	MongoDB	Optional; custom Logback appender (`ch.qos.logback`) persists all application logs to MongoDB, accessible via the admin console
LLM Providers	Anthropic Claude, OpenAI, Azure OpenAI, Google Gemini, Gemini (OpenAI-compatible), Ollama	Configured per site or globally; one active at a time

LAYER	TECHNOLOGY	NOTES
Tool Calling	27 native tools across 7 categories + MCP (external servers)	Semantic Nav (15), RAG/KB (4), Web Crawler (2), Finance (2), Weather (1), Image Search (1), DateTime (1), Code Interpreter (1); MCP via HTTP or stdio
Identity Management	Keycloak	OAuth2 / OpenID Connect; optional for deployments without SSO
Load Balancer	Apache HTTP Server	Optional; required for high-availability cluster deployments
Connector System	Viglet Dumont DEP	Separate application; sends documents to Turing ES via REST API, which queues them internally to Artemis
Build System	Apache Maven	Multi-module project
Frontend	React + TypeScript + shadcn/ui + Vite	Admin console (`turing-react`) — includes SN configuration, Assets manager, Chat interface, Token Usage dashboard
Containerization	Docker / Docker Compose	Available in `containers/` directory
Orchestration	Kubernetes	Manifests available in `k8s/` directory
API Protocols	REST + GraphQL	Swagger UI available in development mode
Java SDK	`turing-java-sdk`	Available on Maven Central; typed client for search and indexing
JavaScript SDK	`@viglet/turing-sdk`	Available on npm; TypeScript-ready client for web and Node.js

Deployment Topologies

Turing ES supports several deployment configurations. Each builds on the previous one — start with what you need and add components as requirements grow.

Development

Minimal setup for local development and evaluation.

```
Turing ES (H2 embedded) + Apache Solr
```

Turing ES starts with an embedded H2 database. No external database is needed. Not suitable for production.

Simple Production

Recommended baseline for production environments.

```
Turing ES + Apache Solr + Zookeeper + MariaDB / MySQL
```

Solr runs in SolrCloud mode coordinated by Zookeeper, enabling index replication and fault tolerance at the Solr layer. MariaDB or MySQL provides durable persistence for Turing ES configuration and metadata.

Production with Security (SSO)

For environments that require integration with an identity provider or corporate SSO.

```
Turing ES + Apache Solr + Zookeeper + MariaDB / MySQL + Keycloak
```

Keycloak handles authentication via OAuth2 / OpenID Connect. Users log in through Keycloak and receive tokens that are validated by Turing ES. See Security & Keycloak for configuration details.

Production with Log UI

For environments where administrators need visibility into application behavior directly from the Turing ES admin console — without access to server logs or external log tools.

```
Turing ES + Apache Solr + Zookeeper + MariaDB / MySQL + MongoDB
```

When MongoDB is configured, a custom Logback appender persists every log entry generated by the application to MongoDB. The admin console exposes these logs in a searchable interface, showing errors, warnings, indexing events, and system activity in real time.

High Availability

For environments requiring horizontal scaling and zero-downtime deployments.

```
Apache HTTP Server (load balancer)
    └── Turing ES node 1
    └── Turing ES node 2
    └── Turing ES node N
Apache Solr + Zookeeper (cluster)
MariaDB / MySQL (primary + replica)
Keycloak (optional)
MongoDB (optional)
```

Multiple Turing ES instances run behind Apache HTTP Server configured as a reverse proxy and load balancer. Solr runs as a multi-node SolrCloud cluster. The database should be configured with at least one replica for redundancy.

Related Pages

PAGE	DESCRIPTION
Installation Guide	Setup with Docker, JAR, or build from source
Configuration Reference	All application.yaml properties
Integration	Manage content connectors in the admin console
Dumont DEP — Architecture	Connector-side architecture — pipeline engine, message queue, and indexing plugins
Dumont DEP — Connectors	Available connectors and deployment types

Viglet Turing ES: Installation Guide

Viglet Turing ES (https://viglet.com/turing) is an open source enterprise search platform (https://github.com/openviglet) with Semantic Navigation and Generative AI as its main features. All content is indexed in Apache Solr as the primary search engine.

Installing Java

Turing ES requires Java.

> **NOTE**
> Turing ES only supports Java 21.

During installation, Turing ES reads the JAVA_HOME and PATH variables to install the necessary services. Therefore, you must modify the variables as specified here:

On Windows:

Set the JAVA_HOME system variable to point to the JAVA location.

Attach the PATH system variable with `%JAVA_HOME%/bin`.

On Unix:

Create an environment variable called JAVA_HOME.

Provide the JAVA bin path in the PATH variable.

Turing ES Requirements

The information below describes the resource modes available to run Turing ES.

> **NOTE**
> Use a screen resolution of at least 1024 x 768 for optimum clarity.

This section describes the minimum and suggested features for systems on which you can configure Turing ES.

Hardware Requirements

MINIMUM	RECOMMENDED
Single 1.5 GHz processor, 1 GB available memory	Dual 2 GHz processors with Hyper-Threading enabled, 2 GB available memory

In addition, the following is required:

1GB of disk space for software.

A compatible version of a database.

Installing and Configuring Databases

Turing ES supports multiple database platforms, each with specific unique requirements to maximize effectiveness with Turing ES. The following sections describe these requirements.

Turing ES Database Table Requirements

Turing ES uses multiple tables in multiple databases (also called database schemas by some vendors such as Oracle). You must create new databases / schemas that will contain the sets of tables described below.

Oracle Database Requirements

This section describes configuration requirements for an existing database.

To set up an Oracle database:

Install and configure a single database instance for use by Turing ES.

Create a database schema to contain the Turing ES system tables.

Configure the Turing ES admin user for the database. For Oracle, the database user with the grant of the CONNECT and RESOURCE roles. Make sure the user has UNLIMITED TABLESPACE and CREATE VIEW privileges.

Microsoft SQL Server Database Requirements

This section discusses the steps required to configure Microsoft SQL Server for your Turing ES databases.

When creating the Turing ES database, use the following options:

You must install the Database Engine feature and Management Tools are recommended. All other features are optional.

Set the Collation configuration variable to `SQL_Latin1_General_CP1_CS_AS` or `Latin1_General / Case-sensitive / Accent-Sensitive`.

Set authentication mode to mixed mode.

Turing ES system user cannot be sa.

Configuring the SQL Server Database and Schemas

> **NOTE**
>
> The database and usernames provided are examples only.

To create the SQL Server database:

Select Databases > New Database.

Enter the database name (eg turing) and the size of the data and log files according to the environment.

Go to the Options page and set the following values:

Collation = `Latin1_General_CS_AS` or `SQL_Latin1_General_CP1_CS_AS`

Miscellaneous > Allow Snapshot Isolation = True

Miscellaneous > Is Read Committed Snapshot On = True

```
USE master
GO
ALTER DATABASE turing
SET SINGLE_USER
WITH ROLLBACK IMMEDIATE
GO
ALTER DATABASE turing
SET ALLOW_SNAPSHOT_ISOLATION ON
GO
ALTER DATABASE turing
SET READ_COMMITTED_SNAPSHOT ON
GO
ALTER DATABASE turing
SET MULTI_USER
GO
```

Under Security > Logins, select New Login and set the following attributes:

Enter the Login name (for example, turing)

Deselect the Enforce Password Expiration option.

Set the Default database.

On the User Mapping page, set the following attributes for the turing database:

Default user = turing

Default schema = turing

With the turing database is selected, select the option for db_owner

Click OK.

PostgreSQL Database Requirements

This section discusses the steps required to configure PostgreSQL for your Turing ES databases:

Installing and configuring the PostgreSQL Database

Configuring the Turing ES user, database, and schema

Installing and configuring the PostgreSQL Database

This section describes the steps required to install and configure a PostgreSQL database for your Turing ES environment. You will also need to configure the Turing user, database and schemas.

To install the PostgreSQL database:

Navigate to the following URL and follow the instructions provided for your supported version of PostgreSQL: https://www.postgresql.org/docs/manuals/

To configure the PostgreSQL instance:

Open the `postgresql.conf` file (located in `<PostgreSQL_installDir>/<version>/data/`) and set the following parameters:

```
max_connections = 10
max_prepared_transactions = 10
shared_buffers = 512MB
autovacuum = off
```

IMPORTANT

The autovacuum daemon is used to reclaim memory/storage space the database is no longer using. You must re-enable this utility after you have finished installing and configuring Turing ES.

> **NOTE**
>
> Configuration values will vary for higher transaction environments and numbers of users. `max_prepared_transactions` should be at least as large as `max_connections` so that every session can have a prepared transaction pending.

Configuring the Turing user, database, and schemas

> **NOTE**
>
> You must be a PostgreSQL super-user or have createuser privileges to create Turing users. To create and configure the Turing ES database, you must be a super-user or have create privileges.

> **IMPORTANT**
>
> When you create a database table in PostgreSQL, the field name must be lowercase.

To configure the Turing ES user:

Create a user as the owner of the Turing ES database:

```
CREATE USER turing PASSWORD <password> with createuser;
```

Create a Turing ES database:

```
CREATE DATABASE turing_db;
ALTER DATABASE turing_db OWNER TO turing;
```

To create and access the Turing ES user and schema:

Log in to the turing (database) using the turing account:

```
$psql -U turing -d turing_db
```

Create the Turing schema and grant ownership to the Turing user:

```
CREATE SCHEMA turing;
ALTER SCHEMA turing OWNER TO turing;
```

Set the PostgreSQL search_path variable:

```
ALTER USER turing SET SEARCH_PATH = turing;
```

Turing Utils

First you need install Turing Utils, that contains sample configurations and script to facilitate the use of Turing ES, so go to https://viglet.org/turing/download/ and click on "Integration > Utils" link to download it.

Extract the turing-utils.zip to `/appl/viglet/turing/utils`:

```
mkdir -p /appl/viglet/turing/utils
unzip turing-utils.zip -d /appl/viglet/turing/utils
```

Keycloak

For production deployments that require SSO, Keycloak integrates with Turing ES via OAuth2 / OpenID Connect. Full setup instructions — including database creation, keystore generation, Keycloak configuration, realm and client setup, JVM properties, and Apache reverse proxy configuration — are covered in the Security & Keycloak guide.

Solr Configuration

You need to configure Solr to be able to store Semantic Navigation data.

With Solr running, go to Turing Utils and create the Solr collection that will be used by Turing ES:

```
cd <SOLR_DIR>/bin
./solr create_collection -c turing -n turing -d
/appl/viglet/turing/utils/solr/<VERSION>/turing/<LANGUAGE>
```

Start the Solr service after the collection is created.

Installing Turing ES

Turing ES Download

Option 1 — Docker (fastest)

```
docker pull openviglet/turing:2026.1
docker run -p 2700:2700 openviglet/turing:2026.1
```

Option 2 — JAR download

Go to https://viglet.org/turing/download/ and click on "Download Turing ES" button to download `viglet-turing.jar`.

Copy the `viglet-turing.jar` to `/appl/viglet/turing/server`:

```
mkdir -p /appl/viglet/turing/server
cp viglet-turing.jar /appl/viglet/turing/server
```

Option 3 — Build from source

```
git clone https://github.com/openviglet/turing.git
cd turing
mvn clean package -pl turing-app
cp turing-app/target/viglet-turing.jar /appl/viglet/turing/server/
```

Database

By default, Turing uses H2 as its embedded database, but if you prefer another database you can create an external properties file and reference it at startup using `--spring.config.additional-location`.

Create `/appl/viglet/turing/server/viglet-turing.properties`:

MySQL

```
spring.datasource.url=jdbc:mariadb://localhost:3306/turing
spring.datasource.username=turing
spring.datasource.password=turing
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDial
ect
```

Oracle Database

```
spring.datasource.url=jdbc:oracle:thin:@localhost:1521/turing
spring.datasource.username=turing
spring.datasource.password=turing
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
```

PostgreSQL

```
spring.datasource.url=jdbc:postgresql://localhost:5432/turing
spring.datasource.username=turing
spring.datasource.password=turing
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL94Dial
ect
```

Then start Turing ES referencing the properties file:

```
java -Xmx1g -Xms1g -jar viglet-turing.jar \
  --spring.config.additional-location=file:/appl/viglet/turing/server/viglet-
turing.properties
```

Creating Turing ES Service on Linux

As root, create a `/etc/systemd/system/turing.service` file with the following lines:

```
[Unit]
Description=Viglet Turing ES
After=syslog.target

[Service]
User=turing
Group=turing
WorkingDirectory=/appl/viglet/turing/server
ExecStart=/appl/java/jdk21/bin/java -Xmx1g -Xms1g \
  -jar /appl/viglet/turing/server/viglet-turing.jar \
  --spring.config.additional-location=file:/appl/viglet/turing/server/viglet-turing.properties
SuccessExitStatus=143

[Install]
WantedBy=multi-user.target
```

SPRING BOOT 4

Starting with Spring Boot 4, JAR files are no longer directly executable (`./viglet-turing.jar`). You must launch Turing ES using `java -jar` explicitly. The `.conf` file pattern used in older versions is no longer supported — use `--spring.config.additional-location` to load external properties instead.

Enable and start the service:

```
systemctl daemon-reload
systemctl enable turing.service
systemctl start turing.service
```

Other useful commands:

```
systemctl stop turing.service
systemctl restart turing.service
systemctl status turing.service
```

Starting Turing ES

When Turing ES is started for the first time, it will perform initial setup.

```
$ java -jar viglet-turing.jar
  _____              _                ___  ___
 |_   _|_  _  _ _ (_) _ _   __ _  | __|/ __|
   | | | || || '_|| || ' \ / _` | | _| \__ \
   |_|  \_,_||_|  |_||_||_|\__, | |___||___/ 2026.1.17
                           |___/

:: Copyright (C) 2016-2026 Viglet Turing Enterprise Search

:: Built with Spring Boot :: 3.5.0

First Time Configuration ...
Configuration finished.
```

Accessing the Turing Console

Turing provides remote access to administration, configuration, and management through its Web application interfaces. Once setup is complete, the Console become browser-accessible through the following URL: `http://<host>:<port>/console` where `<host>:<port>` is the listening host and port for the Turing ES. The default port is **2700**.

The default username is **admin**. The password is set via the `TURING_ADMIN_PASSWORD` environment variable before first startup — see the Administration Guide for details.

Appendix A: Installation Modes

Turing ES Server

Simple

Minimal setup using an embedded H2 database. Suitable for local development and evaluation only — not for production.

Prerequisites:

- Linux server
- Java 21
- 50 GB HDD
- 2 GB RAM

Target Audience: Development and testing environments.

Docker Compose

Turing ES and its dependencies installed via Docker Compose, including:

- MariaDB — Turing ES system tables
- Solr — Semantic Navigation search backend
- Nginx — reverse proxy on port 80
- Turing ES

Prerequisites:

- Linux server with Docker and Docker Compose installed
- 50 GB HDD
- 4 GB RAM

Target Audience: Teams that want a complete environment without manual service installation. Suitable for QA and production.

Kubernetes

Turing ES and its dependencies deployed via Kubernetes manifests (available in the `k8s/` directory), including:

MariaDB — Turing ES system tables
Solr — Semantic Navigation search backend
Nginx — reverse proxy
Turing ES

Prerequisites:

Linux server with Kubernetes, or a cloud Kubernetes service (GKE, EKS, AKS)
100 GB storage
4 GB RAM

Target Audience: Cloud deployments requiring horizontal scaling and infrastructure-as-code.

Manual Installation

Each service installed individually following this guide:

MariaDB — Turing ES system tables
Solr + Zookeeper — Semantic Navigation search backend
Apache HTTP Server — reverse proxy (required for Keycloak integration)
Turing ES

Prerequisites:

One to four Linux servers
50–100 GB storage per server
Minimum 2 GB RAM per server

Connectors

Content ingestion is handled by **Viglet Dumont DEP**, a separate application. Available connectors include WebCrawler (Apache Nutch), Database, FileSystem, AEM, and WordPress. Refer to the Dumont DEP documentation for connector setup.

Prerequisites:

A server with access to the content sources to be indexed
50 GB storage

Appendix B: Docker Compose

You can start the Turing ES using MariaDB, Solr and Nginx.

```
./gradlew turing-app:build -x test -i --stacktrace
docker-compose up
```

> **NOTE**
>
> If you have problems with permissions on directories, run `chmod -R 777 volumes`

Docker Commands

Turing

```
docker exec -it turing /bin/bash
```

Solr

```
docker exec -it turing-solr /bin/bash
```

Check logs:

```
docker-compose exec turing-solr cat /opt/solr/server/logs/solr.log
# or
docker-compose exec turing-solr tail -f /opt/solr/server/logs/solr.log
```

MariaDB

```
docker exec -it turing-mariadb /bin/bash
```

Nginx

```
docker exec -it turing-nginx /bin/bash
```

Configuration Reference

This page documents every significant property in the Turing ES `application.yaml` file. The file uses standard Spring Boot configuration — any property can be overridden via environment variables, JVM system properties (`-Dkey=value`), or a separate `application-production.yaml` file in the working directory.

OVERRIDE PATTERN

To override a property at runtime without editing the file, use the environment variable convention: replace `.` and `-` with `_` and uppercase everything. For example, `turing.minio.enabled` becomes `TURING_MINIO_ENABLED=true`.

Full Default Configuration

```yaml
spring:
  datasource.hikari:
    maximum-pool-size: 20
    minimum-idle: 5
    idle-timeout: 600000
    pool-name: TurConnectionPool
  jmx.enabled: true
  profiles:
    active: production
  h2:
    console:
      enabled: false
      path: /h2
      settings:
        web-allow-others: true
  datasource:
    url:
jdbc:h2:file:./store/db/turingDB;DATABASE_TO_UPPER=false;CASE_INSENSITIVE_IDEN
TIFIERS=true
    username: sa
    password: ""
    driver-class-name: org.h2.Driver
  liquibase:
    change-log: classpath:db/changelog/db.changelog-master.yaml
    enabled: true
    default-schema: PUBLIC
  jpa:
    properties:
      hibernate:
        "[globally_quoted_identifiers]": true
        "[format_sql]": false
        "[enable_lazy_load_no_trans]": true
        "[generate_statistics]": false
      "[jakarta.persistence.sharedCache.mode]": ALL
    hibernate:
      ddl-auto: none
    show-sql: false
  jackson:
    mapper:
      "[DEFAULT_VIEW_INCLUSION]": true
  thymeleaf:
    mode: HTML
    check-template: true
    check-template-location: true
    prefix: classpath:public/
    suffix: .html
```

```
    artemis:
      mode: embedded
      broker-url: localhost:61616
      embedded:
        enabled: true
        persistent: true
        data-directory: store/queue
        queues: indexing.queue
      pool:
        max-connections: 10
    jms:
      template:
        default-destination: indexing.queue
    servlet:
      multipart:
        max-file-size: 1024MB
        max-request-size: 1024MB
    mvc:
      async:
        request-timeout: 3600000
    output:
      ansi:
        enabled: always
    graphql:
      graphiql:
        enabled: true
        path: /graphiql
      http.path: /graphql
      cors:
        allowed-origins: ${turing.allowedOrigins}
        allowed-methods: GET,POST
        allowed-headers: "*"

  server:
    port: ${PORT:2700}
    compression:
      enabled: true
      mime-types: application/json,text/css,application/javascript
      min-response-size: 2048
    tomcat:
      accesslog:
        enabled: false
        suffix: .log
        prefix: access_log
        file-date-format: .yyyy-MM-dd
        directory: logs
      basedir: store
      threads:
        max: 600
      max-swallow-size: 200MB
```

```yaml
    max-http-form-post-size: 200MB
    maxHttpResponseHeaderSize: 800KB

turing:
  ai.crypto.key: sample-key-for-crypto
  allowedOrigins: http://localhost:4200,http://localhost:5173
  multi-tenant: false
  keycloak: false
  url: http://localhost:2700
  open-browser: true
  jms.concurrency: 1-1
  code-interpreter:
    python-executable: ""
  solr:
    timeout: 30000
    cloud: false
    commit:
      within: 10000
      enabled: false
  elasticsearch:
    timeout: 30000
  search:
    metrics.enabled: true
    cache:
      ttl:
        seconds: 86400000
      enabled: false
  minio:
    enabled: false
    endpoint: http://localhost:9000
    access-key: admin
    secret-key: minha_senha_forte
    bucket: turing-assets
  mongodb:
    enabled: false
    uri: mongodb://localhost:27017
    logging:
      database: turingLog
      collection:
        server: server
        aem: aem
        indexing: indexing
      purge:
        days: 30

springdoc:
  pathsToMatch: /api/**
  swagger-ui:
    path: /swagger-ui.html
```

```yaml
management:
  endpoints:
    web:
      exposure:
        include: "*"

logging:
  config: classpath:logback-spring.xml
  level:
    org:
      springframework: INFO
      apache: INFO
      "[apache.activemq]": ERROR
    com:
      viglet: INFO
    dev:
      langchain4j: INFO
      ai4j:
        openai4j: INFO
  file:
    name: store/logs/turing.log
  logback:
    rollingpolicy:
      max-file-size: 25MB
      max-history: 10
```

Property Reference

Server

PROPERTY	DEFAULT	DESCRIPTION
`server.port`	`${PORT:2700 0}`	HTTP port. Override with the `PORT` environment variable.
`server.compression.enabled`	`true`	Enables gzip compression for JSON, CSS, and JavaScript responses
`server.compression.min-response-size`	`2048`	Minimum response size in bytes before compression is applied
`server.tomcat.threads.max`	`600`	Maximum number of Tomcat worker threads
`server.tomcat.basedir`	`store`	Base directory for Tomcat internal files
`server.tomcat.max-swallow-size`	`200MB`	Maximum size of request body Tomcat will absorb
`server.tomcat.max-http-form-post-size`	`200MB`	Maximum size of HTML form POST bodies
`server.tomcat.maxHttpResponseHeaderSize`	`800KB`	Maximum size of HTTP response headers
`server.tomcat.accesslog.enabled`	`false`	Enable Tomcat access log file

Database (Spring Datasource)

The default database is **H2** (embedded, file-based) — suitable for development and simple deployments. For production, replace with MariaDB, MySQL, or PostgreSQL.

PROPERTY	DEFAULT	DESCRIPTION
`spring.datasource.url`	`jdbc:h2:file:./store/db/turingDB`	JDBC URL. Change to your production DB URL.
`spring.datasource.username`	`sa`	Database username
`spring.datasource.password`	`""`	Database password
`spring.datasource.driver-class-name`	`org.h2.Driver`	JDBC driver class
`spring.datasource.hikari.maximum-pool-size`	`20`	Maximum connections in the HikariCP pool
`spring.datasource.hikari.minimum-idle`	`5`	Minimum idle connections kept alive
`spring.datasource.hikari.idle-timeout`	`600000`	Time (ms) a connection can remain idle before being removed
`spring.h2.console.enabled`	`false`	Enable H2 web console — **do not enable in production**

Switching to MariaDB / MySQL:

```yaml
spring:
  datasource:
    url: jdbc:mariadb://localhost:3306/turingdb
    username: turing
    password: your_password
    driver-class-name: org.mariadb.jdbc.Driver
```

Message Queue (Apache Artemis)

Apache Artemis runs **embedded inside Turing ES** by default — no separate installation is required. For high-availability or multi-node deployments, you can switch to an external Artemis broker.

PROPERTY	DEFAULT	DESCRIPTION
`spring.artemis.mode`	`embedded`	`embedded` (default) or `native` (external broker)
`spring.artemis.broker-url`	`localhost:61616`	Used when `mode: native` to point to an external broker
`spring.artemis.embedded.enabled`	`true`	Enables the embedded broker
`spring.artemis.embedded.persistent`	`true`	Persist queue messages to disk (survives restarts)
`spring.artemis.embedded.data-directory`	`store/queue`	Directory for persisted queue data
`spring.artemis.embedded.queues`	`indexing.queue`	Queue names to create on startup
`spring.artemis.pool.max-connections`	`10`	Maximum JMS connection pool size
`spring.jms.template.default-destination`	`indexing.queue`	Default JMS destination for indexing jobs
`turing.jms.concurrency`	`1-1`	JMS listener concurrency range (min-max consumers). Increase for higher indexing throughput.

EXTERNAL ARTEMIS FOR SCALABILITY

To use an external Artemis broker (e.g., for multi-node deployments), set `spring.artemis.mode: native` and point `broker-url` to your broker. The embedded broker should be disabled in that case (`spring.artemis.embedded.enabled: false`).

File Uploads

PROPERTY	DEFAULT	DESCRIPTION
`spring.servlet.multipart.max-file-size`	`1024MB`	Maximum size of a single uploaded file
`spring.servlet.multipart.max-request-size`	`1024MB`	Maximum total size of a multipart request

GraphQL

PROPERTY	DEFAULT	DESCRIPTION
`spring.graphql.http.path`	`/graphql`	GraphQL endpoint path
`spring.graphql.graphiql.enabled`	`true`	Enable GraphiQL browser IDE
`spring.graphql.graphiql.path`	`/graphiql`	GraphiQL IDE path
`spring.graphql.cors.allowed-origins`	`${turing.allowedOrigins}`	CORS allowed origins for GraphQL

Async / Timeout

PROPERTY	DEFAULT	DESCRIPTION
`spring.mvc.async.request-timeout`	`3600000`	Maximum time (ms) for async requests — 1 hour. Covers streaming LLM responses.

Turing ES Core

PROPERTY	DEFAULT	DESCRIPTION
`turing.url`	`http://localhost:2700`	Public base URL of this Turing ES instance
`turing.allowedOrigins`	`http://localhost:4200, http://localhost:5173`	CORS allowed origins for the REST API and GraphQL. Add your frontend URL here.
`turing.keycloak`	`false`	Set `true` to enable Keycloak OAuth2/OIDC. See Security & Keycloak.
`turing.multi-tenant`	`false`	Enable multi-tenant mode
`turing.open-browser`	`true`	Automatically open the admin console in the browser on startup
`turing.ai.crypto.key`	`sample-key-for-crypto`	Encryption key for stored AI provider credentials. **Change this in production.**
`turing.code-interpreter.python-executable`	*(auto-detected)*	Absolute path to the Python 3 binary used by the Code Interpreter GenAI tool. When blank, Turing searches standard OS locations automatically.

CHANGE THE CRYPTO KEY

The `turing.ai.crypto.key` is used to encrypt LLM provider API keys stored in the database. Always set a strong, unique value in production.

Solr

PROPERTY	DEFAULT	DESCRIPTION
`turing.solr.timeout`	`30000`	Connection and read timeout (ms) for Solr requests
`turing.solr.cloud`	`false`	Set `true` for SolrCloud mode with Zookeeper
`turing.solr.commit.within`	`10000`	Soft commit time (ms) — Solr commits changes within this window
`turing.solr.commit.enabled`	`false`	Enable explicit commits after indexing. Leave `false` to use Solr's auto-commit.

Elasticsearch

PROPERTY	DEFAULT	DESCRIPTION
`turing.elasticsearch.timeout`	`30000`	Connection and read timeout (ms) for Elasticsearch requests

Search Cache

PROPERTY	DEFAULT	DESCRIPTION
`turing.search.metrics.enabled`	`true`	Record search metrics (query terms, result counts, timestamps)
`turing.search.cache.enabled`	`false`	Enable server-side search result caching
`turing.search.cache.ttl.seconds`	`86400000`	Cache TTL in seconds (default ~1000 days — effectively no expiry when enabled)

MinIO (Assets & Knowledge Base)

MinIO powers the Assets file manager and the RAG Knowledge Base. Disabled by default.

PROPERTY	DEFAULT	DESCRIPTION
`turing.minio.enabled`	`false`	Set `true` to enable MinIO integration
`turing.minio.endpoint`	`http://localhost:9000`	MinIO server URL
`turing.minio.access-key`	`admin`	MinIO access key
`turing.minio.secret-key`	`minha_senha_forte`	MinIO secret key
`turing.minio.bucket`	`turing-assets`	Bucket name — created automatically on startup if it does not exist

> **NOTE**
>
> The Assets section only appears in the sidebar when `turing.minio.enabled: true`.

MongoDB (Application Logs)

MongoDB is used to persist application logs, making them accessible from the admin console. Disabled by default.

PROPERTY	DEFAULT	DESCRIPTION
`turing.mongodb.enabled`	`false`	Set `true` to enable MongoDB log persistence
`turing.mongodb.uri`	`mongodb://localhost:27017`	MongoDB connection URI
`turing.mongodb.logging.database`	`turingLog`	Database name for logs
`turing.mongodb.logging.collection.server`	`server`	Collection for general server logs
`turing.mongodb.logging.collection.aem`	`aem`	Collection for AEM-specific logs
`turing.mongodb.logging.collection.indexing`	`indexing`	Collection for indexing pipeline logs
`turing.mongodb.logging.purge.days`	`30`	Automatically delete log entries older than this many days

API Documentation (SpringDoc)

PROPERTY	DEFAULT	DESCRIPTION
`springdoc.pathsToMatch`	`/api/**`	API paths included in the OpenAPI spec
`springdoc.swagger-ui.path`	`/swagger-ui.html`	Swagger UI path

Logging

PROPERTY	DEFAULT	DESCRIPTION
`logging.file.name`	`store/logs/turing.log`	Log file path
`logging.logback.rollingpolicy.max-file-size`	`25MB`	Maximum size of each log file before rotation
`logging.logback.rollingpolicy.max-history`	`10`	Number of rotated log files to keep
`logging.level.com.viglet`	`INFO`	Log level for Turing ES application code
`logging.level.org.springframework`	`INFO`	Log level for Spring framework
`logging.level.dev.langchain4j`	`INFO`	Log level for LangChain4j (LLM library)

Keycloak (OAuth2 / OIDC)

When `turing.keycloak: true`, uncomment and configure the following block:

```yaml
spring:
  security:
    oauth2:
      client:
        registration:
          keycloak:
            client-id: demo-app
            client-secret: your-client-secret
            scope: openid
            authorization-grant-type: authorization_code
            redirect-uri: http://localhost:2700/login/oauth2/code/demo-app
        provider:
          keycloak:
            issuer-uri: http://localhost:8080/realms/demo
            jwk-set-uri: http://localhost:8080/realms/demo/protocol/openid-
connect/certs
            user-info-uri: http://localhost:8080/realms/demo/protocol/openid-
connect/userinfo
      resourceserver:
        jwt:
          issuer-uri: http://localhost:8080/realms/demo
```

For full Keycloak setup instructions, see Security & Keycloak.

GenAI / LLM (Database Settings)

LLM providers, models, and GenAI behavior are managed through the **Administration Console** and stored in the database — not in `application.yaml`. The following settings are available via **Settings > Global Settings** in the admin UI:

SETTING	DEFAULT	DESCRIPTION
Default LLM	*(none)*	Global default LLM instance used when a Semantic Navigation site does not specify one
LLM Cache Enabled	`false`	Cache LLM responses to reduce API calls for repeated queries
LLM Cache TTL (ms)	`3600000` (1 hour)	How long cached LLM responses remain valid
LLM Cache Regenerate	`false`	When `true`, regenerate cached responses in the background
RAG Enabled	`false`	Enable Retrieval-Augmented Generation globally
Default Embedding Model	*(none)*	Global default embedding model for vector operations
Default Embedding Store	*(none)*	Global default embedding store instance
Python Executable	*(auto-detected)*	Path to Python 3 for the Code Interpreter tool (also overridable via `turing.code-interpreter.python-executable` in YAML)

Supported LLM providers: OpenAI, Ollama, Anthropic (Claude), Google Gemini, Google Gemini (OpenAI-compatible), and Azure OpenAI.

LLM instances are created under **GenAI > LLM** in the admin console, where you configure the provider, API key, base URL, and model name.

Embedding Store (Database Settings)

Embedding store instances are configured through the **Administration Console** under **GenAI > Embedding Store**. Each instance specifies a vendor, connection URL, credentials, and optional provider-specific options via a JSON field.

Supported vendors:

VENDOR	PLUGIN ID	KEY PROVIDER OPTIONS
ChromaDB	`chroma`	`baseUrl` (default `http://localhost:8000`), `collectionName`, `tenantName`, `databaseName`, `initializeSchema`, `keyToken`, `basicUsername`, `basicPassword`
Milvus	`milvus`	`baseUrl`, `collectionName`, `databaseName`, `token`, `embeddingDimension`, `metricType`, `indexType`, `indexParameters`, `initializeSchema`
PGVector	`pgvector`	JDBC URL (set as instance URL), `tableName` (default `vector_store`), `schemaName` (default `public`), `dimensions`, `distanceType`, `indexType`, `initializeSchema`

Provider options are passed as a JSON object in the **Provider Options** field when creating or editing an embedding store instance.

Common Production Overrides

A minimal production override file (`application-production.yaml`) covering the most important changes:

```yaml
spring:
  profiles:
    active: production
  datasource:
    url: jdbc:mariadb://db-host:3306/turingdb
    username: turing
    password: strong_db_password
    driver-class-name: org.mariadb.jdbc.Driver
  h2:
    console:
      enabled: false

turing:
  url: https://search.yourcompany.com
  allowedOrigins: https://search.yourcompany.com
  ai.crypto.key: your-strong-random-key-here
  minio:
    enabled: true
    endpoint: http://minio:9000
    access-key: your-minio-user
    secret-key: your-minio-password
  mongodb:
    enabled: true
    uri: mongodb://mongo-host:27017

server:
  tomcat:
    accesslog:
      enabled: true
```

Viglet Turing ES: Administration Guide

Viglet Turing ES (https://viglet.com/turing) is an open source enterprise search platform (https://github.com/openviglet) with Semantic Navigation and Generative AI as its main features. All content is indexed in Apache Solr as the primary search engine.

For an overview of the system architecture and component interactions, see Architecture Overview.

Documents and OCR

It can read PDFs and Documents and convert to plain text and also it uses OCR to detect text in images and images into documents.

Semantic Navigation

Semantic Navigation Sites are the central configuration objects in Turing ES. Each site defines what content is indexed, how it is searched, how results are presented, and whether GenAI is enabled. For a conceptual overview, see Core Concepts. For advanced configuration — Targeting Rules, Spotlights, Merge Providers, Facets, and the search response structure — see Semantic Navigation.

Generative AI Administration

The GenAI system is configured across several administration sections. For full configuration details — LLM providers, embedding stores, RAG architecture, Tool Calling, MCP Servers, and AI Agents — see Generative AI & LLM Configuration.

A brief overview of each administration section:

SECTION	PATH	PURPOSE
Settings	Administration → Settings	Global defaults: LLM instance, embedding store, embedding model, Python path, email
LLM Instances	Generative AI → Language Model	Configure connections to Anthropic Claude, OpenAI, Azure OpenAI, Gemini, and Ollama. See LLM Instances
MCP Servers	Administration → MCP Servers	Register external MCP servers (HTTP or stdio) to extend agent tool calling
AI Agents	Administration → AI Agents	Compose agents from an LLM Instance + selected tools + MCP Servers
Knowledge Base	Management → Assets	Upload and organize files in MinIO; files are indexed as vector embeddings and queried by AI Agents. See Assets

Integration

Turing ES exposes its capabilities through four integration options:

METHOD	DESCRIPTION
REST API	Primary method. All search, indexing, and administration operations are available as HTTP endpoints.
GraphQL	`POST /graphql` — for clients that prefer a graph-based query model
Java SDK	Available on Maven Central (`com.viglet.turing:turing-java-sdk`)
JavaScript / TypeScript SDK	Available on npm: `npm install @viglet/turing-sdk`

Turing ES Console

Turing ES has many components: Search Engine, Semantic Navigation, and Generative AI.

Login

When accessing Turing ES, a login page is displayed. The default username is `admin`. The password is defined at first startup via the `TURING_ADMIN_PASSWORD` environment variable — if not set, Turing ES will not create the admin account with a default password.

Set the environment variable before starting Turing ES for the first time:

Windows

```
set TURING_ADMIN_PASSWORD=your_password
```

Linux / macOS

```
export TURING_ADMIN_PASSWORD=your_password
```

Administration

The **Administration** section is accessed via the main sidebar and lives at `/admin-settings`. It manages users, access control, API credentials, global configuration, and system diagnostics.

> **KEYCLOAK MODE**
> When Keycloak is enabled (`turing.keycloak=true`), the **Users**, **Groups**, and **Roles** subsections are hidden — identity and access management is fully delegated to Keycloak. See Security & Keycloak.

Users

Lists all local user accounts in the system.

Form fields:

FIELD	DESCRIPTION
Avatar	Profile picture
Username	Unique login identifier
First Name / Last Name	Display name
Email	User's email address
Password	Account password

Sections:

SECTION	DESCRIPTION
Groups	Add or remove the user from groups
Roles	Displays roles inherited from the user's groups (read-only)

Groups

Organises users into groups for role-based access control.

Form fields:

FIELD	DESCRIPTION
Name	Group name
Description	Purpose or scope of the group

Sections:

SECTION	DESCRIPTION
Users	Add or remove members of this group
Roles	Assign or remove roles granted to this group

Roles

Defines permissions that can be assigned to groups.

FIELD	DESCRIPTION
Name	Role identifier
Description	What this role permits

API Tokens

Manages API tokens used to authenticate REST API requests. Every token is passed in the `Key` request header.

Form fields:

FIELD	DESCRIPTION
Title	A human-readable name for the token
Description	Purpose or owner of this token

> **INFO**
> The token value is generated automatically on creation and displayed once with a copy button. It cannot be retrieved again — store it securely.

Using the token in API requests:

```
curl -X GET "http://localhost:2700/api/sn/Sample/search?
q=cloud&_setlocale=en_US" \
  -H "Key: <YOUR_API_TOKEN>"
```

Global Settings

The central configuration panel for defaults and external service integrations. Divided into four sections:

General

FIELD	DESCRIPTION
Decimal Separator	Choose between period (`.`) and comma (`,`) for numeric display
Python Path	Absolute path to the Python executable (required for Python-based processing)

LLM Settings

FIELD	DESCRIPTION
Default LLM Instance	The LLM used when no site-level instance is configured
Response Cache	Enable caching of LLM responses to reduce latency and cost
Cache Duration	How long cached responses are retained
Regenerate Cache	Button to manually invalidate and rebuild the response cache

> **WARNING**
> Caching LLM responses improves performance but may return stale answers if the underlying content changes frequently. Tune the duration to match your content update cadence.

RAG Settings

FIELD	DESCRIPTION
Enable RAG Globally	Master switch for Retrieval-Augmented Generation across all sites
Default Embedding Model	The model used to vectorize documents at indexing time and queries at search time
Default Embedding Store	The vector store for persisting embeddings (ChromaDB, PgVector, or Milvus)

> **WARNING**
> Changing the Default Embedding Model invalidates all existing embeddings. All indexed content must be re-indexed after changing this setting.

> **NOTE**
> The RAG Settings section is only visible if an embedding store is configured and available. If MinIO is not configured, the Knowledge Base and related RAG options will not appear.

Email Settings

Used by Turing ES to send notifications and test email connectivity.

FIELD	DESCRIPTION
Provider	Email service provider (e.g., Brevo)
API Key	API key for the email provider
Sender Email	The `From` email address
Sender Name	The display name shown to recipients
Recipient Email	Default destination for test emails
Send Test Email	Button to send a test message and verify configuration

System Information

A diagnostic panel to monitor the health of the Turing ES instance. Divided into two sections:

Overview

ITEM	DESCRIPTION
Application Version	Current Turing ES build version
Java Version	JVM version in use
Operating System	OS name and version
Database Status	Connection status of the primary database
RAM Usage	Current and total system memory
JVM Heap	Used and available JVM heap space
Disk Usage	Available storage on the host volume
MongoDB Status	Connected / disconnected (shown only if MongoDB is enabled)
MinIO Status	Connected / disconnected (shown only if MinIO is enabled)

System Variables

A searchable table of all JVM properties and environment variables active at runtime. Useful for verifying configuration at deployment.

Search Engine

The Search Engine section manages connections to the search backend (Apache Solr, Elasticsearch, or Lucene) and the cores (collections) that each Semantic Navigation Site locale maps to.

For full documentation — instance creation, core management, system monitoring, the plugin architecture, and protections — see **Search Engine**.

Semantic Navigation

The **Semantic Navigation** section manages SN Sites — the central configuration objects that drive every search experience in Turing ES. Each site defines its search engine binding, indexed fields, facets, AI settings, and targeting rules.

For full documentation of all SN Site configuration tabs (Settings, Multi Languages, Behavior, Fields, Merge Providers, Targeting Rules, Spotlights, Top Search Terms, Result Ranking, AI Insights, and Generative AI) and the search REST API, see **Semantic Navigation**.

For deep technical content — Targeting Rules Solr query mechanics, Spotlight injection, Merge Provider field overwrite rules, facet operators, and the self-describing search response structure — see **Semantic Navigation**.

Search Engine

The **Search Engine** page (`/admin/se/instance`) manages the search backends that power Semantic Navigation Sites. It is accessible from the **Enterprise Search** section of the sidebar.

Each **Search Engine instance** is a configured connection to a search backend. Semantic Navigation Sites bind to a specific instance when their cores (collections) are created. Turing ES supports three backends via a plugin architecture: **Apache Solr** (recommended), **Apache Lucene** (embedded), and **Elasticsearch**.

Instance Listing

The page displays all configured instances as a grid of cards (title and description). Use the **"New"** button to add an instance. When no instances exist, a blank state guides you to create the first one.

Create / Edit Form

General Information

FIELD	REQUIRED	DESCRIPTION
Title	✅	Display name for this instance — shown in SN Site dropdowns
Description		Free-text notes about this instance

Connection Settings

FIELD	REQUIRED	DESCRIPTION
Vendor	✅	Backend type: `SOLR`, `LUCENE`, or `ES`
Endpoint URL	✅	Connection URL for the backend service

Default endpoint URLs per vendor:

VENDOR	DEFAULT ENDPOINT URL
SOLR (Apache Solr)	`http://localhost:8983/solr`
LUCENE (embedded Apache Lucene)	`/data/turing/lucene`
ES (Elasticsearch)	`http://localhost:9200`

Instance Detail Pages

After creating an instance, its detail page exposes three navigation sections.

Settings

Editing form for all fields listed above, plus a **Delete instance** button. An instance cannot be deleted if any SN Site cores are still using it.

Cores (Collections)

Manages the search indices — called **cores** in Solr terminology — within this search engine instance. Each Semantic Navigation Site locale maps to one core.

Core Listing

The list shows all cores found in the connected backend:

COLUMN	DESCRIPTION
Name	Core identifier (e.g., `my-site_en_US`)
Documents	Number of indexed documents (`numDocs`) reported by the engine
Sites	Badges showing each SN Site and locale using this core, with a country flag for quick recognition

Cores are grouped by locale pattern: `{base}_{lang}_{COUNTRY}` (e.g., `product-docs_pt_BR`). A **search/filter** field at the top narrows the list by core name.

Core Actions

ACTION	DESCRIPTION
Delete core	Permanently removes the core and all its indexed data. **Blocked** if the core is in use by any SN Site — the UI shows which sites and locales are preventing deletion.
Clear documents	Erases all indexed documents from the core without removing the core itself. Useful for a clean re-index without reconfiguring site associations.

Create a New Core

FIELD	REQUIRED	DESCRIPTION
Name / Name Prefix	✅	Core name or prefix
Locale	✅	Language and country (e.g., `en_US`, `pt_BR`)
Append locale to name		Toggle — when enabled, the locale is automatically appended to the prefix, generating the canonical name (e.g., prefix `my-site` + locale `en_US` → `my-site_en_US`)

A **name preview** shows the final core name as you type, before saving.

> **NAMING CONVENTION**
> Use the `{site-name}_{lang}_{COUNTRY}` pattern consistently. This makes it easy to identify which site and locale each core belongs to when browsing multiple instances.

System Information

Live monitoring panel for the connected backend:

ITEM	DESCRIPTION
Status badge	`UP` (green) or `DOWN` (red) — real-time connectivity check
Engine version	Solr or Elasticsearch version string
Lucene version	Underlying Lucene library version
Operating system	OS name and version of the engine host
Java version	JVM version running the search engine
JVM memory	Heap usage reported by the engine
Other properties	Additional dynamic properties returned by the engine's status API

Plugin Architecture

Turing ES uses a plugin architecture to support multiple search backends behind a unified `TurSearchEnginePlugin` interface. The active plugin is resolved at runtime based on the vendor configured per instance. If a vendor is unrecognised, the factory falls back to Solr.

For the full interface reference — all methods across search, index management, schema management, and document operations — and instructions on implementing a new backend, see Developer Guide → Search Engine Plugin Architecture.

Protections

SCENARIO	BEHAVIOUR
Delete core in use	Blocked — the UI displays a list of SN Sites and locales currently using the core
Delete instance in use	Should be avoided — removing an instance with active sites will break indexing and search for those sites

Repository-level **caching** is enabled for search engine instances to avoid repeated database reads during high-frequency searches.

Related Pages

PAGE	DESCRIPTION
Administration Guide	Full console reference
Semantic Navigation	How SN Sites use cores and search engines
Architecture Overview	Solr, Elasticsearch, and Lucene in the system architecture
Configuration Reference	Solr and Elasticsearch timeout settings in `application.yaml`

Semantic Navigation

The **Semantic Navigation** page (`/admin/sn`) manages the search experiences that power every user-facing search in Turing ES. It is accessible from the **Enterprise Search** section of the sidebar.

Each **Semantic Navigation Site** (SN Site) is a complete, independently configured search experience: it defines which content is indexed, which search engine backend is used, how results are presented, which AI features are enabled, and how the front-end should render the response. A single Turing ES instance can host multiple SN Sites.

For a conceptual overview of what SN Sites are and how they relate to the rest of the system, see Core Concepts.

Site Listing

The page displays all configured SN Sites as a grid of cards (title and description). Use the **"New"** button to create a new site. When no sites exist, a blank state guides you to create the first one.

Site Configuration

Clicking a site opens its configuration, organized into tabs. Each tab covers a distinct aspect of the site's behaviour.

Settings

General identity and search engine binding for the site.

ATTRIBUTE	DESCRIPTION
Name	Identifier for the site — used in API URLs (e.g., `/api/sn/{Name}/search`)
Description	Human-readable description shown in the console
Search Engine	The Search Engine instance that stores and searches content for this site. See Search Engine
Thesaurus	Enables thesaurus-based query expansion — matching synonyms and related terms

Multi Languages

Defines which locales are active for this site and maps each locale to a search engine core (collection).

ATTRIBUTE	DESCRIPTION
Language	Locale code for this language variant (e.g., `en_US`, `pt_BR`)
Core	The Solr core (or Elasticsearch index) where content for this locale is stored and searched

Each locale is indexed and searched independently. The `Accept-Language` request header or the `_setlocale` query parameter selects the active locale at search time.

Open Search button: Next to each locale row, the **Open Search** button opens the HTML search page for that locale at:

```
GET http://localhost:2700/sn/<SITE_NAME>
```

Behavior

Controls how the search engine processes queries and assembles results for this site. All settings are stored on the `TurSNSite` entity and managed via `GET / PUT /api/sn/{id}`.

General

SETTING	DESCRIPTION
Rows per Page	Number of results returned per search request
Exact Match	When enabled, terms wrapped in quotes trigger an exact phrase match instead of a tokenized search

Wildcard

SETTING	DESCRIPTION
Wildcard No Results	Automatically appends a wildcard (`*`) to the query when the original search returns no results, broadening the match
Wildcard Always	Always appends a wildcard to every search term, regardless of result count

Use **Wildcard No Results** to recover gracefully from zero-result queries.

> **WILDCARD ALWAYS REDUCES PRECISION**
>
> **Wildcard Always** appends `*` to every query term regardless of result count. This increases recall — more documents match — but significantly reduces relevance precision. Avoid using it in sites where ranking quality matters. Prefer **Wildcard No Results** as a safer fallback.

Facets

SETTING	DESCRIPTION
Facet	Enables or disables faceted navigation for this site
Items per Facet	Maximum number of values displayed per facet in the result panel
Facet Sort	How facet values are ordered: by document count (`COUNT`) or alphabetically (`ALPHABETICAL`)
Facet Type	How **different facets** combine with each other: `AND` or `OR` . Default: `AND`
Facet Item Type	How **values within the same facet** combine with each other: `AND` or `OR` . Default: `AND`

Understanding Facet Type vs Facet Item Type

These two settings control **different levels** of the filter query. Think of a search where the user has selected filters from two facets:

Category facet: selected `News` and `Blog`

Year facet: selected `2024`

Facet Item Type controls how `News` and `Blog` combine (they're items **within the same facet**):

FACET ITEM TYPE	FILTER LOGIC	PRACTICAL EFFECT
OR (recommended)	`category:"News" OR category:"Blog"`	Show documents that are News **or** Blog — **broadens** results within this facet
AND	`category:"News" AND category:"Blog"`	Show documents that are News **and** Blog simultaneously — **narrows** results (often returns 0 results for single-value fields)

Facet Type controls how the Category group and the Year group combine (they're **different facets**):

FACET TYPE	FILTER LOGIC	PRACTICAL EFFECT
AND (recommended)	`(category filter) AND (year filter)`	Results must match **both** facets — **narrows** results progressively as more facets are selected
OR	`(category filter) OR (year filter)`	Results can match **either** facet — **broadens** results when more facets are selected

The four combinations

```
User selects: Category = [News, Blog] + Year = [2024]

┌──────────────────────────────────────────────────────────────────┐
│ Facet Type = AND, Facet Item Type = OR  (recommended default)    │
│                                                                  │
│ Solr fq: (category:"News" OR category:"Blog") AND (year:2024)    │
│                                                                  │
│ "Show me News or Blog articles, but only from 2024"              │
│ → Progressive filtering: each new facet narrows results          │
│ → Multiple values in same facet broaden within that dimension    │
└──────────────────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────────────────────┐
│ Facet Type = AND, Facet Item Type = AND                          │
│                                                                  │
│ Solr fq: (category:"News" AND category:"Blog") AND (year:2024)   │
│                                                                  │
│ "Show me documents tagged both News AND Blog, from 2024"         │
│ → Only useful for multi-valued fields (tags, categories)         │
│ → Single-value fields return 0 results with multiple selections  │
└──────────────────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────────────────────┐
│ Facet Type = OR, Facet Item Type = OR                            │
│                                                                  │
│ Solr fq: (category:"News" OR category:"Blog") OR (year:2024)     │
│                                                                  │
│ "Show me anything that is News, Blog, OR from 2024"              │
│ → Very broad: adding more facets increases results               │
│ → Useful for discovery/exploration interfaces                    │
└──────────────────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────────────────────┐
│ Facet Type = OR, Facet Item Type = AND                           │
│                                                                  │
│ Solr fq: (category:"News" AND category:"Blog") OR (year:2024)    │
│                                                                  │
│ "Show me documents tagged both News AND Blog, OR anything from 2024" │
│ → Uncommon combination                                           │
└──────────────────────────────────────────────────────────────────┘
```

RECOMMENDED CONFIGURATION

For most search interfaces, use **Facet Type = AND** with **Facet Item Type = OR**. This gives users the intuitive behavior of narrowing results as they add facets, while allowing multiple selections within a single facet to broaden within that dimension. This is the standard pattern used by e-commerce sites and enterprise search portals.

Override priority

The facet operators are resolved with a three-level priority:

- **Request-level** (highest) — the `fqOperator` and `fqItemOperator` fields in the POST body override everything
- **Field-level** — each field can set its own `Facet Type` and `Facet Item Type` in the Fields configuration (set to `DEFAULT` to inherit from site)
- **Site-level** (fallback) — the values configured here in the Behavior tab

This allows fine-grained control: for example, the site default can be `Facet Type = AND`, while a `tags` field overrides its `Facet Item Type` to `OR` because tags are multi-valued and should always use OR logic.

Secondary Facets

Any field configured as a facet can additionally be promoted to a **Secondary Facet** in the field's settings. Secondary facets are returned in a separate `widget.secondaryFacet` section of the search response, completely independent from the regular `widget.facet` section. This gives the front-end full control over how to render them — for example, a `content_type` field configured as a secondary facet can be rendered as navigation tabs ("All · Articles · Videos · Downloads") while the remaining facets appear as sidebar filters. The field must be a facet first; secondary facet is a promotion, not a replacement.

Highlighting

SETTING	DESCRIPTION
HL	Enables term highlighting in search result snippets
HL Pre	Opening HTML tag wrapping matched terms (e.g., `<mark>` or `<b>`)
HL Post	Closing HTML tag (e.g., `</mark>` or `</b>`)

The HL Pre and HL Post values are injected directly around matched terms in the `text` and `description` fields of each result. Your front-end application receives the pre-highlighted HTML and renders it as-is.

Spell Check

SETTING	DESCRIPTION
Spell Check	Enables spelling suggestion when the query term is not found or returns few results
Spell Check Fixes	When enabled, automatically re-runs the search using the corrected term and returns those results instead of showing a "Did you mean?" prompt

More Like This (MLT)

SETTING	DESCRIPTION
MLT	Enables the More Like This feature, which finds documents semantically similar to a given result

When MLT is enabled, each search result can be expanded to show related documents — useful for discovery experiences and content recommendation.

Spotlight

SETTING	DESCRIPTION
Spotlight with Results	When enabled, curated Spotlight documents are displayed alongside organic results. When disabled, Spotlights are shown only when the query matches a spotlight term and no organic results are returned

Default Field Mappings

These settings map the SN Site's canonical display fields to the actual field names in the Solr index. They allow the front-end application to always reference a consistent set of fields regardless of how connectors named the fields at indexing time.

SETTING	DESCRIPTION
Title	Field used as the result title
Text	Field used as the main body / snippet text
Description	Field used as the result description
Date	Field used as the result date
Image	Field used as the result thumbnail image URL
URL	Field used as the result link
Default Field	The field Solr uses for full-text search when no specific field is targeted in the query
Exact Match Field	The field used when the user searches with quoted terms (exact match mode)

Fields

Lists all fields configured for this site. Each field defines how a document attribute is indexed, searched, and presented.

Field Listing

COLUMN	DESCRIPTION
Field	Field name as it appears in the Solr schema
Enabled	Whether this field is active
MLT	Whether this field is included in More Like This queries
Facets	Whether this field is exposed as a facet filter
Secondary Facet	Whether this facet is promoted to the `secondaryFacet` section of the search response
Highlighting	Whether highlighted snippets are returned for this field

Field Detail

Clicking a field opens its detail form:

ATTRIBUTE	DESCRIPTION
Name	Field identifier — must match the Solr schema field name
Description	Purpose or contents of this field
Type	Data type: `INT`, `LONG`, `FLOAT`, `DOUBLE`, `CURRENCY`, `STRING`, `DATE`, or `BOOL`
Multi Valued	Whether the field holds an array of values
Facet Name	Display label shown in the search UI facet panel
Facet	Whether to expose this field as a filter facet
Secondary Facet	Promotes this facet to the `secondaryFacet` section of the search response (e.g., for rendering as navigation tabs)
Highlighting	Whether to return highlighted snippets for this field
MLT	Whether this field participates in More Like This queries
Enabled	Whether this field is active
Required	Whether the field must be present when indexing a document
Default Value	Value used when a document is indexed without this field

SECONDARY FACETS

A secondary facet is a regular facet that is additionally returned in a separate `secondaryFacet` section of the search response. This lets the front-end render it differently — for example, as horizontal tabs ("All · Articles · Videos") — while other facets appear as sidebar filters. The field must have **Facet** enabled first; **Secondary Facet** is a promotion, not a replacement.

Field-Level Facet Overrides

Beyond the standard settings, each field can override the site-level facet behavior:

FIELD SETTING	DESCRIPTION
Facet Type	Overrides the site-level operator between this facet and others (`AND` / `OR`)
Facet Item Type	Overrides the site-level operator between values within this facet (`AND` / `OR`)
Facet Range	Enables date range faceting for this field. Instead of listing discrete values, the facet groups documents by predefined date periods: **day**, **month**, or **year**

A date field configured with **Facet Range** groups results into date period buckets rather than listing individual values. The available granularities are **day**, **month**, and **year** — selected in the field configuration. The response returns one facet item per period that has at least one matching document, each with its pre-built filter link.

Custom Facets

Custom Facets let you define the exact items that appear in a facet, each with its own filter rule, label, and position — rather than relying on the values that happen to exist in the index. They are created from an existing field and appear automatically in the search response alongside regular facets, following the site's behavior settings.

This is useful when the raw field values are not user-friendly, when you want to group multiple values into a named bucket, or when you need range-based navigation (e.g., price ranges, date periods, score thresholds).

Custom Facet item operators

Each item in a Custom Facet is defined by an operator (`TurSNSiteCustomFacetOperatorEnum`) that generates its Solr filter query:

OPERATOR	BEHAVIOR	EXAMPLE
EQUAL	Matches documents where the field equals the specified value exactly	`content_type = "video"`
BETWEEN	Matches documents where the field value falls within a range (inclusive)	`price between 100 and 500`
GREATER_THAN	Matches documents where the field value is above the threshold	`score > 80`
LESS_THAN	Matches documents where the field value is below the threshold	`date < 2020-01-01`

Custom Facet item configuration

SETTING	TYPE	DESCRIPTION
label	`string`	The display name shown in the facet panel (e.g., "Last 30 days", "Under $100")
operator	`enum`	The filter operator: `EQUAL`, `BETWEEN`, `GREATER_THAN`, or `LESS_THAN`
rangeStart	`BigDecimal`	Start of a numeric range (used with `BETWEEN` and `GREATER_THAN`)
rangeEnd	`BigDecimal`	End of a numeric range (used with `BETWEEN` and `LESS_THAN`)
rangeStartDate	`Instant`	Start of a date range (used with `BETWEEN` and `GREATER_THAN` for date fields)
rangeEndDate	`Instant`	End of a date range (used with `BETWEEN` and `LESS_THAN` for date fields)
position	`integer`	The order in which this item appears within the facet

How it works

Once a Custom Facet is saved, Turing ES generates the facet items automatically on every search request. Each item appears in the `widget.facet` (or `widget.secondaryFacet`) section of the response with its pre-built filter link, exactly like a regular facet value. The front-end renders it without any special handling — Custom Facets are indistinguishable from regular facets in the response structure.

Example: Price range facet

A field `price` (numeric type) configured as a Custom Facet with four items:

LABEL	OPERATOR	RANGESTART	RANGEEND
Under $50	`LESS_THAN`	—	50
$50 – $200	`BETWEEN`	50	200
$200 – $500	`BETWEEN`	200	500
Over $500	`GREATER_THAN`	500	—

The search response returns these four items as a facet, each with a pre-built filter link — exactly like regular facet values.

Example: Score threshold facet

A field `relevance_score` configured as a Custom Facet:

LABEL	OPERATOR	RANGESTART	RANGEEND
High relevance	`GREATER_THAN`	80	—
Medium relevance	`BETWEEN`	50	80
Low relevance	`LESS_THAN`	—	50

Example: Date range facet

For date fields, use `rangeStartDate` and `rangeEndDate` (Instant values) instead of the numeric fields:

LABEL	OPERATOR	RANGESTARTDATE	RANGEENDDATE
This year	`GREATER_THAN`	`2026-01-01T00:00:00Z`	—
Last year	`BETWEEN`	`2025-01-01T00:00:00Z`	`2025-12-31T23:59:59Z`
Older	`LESS_THAN`	—	`2025-01-01T00:00:00Z`

CUSTOM FACETS USE EXPLICIT VALUES, NOT SOLR EXPRESSIONS

Custom Facet items use concrete `BigDecimal` values (for numeric fields) or `Instant` timestamps (for date fields). Solr query expressions like `NOW-7DAYS` are not supported — use absolute values instead.

Merge Providers

Merge Providers enable Turing ES to detect when two connectors have indexed the same real-world document and merge their fields into a single enriched Solr document.

What they are

Merge Providers solve the problem of having two independent connectors that index different representations of the same real-world document. Without merging, the same content ends up as two separate entries in the search index — one from each connector — with incomplete information in each.

A classic example: an **AEM connector** indexes structured data from the CMS (`model.json`), capturing content metadata such as content type, author, tags, and structured fields. A **WebCrawler** indexes the same page as rendered HTML, capturing the full readable text. Both connectors run independently and neither has access to the data the other collects.

Merge Providers instruct Turing ES to detect when both connectors have indexed the same document and merge their content into a single, enriched Solr document.

How they work

A Merge Provider is configured with:

- **A join key** — a pair of fields, one from each connector, whose values identify the same document. For example, the `url` field in the AEM document equals the `id` field in the WebCrawler document.
- **A set of field overwrite rules** — which fields from the source connector (the one arriving second) should overwrite fields in the destination document (the one already in the index).

When the indexing pipeline receives a document from connector B and finds that a document from connector A already exists in the index with a matching join key value, it fetches the existing document, applies the field overwrite rules, and writes the merged result back to Solr.

```
AEM Connector indexes:
  id:        "aem-12345"
  url:       "https://example.com/products/widget"
  title:     "Widget Pro"
  author:    "Marketing"
  tags:      ["product", "hardware"]
  text:      ""   ← empty, AEM doesn't have full text

WebCrawler indexes:
  id:        "https://example.com/products/widget"
  text:      "The Widget Pro is a high-performance..."   ← full page text
  title:     "Widget Pro - Example"

Merge Provider configuration:
  Join key:        AEM.url  =  WebCrawler.id
  Overwrite field: text  ← WebCrawler.text → AEM document

Result in Solr:
  id:        "aem-12345"
  url:       "https://example.com/products/widget"
  title:     "Widget Pro"
  author:    "Marketing"
  tags:      ["product", "hardware"]
  text:      "The Widget Pro is a high-performance..."   ← from WebCrawler
```

The final document in the index has the structured metadata from AEM and the full text from the WebCrawler, enabling both faceted navigation (using AEM's tags and metadata) and full-text search (using the crawled content).

Configuration

Each Merge Provider entry defines:

FIELD	DESCRIPTION
providerFrom	The source connector whose incoming document provides field values
providerTo	The destination connector whose existing document in the index will be updated
relationFrom	The field in the source document used as the join key
relationTo	The field in the destination document used as the join key
locale	The locale this merge provider applies to
description	Optional description of the merge provider's purpose

Each entry has a set of **overwrittenFields** — the field names from the source that replace values in the destination document.

Important considerations

> **CONNECTOR INDEXING ORDER MATTERS**
>
> The merge is triggered when the **source** connector's document arrives and a matching **destination** document already exists in the index. If the destination connector runs after the source, no merge will occur until the source re-indexes. Always schedule the **destination connector first** so the document is already in Solr when the source arrives.

Join key uniqueness: The field used as the join key must uniquely identify a document within its connector's output. If multiple documents share the same join key value, the merge behavior is undefined.

Field type compatibility: The fields being merged must have compatible types in the Solr schema. Merging a multi-value field into a single-value field, or a date into a string, will cause indexing errors.

Merge scope: Merge Providers are site-scoped. The same connector can participate in different merge configurations across different SN Sites.

Targeting Rules

Targeting Rules are a search-time personalization mechanism that filters results based on user profile attributes — group, role, segment, country, department, or any indexed field. They allow different users to see different content from the same Solr index without maintaining separate cores or running multiple queries.

There is no admin UI for Targeting Rules. Rules are passed by the client in the body of the `POST /api/sn/{siteName}/search` request and applied dynamically at query time. The Solr schema only needs to have the targeting attributes indexed on the documents.

Common use cases:

- Corporate portal: show HR documents only to employees in the HR department
- E-commerce: show promotions for the user's country and loyalty segment
- Intranet: restrict confidential documents to users with the correct access group
- SaaS platform: filter documentation by the user's subscription tier

How the pipeline works

Four rule types

Targeting Rules are sent in the POST request body (see POST Body Parameters). The four types differ in how values are combined across attributes:

1. `targetingRules` — simple list

A flat array of `attribute:value` strings. Values for the **same attribute** are combined with **OR**; different attributes are combined with **AND**.

```
{
  "query": "benefits",
  "targetingRules": ["department:HR", "department:Finance",
"clearance:confidential"]
}
```

Result: documents where `(department=HR OR department=Finance)` AND `(clearance=confidential OR no clearance)`.

2. `targetingRulesWithCondition` — map with default logic

A map of `attribute → list of values`. Behaves identically to `targetingRules` but uses a structured map format instead of a flat list. Values for the **same attribute** are combined with **OR**; different attributes are combined with **AND**.

```
{
  "query": "benefits",
  "targetingRulesWithCondition": {
    "department": ["HR", "Finance"],
    "clearance": ["confidential"]
  }
}
```

Result: equivalent to the `targetingRules` example above — `(department=HR OR department=Finance)` AND `(clearance=confidential OR no clearance)`.

3. `targetingRulesWithConditionAND` — explicit AND

A map of `attribute → list of values`. Every attribute group must be satisfied simultaneously — AND between groups, OR within each group.

```
{
  "query": "promotions",
  "targetingRulesWithConditionAND": {
    "country": ["BR"],
    "language": ["pt"]
  }
}
```

Result: documents matching `country=BR` **AND** `language=pt` (or documents with no restrictions on either attribute).

4. `targetingRulesWithConditionOR` — explicit OR

A map of `attribute → list of values`. Any condition is sufficient — OR across all groups.

```
{
  "query": "discount",
  "targetingRulesWithConditionOR": {
    "segment": ["premium", "gold"],
    "loyalty": ["active"]
  }
}
```

Result: documents matching any of the conditions — `segment=premium`, `segment=gold`, or `loyalty=active`. More permissive than AND.

Solr filter query generation

`TurSolrQueryBuilder` converts the rule type into Solr `fq` clauses via `TurSNTargetingRules`.

AND logic (each attribute group becomes a clause, all clauses joined with AND):

```
(group:admin OR group:user OR (*:* NOT group:*))
AND
(role:editor OR (*:* NOT role:*))
```

Each clause includes `(*:* NOT attribute:*)` — documents that are not tagged with the attribute are always included (the fallback clause). This ensures untagged, unrestricted content is always visible regardless of the active rules.

OR logic (all attribute-value pairs flattened, joined with OR):

```
(attr1:val1 OR attr2:val2)
OR
(*:* NOT attr1:* AND NOT attr2:*)
```

Documents that have none of the targeting attributes are also included.

The fallback clause

Both AND and OR methods include `(*:* NOT attribute:*)` to ensure documents that were never tagged with a targeting attribute are always returned. This means:

Adding targeting rules to a search request does **not** hide untagged content

Only documents explicitly tagged with a **conflicting** attribute value are filtered out

Documents with no targeting attributes are always visible

Practical examples

Example 1 — Corporate portal with department-scoped content

```
POST /api/sn/portal/search
{
  "query": "benefits",
  "targetingRules": ["department:HR", "department:Finance"]
}
```

Returns documents tagged for HR or Finance departments, plus all untagged public documents. A Marketing employee does not see HR-restricted content.

Example 2 — E-commerce: country and language (AND)

```
{
  "query": "promotions",
  "targetingRulesWithConditionAND": {
    "country": ["BR"],
    "language": ["pt"]
  }
}
```

Solr `fq`: `(country:BR OR (*:* NOT country:*)) AND (language:pt OR (*:* NOT language:*))`

Returns only documents for Brazil **and** in Portuguese, plus documents with no country or language restriction. A document tagged `country:US` is filtered out.

Example 3 — Internal system with role and group (AND)

```
{
  "query": "reports",
  "targetingRulesWithConditionAND": {
    "role": ["admin", "manager"],
    "group": ["sales"]
  }
}
```

Solr `fq`: `(role:admin OR role:manager OR (*:* NOT role:*)) AND (group:sales OR (*:* NOT group:*))`

Documents visible to admins or managers (or no role restriction), and within the sales group (or no group restriction).

Example 4 — Promotional content by segment (OR)

```
{
  "query": "discount",
  "targetingRulesWithConditionOR": {
    "segment": ["premium", "gold"],
    "loyalty": ["active"]
  }
}
```

Returns documents matching any condition — premium, gold, or active loyalty. More permissive than AND; used when any segment match is sufficient.

Example 5 — Intranet with access group control

```
{
  "query": "security policy",
  "targetingRules": ["access_group:it", "access_group:security",
"clearance:confidential"]
}
```

Returns documents accessible to IT or Security groups, AND with clearance=confidential (or no clearance). Documents tagged `access_group:directors` remain hidden.

Indexing requirements

The targeting attributes must be **indexed fields** in the Solr schema and **populated at indexing time** by the Dumont DEP connector. Add the desired targeting fields to the SN Site's Fields configuration so they are included in the schema.

If a document is indexed without a targeting attribute (the field is absent or empty), it is treated as unrestricted and always visible — the fallback clause ensures this.

Metrics

Every search request with targeting rules is recorded in `sn_site_metric_access_trs`, allowing analytics on which rule combinations were applied, how often, and how they affect result volume.

Spotlights

Spotlights are curated search results that are pinned to specific search terms. When a user's query matches a spotlight term, configured documents are injected into the result list at defined positions, before the organic (ranked) results at those positions.

What they are

Spotlights are used for:

- Promoting official pages for important queries (e.g., "benefits" always surfaces the HR benefits page first)
- Pinning announcements or featured content to relevant search terms
- Ensuring a specific URL appears for a known high-traffic query

How matching works

Each Spotlight is associated with one or more **terms**. When a search request arrives, the system checks whether the query string contains any of the spotlight terms as a substring. The match is case-insensitive.

For example, a spotlight with the term `annual report` will match the queries `annual report 2024`, `download annual report`, and `annual report Q3` — any query that contains the phrase.

The term cache (`TurSpotlightCache`) keeps all spotlight terms in memory to avoid repeated database lookups on every search request, making the matching step fast regardless of the number of spotlights configured.

How injection works

Each document within a Spotlight has a configured **position** — the ordinal slot in the result list where it should appear. When a spotlight match is detected, the system iterates through the result list and inserts each spotlight document at its configured position, shifting organic results down.

If a spotlight references a document by its Solr ID and that document exists in the index, the full indexed document is retrieved and injected with its live metadata. If the document is not found in the index (e.g., it was deleted or not yet indexed), the system falls back to the raw content configured directly in the spotlight — a manually defined title, description, URL, and type.

```
Search request: "annual report"
                    |
                    ▼
          Spotlight term match found
                    |
                    ▼
      Position 1 ── Spotlight Doc A (pinned)
      Position 2 ── Organic result #1 (shifted)
      Position 3 ── Organic result #2 (shifted)
      Position 4 ── Spotlight Doc B (pinned)
      Position 5 ── Organic result #3 (shifted)
```

Configuring Spotlights

Each spotlight definition contains:

FIELD	DESCRIPTION
name	Descriptive label for the spotlight (admin only)
description	Optional notes about the spotlight's purpose
language	Locale for this spotlight (spotlights are scoped by language)
managed	`1` for managed mode (references indexed documents), `0` for unmanaged (raw content)
provider	Source provider (default: `TURING`)
terms	One or more search terms that trigger this spotlight. Matching is case-insensitive substring matching

A document entry within a spotlight defines:

FIELD	DESCRIPTION
position	The slot in the result list where this document is injected (1-based)
referenceId	The `id` field value of the target document in the Solr index (optional — used in managed mode)
title	Fallback title if the document is not found in the index
content	Fallback description (max 2000 characters)
link	Fallback URL
type	Content type label shown in the UI

Spotlight modes

Spotlights operate in two modes:

Managed mode: The spotlight references documents that exist in the Solr index. The system retrieves live metadata from the index at query time, so the result always reflects the current state of the document (updated title, description, etc.).

Unmanaged mode: The spotlight stores its own raw content (title, description, URL, type) independently of the index. This is useful for pinning external URLs or content that is not indexed in Turing ES.

Top Search Terms

Displays reports of the most frequently searched terms for this site. Turing ES records every search query and aggregates statistics across four time windows:

REPORT	DESCRIPTION
Today	Top 50 terms searched today
This Week	Top 50 terms for the current week
This Month	Top 50 terms for the current month
All Time	Top 50 most searched terms of all time

Each report shows the term and its search count for the selected period. Use this data to identify popular queries for Spotlight configuration, content gaps, and relevance tuning.

Result Ranking

Configures boost expressions that influence search relevance. Each ranking expression boosts documents matching a set of conditions by a configured weight. Expressions are converted into Solr boost queries in the format `(condition)^weight` and applied at query time.

Ranking Expression Form

General Information

ATTRIBUTE	DESCRIPTION
Name	Name for this ranking rule
Description	What this rule does and why

Ranking Conditions

A dynamic list of one or more field-value conditions. At least one condition is required.

ATTRIBUTE	DESCRIPTION
Attribute	An indexed field in the document, selected from the site's configured fields
Condition	`Is` (equals) or `Is not` (not equals)
Value	The value to compare against

Multiple conditions are combined with AND logic — all conditions must match for the boost to apply.

Ranking Weight

A slider from **0** to **10** that sets the boost intensity. Higher weight causes matching documents to rank higher in results.

Example: A condition `type Is news` with weight `8` causes documents of type "news" to rank significantly higher than other results.

AI Insights

The AI Insights tab displays an AI-generated natural language summary of this SN Site — aggregating information about its configuration, fields, behavior settings, and indexed content. The summary is generated on demand and gives administrators a quick overview of what the site is configured to do, without having to navigate through all the tabs.

Click **Generate** to trigger the summary. The response streams in progressively as it is generated.

Generative AI

Configures the RAG (Retrieval-Augmented Generation) system for this site, enabling conversational AI responses grounded in the indexed documents. The tab is divided into four sections.

Site Prompt

A field for generating an automatic description of the site using AI, with streaming support. The generated description can overwrite an existing one after a confirmation dialog.

RAG Activation

ATTRIBUTE	DESCRIPTION
Enabled	Enables the generative AI chat endpoint for this site (`GET /api/sn/{siteName}/chat`)

AI Model & Embedding Configuration

If not configured here, the site inherits the global defaults from **Administration → Settings**.

ATTRIBUTE	DESCRIPTION
LLM Instance	The language model used to generate responses (e.g., Anthropic Claude, OpenAI GPT-4o). See LLM Instances
Embedding Store	The vector store where document embeddings are persisted: ChromaDB, PgVector, or Milvus
Embedding Model	The model used to vectorize documents at indexing time and queries at search time

> **CHANGING THE EMBEDDING MODEL**
>
> Changing the Embedding Model invalidates all existing embeddings for this site. All indexed content must be re-indexed after this change. The Knowledge Base training status in Assets will reflect the stale state.

System Prompt

An editor for the prompt template sent to the LLM on each chat request. The template must include two required variables:

VARIABLE	DESCRIPTION
`{{question}}`	The user's question
`{{information}}`	The context retrieved from indexed documents via similarity search

Saving is blocked if either variable is missing from the template.

How it works: When documents are indexed, `TurSNGenAi.addDocuments()` extracts title, summary, and body text and stores their embeddings in the configured vector store. When a user sends a chat request (`GET /api/sn/{siteName}/chat?q=...`), the system performs a similarity search, builds the prompt with the retrieved context, sends it to the LLM, and returns the generated response via Server-Sent Events (SSE). See Chat for the front-end experience.

Search Response Structure

The full search response schema — including `pagination`, `queryContext`, `results`, and the `widget` object (containing `facet`, `secondaryFacet`, `similar`, `spellCheck`, `spotlights`, `locales`, and `cleanUpFacets`) — is documented in the REST API Reference → Search Response Structure.

Related Pages

PAGE	DESCRIPTION
REST API Reference	Search, autocomplete, spell check, and all other API endpoints
Search Engine	Manage the Solr/Elasticsearch/Lucene backends that SN Sites connect to
Assets	Knowledge Base files for RAG — requires an embedding-capable LLM Instance
LLM Instances	Configure the language models used by the Generative AI tab
Chat	Front-end chat interface for the Semantic Navigation tab
Architecture Overview	How SN Sites fit into the overall system architecture

Integration

The **Integration** page (`/admin/integration/instance`) manages connectors that import and index content from external sources — such as AEM, web crawlers, or other content repositories — into the Turing ES search engine. It is accessible from the **Enterprise Search** section of the sidebar.

Each **Integration instance** represents a configured connection to a content connector application. The connector runs as a separate process (for example, Viglet Dumont DEP) and communicates with Turing ES via REST. Turing ES acts as a proxy: the API path `/api/v2/integration/{integrationId}/**` forwards requests to the configured connector endpoint, with built-in SSRF protection.

Instance Listing

The listing page shows all configured Integration instances as a grid of cards, each displaying the integration's title and description. A button at the top of the page opens the creation form.

Creating or Editing an Integration

The creation and edit form is divided into two sections.

General Information

FIELD	DESCRIPTION
Integration Name	Human-readable identifier for this integration (required)
Short Description	Optional description of the integration's purpose

Connection Details

FIELD	DESCRIPTION
Integration Type	Connector type — **AEM** or **Web Crawler**
Endpoint	URL of the connector application
Enabled	Toggle to activate or deactivate this integration

Integration Detail — Sections

After an integration is created, its detail page provides navigation to several sections. The sections below are common to all connector types. For AEM-specific configuration (Sources, Content Types, Author/Publish environments, Delta Tracking, Locales, Indexing Rules, and the Indexing Manager), see AEM Connector.

Settings

The Settings section contains the same form fields as the creation form (General Information and Connection Details), allowing you to edit the integration name, description, endpoint, type, and enabled state.

Monitoring

The Monitoring section is a real-time dashboard for tracking the indexing pipeline. It displays indexing events as they are received and processed.

Filters:

FILTER	DESCRIPTION
Date From / Date To	Restrict results to a time window
Object ID	Filter by a specific document ID
Status	Filter by processing status (see status table below)
Environment	Author or Publish
Language	Locale code (for example, `en_US`)
Sites	Filter by target SN Site

Auto-refresh: configurable intervals — Off, 1s, 5s, 10s, 30s, 1m, 5m.

Result columns: Date, Object ID, Status, Environment, Language, Sites.

Results are paginated and sortable.

Indexing Status Values

STATUS	MEANING
`PREPARE_INDEX`	Preparing to index the document
`PREPARE_UNCHANGED`	No changes detected since last indexing
`PREPARE_REINDEX`	Preparing a reindexation
`PREPARE_FORCED_REINDEX`	Forced reindexation triggered
`RECEIVED_AND_SENT_TO_TURING`	Document received by the connector and forwarded to Turing ES
`SENT_TO_QUEUE`	Document placed in the Artemis processing queue
`RECEIVED_FROM_QUEUE`	Document consumed from the queue by the indexing pipeline
`INDEXED`	Document successfully indexed in Solr
`FINISHED`	Operation finished
`DEINDEXED`	Document removed from the index
`NOT_PROCESSED` / `IGNORED`	Document skipped due to an Indexing Rule or connector decision

Indexing Stats

The Indexing Stats section provides a table of completed bulk indexing operations for this integration.

COLUMN	DESCRIPTION
Start Time	Timestamp when the operation started
Source	Source that was operated on
Operation	`INDEX_ALL` or `REINDEX_ALL`
Documents	Number of documents processed
Duration	Total elapsed time
Docs/min	Throughput (documents per minute)

Double Check

The Double Check section validates the consistency between the connector's content and the Turing ES search index. It detects drift between what the connector knows about and what is actually indexed.

- Select a **Source** to inspect
- Results are shown in two views:
  - **Missing** — content that exists in the connector but is not present in the index
  - **Extra** — content that is present in the index but no longer exists in the connector
- Results are grouped by Solr core in an accordion, listing the affected document paths

Use Double Check after a partial failure, a forced reindex, or when users report missing or stale search results.

System Information

Displays live diagnostic information from the remote connector application.

Status badge: UP (green) or DOWN (red) — indicates whether the connector endpoint is reachable.

Application:

ITEM	DESCRIPTION
Application	Connector application name
Version	Connector build version
Java Version	JVM version running the connector
Vendor	JVM vendor
JVM	JVM identifier
OS	Operating system name and version

Memory and disk gauges (progress bars with total, used, and free values):

GAUGE	DESCRIPTION
Physical Memory (RAM)	Host system memory
JVM Heap Memory	Java heap utilisation and limits
Disk Space	Available storage on the connector's host volume

Architecture

Turing ES acts as a transparent proxy to the connector. The API path `/api/v2/integration/{integrationId}/**` forwards all requests to the configured connector endpoint, including authentication headers. This design keeps the connector application decoupled — it does not need to be publicly accessible, only reachable from the Turing ES server.

Built-in SSRF protection validates the endpoint before forwarding. Requests to private IP ranges, loopback addresses, or disallowed schemes are rejected.

Related Pages

PAGE	DESCRIPTION
AEM Connector	AEM-specific configuration — sources, content types, environments, indexing rules, and the indexing manager
Semantic Navigation	Configure the SN Sites that receive indexed content
Architecture Overview	End-to-end indexing flow from connector to Solr
REST API Reference	API endpoints for programmatic indexing
Dumont DEP — Connectors	Available connectors (Web Crawler, AEM, Database, FileSystem, WordPress) and how to deploy them
Dumont DEP — REST API	Connector-side API for triggering indexing, monitoring, and source management

AEM Connector

This page covers the configuration sections that are specific to the **Adobe Experience Manager (AEM)** connector. For general integration management — creating instances, monitoring, statistics, and system information — see Integration.

> **DUMONT DEP — AEM CONNECTOR**
>
> For the connector-side documentation — how the AEM plugin extracts content (infinity.json, tags, model.json), event listener setup, and custom extensions — see the Dumont DEP AEM Connector documentation.

Sources

Sources define the content origins that this integration reads from. Each integration can have multiple sources, and each source is configured independently.

General

FIELD	DESCRIPTION
Name	Source identifier
Endpoint	URL of the content source within the connector
Username / Password	Credentials for authenticated access

Root Path

Defines the root content path within the source (for example, the root node in an AEM repository from which content is traversed).

Content Types

FIELD	DESCRIPTION
Content Type	Primary content type to be indexed (for example, `cq:Page`)
Sub Type	Optional sub-type filter within the content type

Delta Tracking

Controls how incremental indexing is handled.

FIELD	DESCRIPTION
Once Pattern	Pattern used to identify content that should only be indexed once
Delta Class	Java class responsible for detecting changed content since the last run

Author / Publish

Configures which AEM environments are indexed and how they map to Turing ES sites.

FIELD	DESCRIPTION
Author	Enable indexing from the AEM author environment
Publish	Enable indexing from the AEM publish environment
SN Site (Author)	Semantic Navigation Site that receives author content
SN Site (Publish)	Semantic Navigation Site that receives publish content
URL Prefix (Author)	URL prefix prepended to document paths in the author index
URL Prefix (Publish)	URL prefix prepended to document paths in the publish index

Locales

Maps content language codes to repository paths.

FIELD	DESCRIPTION
Default Locale	Locale used when no language-specific path is matched
Locale Class	Java class responsible for resolving document locale
Locale → Path	Dynamic list mapping each locale code (for example, `en_US`) to its root path in the repository

Source Actions

Each source has two action buttons:

Index All — triggers a full indexing run for all content in this source

Reindex All — forces a full reindexation, replacing all previously indexed content

Indexing Rules

Indexing Rules allow you to filter content during indexing — for example, to exclude error pages or draft content before it reaches the search index.

Form fields:

FIELD	DESCRIPTION
Name	Rule identifier (required)
Description	Purpose of this rule
Source	The source this rule applies to
Attribute	Document field to evaluate (for example, `template`)
Rule Type	How the rule is applied — currently supports **IGNORE** (skip documents that match)
Values	Dynamic list of values that trigger the rule (add or remove entries)

Example: A rule with `Attribute = template`, `Rule Type = IGNORE`, and `Values = [error-page]` will prevent any document with `template:error-page` from being indexed.

Indexing Manager

The Indexing Manager provides a stepper form for targeting specific documents for manual operations. Four operation types are available:

OPERATION	DESCRIPTION	COLOUR
INDEXING	Index specific content	Blue
DEINDEXING	Remove specific content from the index	Red
PUBLISHING	Publish content	Green
UNPUBLISHING	Unpublish content	Orange

Each operation step allows you to:

- Select the **Source** to operate on
- Choose the **attribute** to identify documents: **ID** or **URL**
- Enter one or more specific values (IDs or URLs)
- Expand **Advanced Settings** to toggle **Recursive** mode, which traverses child content in hierarchical repositories

Related Pages

PAGE	DESCRIPTION
Integration	General integration management — instances, monitoring, statistics, and system information
Semantic Navigation	Configure the SN Sites that receive indexed content
Architecture Overview	End-to-end indexing flow from connector to Solr
REST API Reference	API endpoints for programmatic indexing
Dumont DEP — AEM Connector	How the AEM connector works — infinity.json traversal, event listeners, API triggering
Dumont DEP — AEM Event Listener	Install the OSGi event listener bundle inside AEM for real-time indexing
Dumont DEP — Extending AEM	Custom attribute extractors, content processors, and configuration JSON reference

Generative AI & LLM Configuration

Turing ES integrates Generative AI throughout the platform via **Spring AI**, providing a unified abstraction layer over multiple LLM providers and embedding backends. The GenAI capabilities are organized around four connected concepts: **LLM Instances**, **Tool Calling**, **MCP Servers**, and **AI Agents** — which together define how the system reasons, retrieves, and responds.

RAG (Retrieval-Augmented Generation) is the primary pattern used to ground LLM responses in real content: both the **Knowledge Base** (files stored in MinIO) and **Semantic Navigation Sites** can serve as RAG sources.

Global Settings

Before configuring individual LLM Instances, AI Agents, or RAG sources, establish platform-wide defaults in **Administration → Settings**. These defaults apply wherever no specific override is configured.

SETTING	DESCRIPTION
Default LLM Instance	The LLM provider used when no specific instance is selected in an AI Agent or SN Site
Default Embedding Store	Which vector database backend to use (ChromaDB, PgVector, or Milvus)
Default Embedding Model	The embedding model used to generate vector representations during indexing and at query time
RAG Enabled by Default	Whether RAG is pre-enabled for new Semantic Navigation Sites
Python Path	Absolute path to the Python executable used by the Code Interpreter tool
Email Settings	SMTP configuration for system notifications

⚠ CHANGING THE EMBEDDING MODEL REQUIRES FULL RE-INDEXING

The embedding store and embedding model must be consistent across indexing and query time. Changing the default embedding model after documents have been indexed will cause embedding dimension mismatches and incorrect similarity results. A full re-indexing of all content is required.

LLM Instances

An **LLM Instance** is a named, configured connection to an LLM provider. Multiple instances can coexist — different AI Agents, SN Sites, and the Chat interface can each use a different instance independently.

Turing ES supports **six vendor types**: OLLAMA, OPENAI, ANTHROPIC, GEMINI, GEMINI_OPENAI, and AZURE_OPENAI.

VENDOR	DEFAULT MODEL	EMBEDDING	TOOL CALLING
OLLAMA	`mistral`	✅	✅
OPENAI	`gpt-4o-mini`	✅	✅
ANTHROPIC	`claude-sonnet-4-20250514`	❌	✅
GEMINI	`gemini-2.0-flash`	❌	✅
GEMINI_OPENAI	`gemini-2.0-flash`	❌	✅
AZURE_OPENAI	`gpt-4o`	✅	✅

For the full UI reference — form sections, provider-specific options, generation parameters, capability matrix, and API key security — see LLM Instances.

Embedding Stores & Models

Turing ES delegates vector storage to one of three backends (ChromaDB, PgVector, Milvus) and uses an embedding model to generate and query vectors. The choice of backend and model affects both storage cost and similarity quality.

See Embedding Stores for backend comparison and Embedding Models for provider support, model selection guidance, and configuration details.

RAG Architecture

Retrieval-Augmented Generation (RAG) grounds LLM responses in indexed content. Rather than relying on training data alone, Turing ES retrieves the most semantically relevant documents and injects them into the prompt before generation.

Turing ES supports two RAG sources:

By default, Turing ES retrieves the **top 10** most similar chunks with a similarity **threshold of 0.7**. Documents below the threshold are excluded from the context.

Knowledge Base (Assets)

The Knowledge Base is built from files managed in the **Assets** section — a file manager backed by MinIO. Files are extracted with Apache Tika, truncated to 100,000 characters, split into 1,024-character chunks, embedded, and stored in the active embedding store.

Full documentation is available on the Assets page.

RAG for Semantic Navigation Sites

When GenAI is enabled for an SN Site, indexed documents are also embedded and stored in the vector store alongside their Solr representation. This allows AI Agents and the SN Site chat to retrieve relevant content via semantic similarity. Configure this in the site's **Generative AI** section — see Semantic Navigation → Generative AI.

> **RE-INDEXING EXISTING CONTENT**
> Documents indexed before GenAI was enabled do not have embeddings. A full re-indexing of the site is required to make existing content available for RAG queries.

Tool Calling

Turing ES includes **27 native tools** across 7 categories that AI Agents can invoke autonomously: Semantic Navigation (15), RAG / Knowledge Base (4), Web Crawler (2), Finance (2), Weather (1), Image Search (1), DateTime (1), and Code Interpreter (1).

See Tool Calling for the full tool reference.

MCP Servers

AI Agents can connect to any external server implementing the **Model Context Protocol (MCP)** to access tools beyond the 27 native ones — company-internal systems, proprietary APIs, or public MCP services. Configured in **Administration → MCP Servers**.

See MCP Servers for transport types, configuration form, and examples.

AI Agents

An **AI Agent** combines a specific LLM Instance, a set of tools, and MCP Servers into a single deployable assistant. Each agent appears as a separate tab in the Chat interface and can be purpose-built for specific roles — enterprise search, data research, IT operations, and more.

See AI Agents for configuration, composition examples, and the agent execution flow.

What is RAG?

RAG (Retrieval-Augmented Generation) is the technique that makes AI responses accurate and grounded in your actual content instead of relying solely on the model's training data.

The core idea is simple: **before the LLM generates a response, the system retrieves the most relevant documents from your content and includes them in the prompt**. The LLM then answers based on real, up-to-date information rather than what it memorized during training.

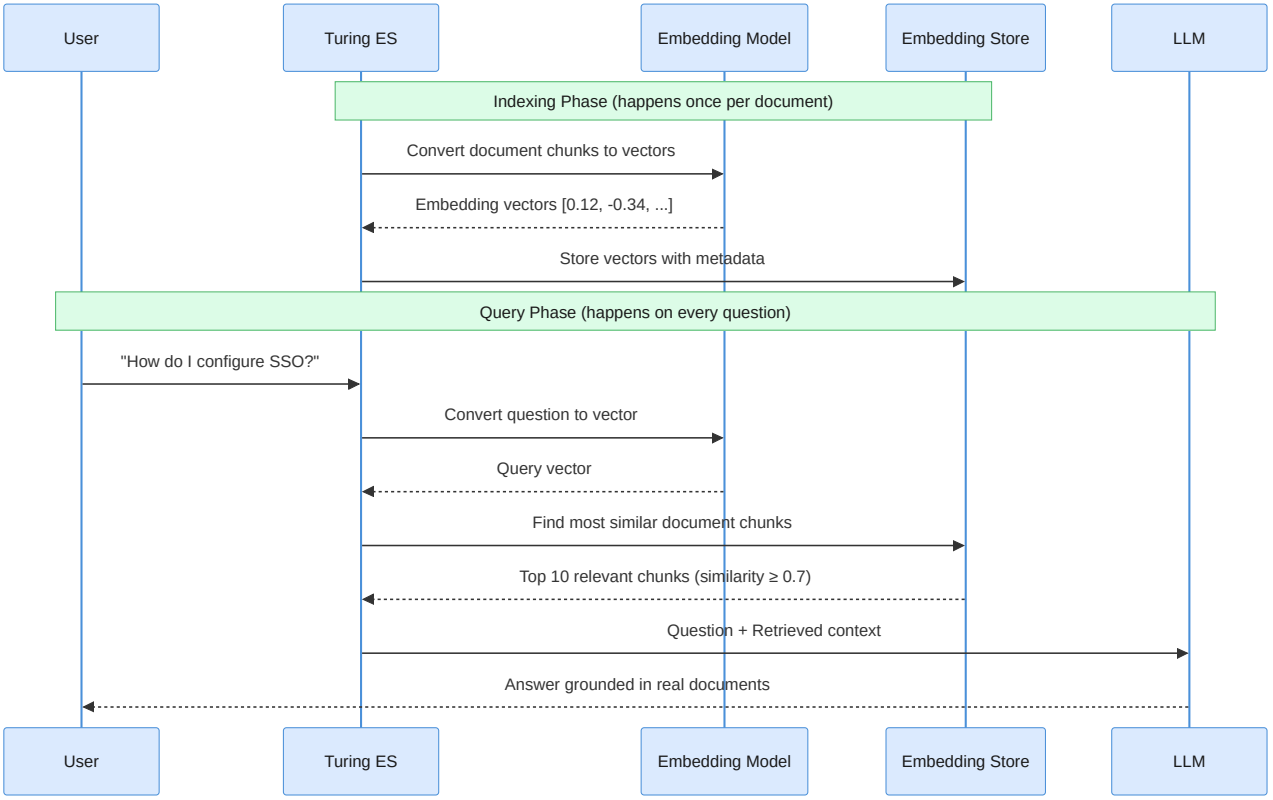
The Problem RAG Solves

Large Language Models are trained on vast amounts of text, but they have critical limitations:

- **Knowledge cutoff** — They don't know about content created after their training date
- **No access to private data** — They've never seen your internal documents, product specs, or company policies
- **Hallucination** — When they don't know an answer, they may generate plausible-sounding but incorrect information
- **Generic responses** — Without specific context, answers are broad and imprecise

RAG eliminates these problems by giving the LLM access to your actual content at query time.

How RAG Works — Step by Step



Phase 1 — Indexing (one-time per document)

Document ingestion — Content enters Turing ES via connectors (AEM, web crawler) or file uploads (Assets/Knowledge Base)

Text extraction — Apache Tika extracts text from PDFs, DOCX, XLSX, HTML, and other formats

Chunking — The extracted text is split into chunks (default: 1,024 characters) to fit within embedding model limits

Embedding — Each chunk is passed through the **Embedding Model**, which converts text into a high-dimensional numerical vector (e.g., a 1,536-dimension array of floats)

Storage — The vectors are stored in the **Embedding Store** alongside metadata (source file, chunk position, original text)

Phase 2 — Retrieval (on every user question)

Query embedding — The user's question is converted into a vector using the **same Embedding Model**

Similarity search — The **Embedding Store** finds the vectors most similar to the query vector (cosine similarity)

Threshold filtering — Only chunks with similarity ≥ 0.7 are included (configurable)

Top-K selection — The top 10 most relevant chunks are selected

Phase 3 — Generation

Prompt construction — Turing ES builds a prompt containing the user's question plus the retrieved document chunks as context

LLM generation — The LLM reads the context and generates an answer based on the real content

Streaming response — The answer is streamed back to the user via SSE

The Three Components

LLM (Large Language Model)

The LLM is the "brain" that reads the retrieved context and generates a natural language response. It does the reasoning, summarization, and articulation — but it does **not** search or retrieve content on its own.

In Turing ES, LLMs are configured as **LLM Instances** supporting six providers:

PROVIDER	EXAMPLE MODELS
OpenAI	GPT-4o, GPT-4o-mini
Anthropic	Claude Sonnet 4
Ollama	Mistral, Llama, Qwen
Google Gemini	Gemini 2.0 Flash
Azure OpenAI	GPT-4o (Azure-hosted)

The LLM does not store knowledge. It only processes what's given to it in the prompt. RAG ensures the prompt contains the right content.

Embedding Model

The Embedding Model is a specialized neural network that converts text into numerical vectors (arrays of floating-point numbers). These vectors capture the **semantic meaning** of the text — similar concepts end up with similar vectors, even if the words are different.

Example:

TEXT	VECTOR (SIMPLIFIED)
"How to configure single sign-on"	`[0.82, -0.15, 0.44, ...]`
"SSO setup and authentication"	`[0.80, -0.13, 0.46, ...]` ← very similar!
"Weather forecast for tomorrow"	`[-0.31, 0.67, -0.22, ...]` ← very different

The embedding model is used **twice**: once during indexing (to embed document chunks) and once at query time (to embed the user's question). Both must use the **same model** — otherwise the vectors are incompatible and similarity search fails.

In Turing ES, providers that support embedding include **OpenAI**, **Ollama**, and **Azure OpenAI**. See Embedding Models for provider details and model selection.

> ⚠ **SAME MODEL FOR INDEXING AND QUERYING**
>
> Changing the embedding model after documents have been indexed causes dimension mismatches and incorrect results. A full re-indexing is required after any model change.

Embedding Store (Vector Database)

The Embedding Store is a specialized database optimized for storing and querying high-dimensional vectors. Unlike a traditional database that matches exact values, an embedding store finds the **most similar** vectors using distance metrics (cosine similarity, dot product, or Euclidean distance).

Turing ES supports three backends:

BACKEND	BEST FOR	HOW IT WORKS
ChromaDB	Development, small/medium deployments	Lightweight, open-source, connects via HTTP API
PgVector	Teams already using PostgreSQL	PostgreSQL extension — embeddings in the same DB as app data
Milvus	Large-scale production, high throughput	Purpose-built vector DB with advanced indexing (IVF, HNSW)

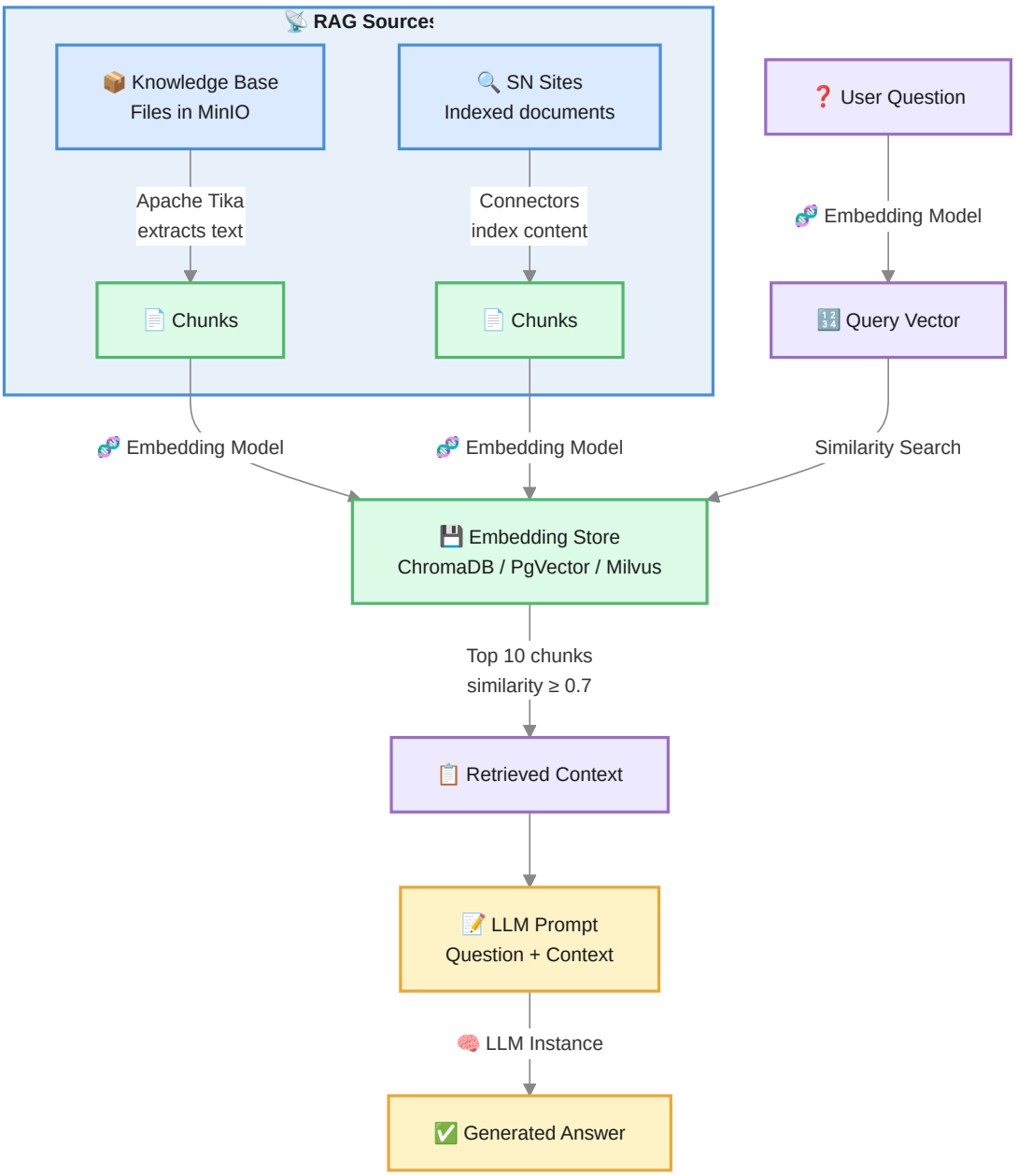
What happens inside the embedding store:

```
Query vector: [0.82, -0.15, 0.44, ...]

Stored vectors:
  doc_chunk_42:  [0.80, -0.13, 0.46, ...]  → similarity: 0.97 ✅ (returned)
  doc_chunk_17:  [0.75, -0.20, 0.39, ...]  → similarity: 0.89 ✅ (returned)
  doc_chunk_91:  [-0.31, 0.67, -0.22, ...]  → similarity: 0.12 ❌ (below
threshold)
```

RAG in Turing ES

Turing ES supports two RAG sources that can be used independently or together:



Knowledge Base (Assets)

Files uploaded to **Assets** are stored in MinIO, extracted with Apache Tika (supporting PDF, DOCX, XLSX, PPTX, HTML, TXT, and more), truncated to 100,000 characters, split into 1,024-character chunks, embedded, and stored in the active embedding store.

The Knowledge Base is queried by the `search_knowledge_base` tool — available in the Chat interface and configurable per AI Agent.

Semantic Navigation Sites

When **GenAI** is enabled for an SN Site, indexed documents are also embedded alongside their Solr representation. This allows the SN Site chat and AI Agents to retrieve relevant content via semantic similarity.

Configure this in **Semantic Navigation → [Site] → Generative AI**. See Semantic Navigation.

> **RE-INDEXING REQUIRED**
> Documents indexed before GenAI was enabled do not have embeddings. A full re-indexing of the site is required to make existing content available for RAG.

Practical Example

Imagine a company with an internal knowledge base containing HR policies, product documentation, and engineering guides.

Without RAG:

> **User:** "What is our vacation policy for remote employees?"
> **LLM:** "Typically, companies offer 15–20 days of PTO..." *(generic, possibly wrong for this company)*

With RAG:

> **User:** "What is our vacation policy for remote employees?"
> **Turing ES retrieves:** Chunk from `hr-policies-2025.pdf` (page 12): *"Remote employees are entitled to 25 days of paid leave per year, plus 5 floating holidays. Requests must be submitted via the HR portal at least 14 days in advance..."*
> **LLM (with context):** "According to our HR policies, remote employees receive 25 days of paid leave per year plus 5 floating holidays. Leave requests must be submitted via the HR portal at least 14 days in advance." *(accurate, sourced from actual company data)*

Key Configuration

SETTING	WHERE	DEFAULT	IMPACT
Embedding Store	Administration → Settings	—	Which vector database to use
Embedding Model	Administration → Settings	Provider default	Quality and dimensions of vectors
Default LLM Instance	Administration → Settings	—	Which model generates responses
RAG Enabled	Administration → Settings	Off	Whether new SN Sites have RAG by default
Top-K results	Internal	10	How many chunks are retrieved
Similarity threshold	Internal	0.7	Minimum similarity to include a chunk

Related Pages

PAGE	DESCRIPTION
Generative AI & LLM Configuration	Global GenAI settings and architecture overview
LLM Instances	Configure LLM providers
Embedding Stores	Vector database backends (ChromaDB, PgVector, Milvus)
Embedding Models	Provider support and model selection guidance
Assets	Knowledge Base file management
AI Agents	Compose agents with RAG tools
Tool Calling	RAG / Knowledge Base tools reference

LLM Instances

The **Language Model** page (`/admin/llm/instance`) is the central place to configure the AI models that power the Turing ES Generative AI features. It is accessible from the **Generative AI** section of the sidebar.

Each **LLM Instance** is a named, configured connection to an LLM provider. Multiple instances can coexist — different AI Agents, SN Sites, and the Chat interface can each use a different instance. This allows you to, for example, use a fast local Ollama model for low-stakes tasks and Anthropic Claude Sonnet for complex reasoning agents.

Instance Listing

The page displays all configured instances as a grid of cards (title and description). Use the **"New language model instance"** button to create a new one.

Create / Edit Form

The form is organized into **5 colour-coded sections** for quick visual orientation.

1. General Information (blue)

FIELD	REQUIRED	DESCRIPTION
Title	✅	Display name for this instance — appears in dropdowns and agent configuration
Vendor	✅	Select the LLM provider. Selecting a vendor applies sensible defaults to Endpoint URL and Model Name automatically.
Description		Free-text notes about this instance's purpose

2. Model Settings (purple)

FIELD	REQUIRED	DESCRIPTION
Endpoint URL	✅	Base URL for the provider API (e.g., `https://api.openai.com`, `http://localhost:11434`)
Model Name	✅	Specific model identifier (e.g., `gpt-4o-mini`, `mistral`, `claude-sonnet-4-20250514`)
API Key		Provider API key — stored encrypted in the database. Leave blank when editing to keep the existing key.

> **API KEY SECURITY**
>
> The API Key field is **write-only**. It is stored encrypted via `TurSecretCryptoService` and never returned in API responses. When editing an existing instance, leaving the field blank preserves the previously saved key. The encryption key is configured in `turing.ai.crypto.key` in `application.yaml`. See Configuration Reference.

3. Generation Parameters (emerald)

Fine-tune how the model generates responses. Defaults are appropriate for most use cases.

FIELD	DESCRIPTION	NOTES
Temperature	Randomness of the output (0.0 = deterministic, 1.0 = very creative)	Applies to all vendors
Top P	Nucleus sampling — restricts token selection to the top P probability mass	Applies to all vendors
Seed	Fixed seed for reproducible outputs	Only available for **OLLAMA**, **OPENAI**, and **AZURE_OPENAI**

4. Advanced Options (amber)

FIELD	DESCRIPTION
Response Format	Output format: `TEXT` (default) or `JSON`
Supported Capabilities	Comma-separated list of feature flags (e.g., `RESPONSE_FORMAT_JSON_SCHEMA`)
Timeout	Maximum time to wait for a response, in ISO 8601 duration format (e.g., `PT60S` = 60 seconds)
Max Retries	Number of retry attempts on transient failures. Default: `3`
Provider Options (Visual)	Vendor-specific fields rendered dynamically based on the selected vendor (see Provider Options below)
Provider Options (JSON)	Raw JSON override for any vendor-specific setting — useful for advanced configurations not exposed in the visual fields

5. Status (slate)

FIELD	DESCRIPTION
Enabled	Toggle to activate or deactivate this instance. Disabled instances are not available for selection in agents or sites.
Tools Enabled	Toggle to allow this instance to use function calling (tools such as web search, code interpreter, etc.)

Supported Vendors

Six vendor types are supported. When a vendor is selected in the form, the Endpoint URL and Model Name fields are pre-filled with the defaults shown below.

VENDOR	DEFAULT ENDPOINT	DEFAULT MODEL
OLLAMA	`http://localhost:11434`	`mistral`
OPENAI	`https://api.openai.com`	`gpt-4o-mini`
ANTHROPIC	`https://api.anthropic.com`	`claude-sonnet-4-20250514`
GEMINI	*(Google native API)*	`gemini-2.0-flash`
GEMINI_OPENAI	`https://generativelanguage.googleapis.com/v1beta/openai`	`gemini-2.0-flash`
AZURE_OPENAI	*(configured via provider options)*	`gpt-4o`

Provider Options

Each vendor exposes additional fields in the **Provider Options (Visual)** section. These fields appear dynamically when that vendor is selected.

VENDOR	AVAILABLE FIELDS
OLLAMA	`embeddingModel`, `topK`, `repeatPenalty`, `numPredict`, `stop`
OPENAI	`embeddingModel`, `maxTokens`
ANTHROPIC	`topK`, `maxTokens`
GEMINI	`topK`, `maxTokens`
GEMINI_OPENAI	`maxTokens`
AZURE_OPENAI	`deploymentName`, `embeddingDeploymentName`, `maxTokens`

AZURE OPENAI

For Azure OpenAI, the `deploymentName` provider option is required — it specifies the name of your deployed model in the Azure portal. The endpoint must also be set to your Azure OpenAI resource URL (e.g., `https://my-resource.openai.azure.com`).

Capabilities by Vendor

Not all vendors support all features. The table below shows which capabilities are available per vendor:

VENDOR	CHAT	EMBEDDING	TOOL CALLING	SEED
OLLAMA	✅	✅ *(configurable)*	✅	✅
OPENAI	✅	✅ (`text-embedding-3-small`)	✅	✅
ANTHROPIC	✅	❌	✅	❌
GEMINI	✅	❌	✅	❌
GEMINI_OPENAI	✅	❌	✅	❌
AZURE_OPENAI	✅	✅ (`text-embedding-ada-002`)	✅	✅

EMBEDDING VENDORS

If you need embedding support (for RAG and the Knowledge Base), use **OLLAMA**, **OPENAI**, or **AZURE_OPENAI**. The other vendors can be used for chat and tool calling but not for vector generation.

Security

API Keys are handled with care at every layer:

Stored encrypted — the key is encrypted via `TurSecretCryptoService` before being persisted to the database in the `apiKeyEncrypted` column.

Never returned — the `apiKey` field is annotated `@Transient` on the JPA entity. It is write-only: it flows in on save but never comes back in API responses or GET endpoints.

Edit safely — leaving the API Key field blank when editing an instance preserves the existing encrypted value without modification.

Encryption key — configured via `turing.ai.crypto.key` in `application.yaml`. **Always set a strong, unique value in production** — the default is a placeholder and must be changed before handling real API keys.

Caching

LLM Instance data is cached at the repository layer to avoid repeated database reads during high-throughput inference:

`turLLMInstancefindAll` — caches the full list of instances

`turLLMInstancefindById` — caches individual instance lookups

Vendor metadata is also cached

Cache entries are invalidated automatically on create, update, or delete.

Related Pages

PAGE	DESCRIPTION
Generative AI & LLM Configuration	Conceptual overview of RAG, embeddings, tool calling, and agents
Chat	Using the chat interface with configured LLM instances
Token Usage	Monitor token consumption per instance
Assets	Knowledge Base files — requires an embedding-capable instance
Configuration Reference	`turing.ai.crypto.key` and other application settings

Embedding Stores

An **Embedding Store** is the specialized vector database that persists and queries document embeddings — the numerical vectors generated by the Embedding Model. It enables similarity search, finding the documents most semantically related to a user's query.

Turing ES supports three backends via Spring AI. The active backend is set globally in **Administration → Settings** and can be overridden per Semantic Navigation Site in its **Generative AI** tab.

Supported Backends

ChromaDB

A lightweight, open-source vector database ideal for development and small to medium deployments.

- Self-hosted, connects via its HTTP API
- Zero infrastructure overhead for teams already running Python tooling
- No special schema setup required — Turing ES manages the collections automatically
- Multi-tenant and multi-database support

Docker Compose quickstart:

```
services:
  chroma:
    image: chromadb/chroma:latest
    ports:
      - "8000:8000"
```

CONFIGURATION	DEFAULT	DESCRIPTION
Base URL	`http://localhost:8000`	Chroma HTTP API endpoint
Collection Name	`turing`	Target collection name
Tenant Name	`default_tenant`	Chroma tenant identifier
Database Name	`default_database`	Chroma database name
Key Token	—	Bearer token for authentication
Basic Username / Password	—	Basic auth credentials

Authentication: ChromaDB supports two methods — **Bearer token** and **Basic auth**. Configure either via the credential field or the provider options. When both are present, the credential field takes precedence.

PgVector

PostgreSQL with the `pgvector` extension — the best choice for deployments that already use PostgreSQL as their primary database.

- Avoids an additional infrastructure dependency
- Embeddings live in the same database as your application data
- Supports standard PostgreSQL backup, replication, and access control
- Connection pooling via HikariCP (max 5 connections per store)

Enable the required extensions in your PostgreSQL instance:

```
CREATE EXTENSION IF NOT EXISTS vector;
CREATE EXTENSION IF NOT EXISTS hstore;
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
```

CONFIGURATION	DEFAULT	DESCRIPTION
JDBC URL	—	PostgreSQL connection string (e.g., `jdbc:postgresql://localhost:5432/turing`)
Username	—	Database user
Password	—	Database password
Table Name	`vector_store`	Table where embeddings are stored
Schema Name	`public`	PostgreSQL schema
Dimensions	—	Vector dimensionality — must match the embedding model
Distance Type	—	`COSINE_DISTANCE`, `EUCLIDEAN_DISTANCE`, or `INNER_PRODUCT`
Index Type	—	`HNSW` or `IVFFLAT`

Table schema created by Turing ES:

```
CREATE TABLE IF NOT EXISTS "vector_store" (
    id       UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
    content  TEXT,
    metadata JSON,
    embedding VECTOR
);
```

Milvus

A purpose-built, cloud-native vector database designed for high-scale similarity search.

- Recommended for large corpora or high-throughput deployments
- Supports distributed operation, horizontal scaling, and advanced index management
- Managed cloud offering available (Zilliz Cloud)

CONFIGURATION	DEFAULT	DESCRIPTION
Base URL	`http://localhost:19530`	Milvus service URI
Collection Name	`turing`	Target collection
Database Name	—	Optional database name
Token	—	Authentication token (`username:password` format)
Embedding Dimension	—	Vector dimensionality — must match the embedding model
Metric Type	—	`COSINE`, `L2`, or `IP` (inner product)
Index Type	—	`HNSW`, `IVF_FLAT`, `IVF_SQ8`, or `DISKANN`
Index Parameters	—	JSON string with index-specific params (e.g., `{"M":16,"efConstruction":200}`)

Store Comparison

FEATURE	CHROMADB	PGVECTOR	MILVUS
Best for	Dev / small-medium	PostgreSQL shops	Large-scale production
Infrastructure	Standalone container	PostgreSQL extension	Dedicated cluster
Scaling	Single node	PostgreSQL replication	Horizontal / distributed
Index types	Automatic	HNSW, IVFFLAT	HNSW, IVF_FLAT, IVF_SQ8, DISKANN
Distance metrics	Cosine, L2	Cosine, L2, Inner Product	Cosine, L2, Inner Product
Multi-tenant	Yes (tenant + database)	Via schema	Yes (database + collection)
Authentication	Token or Basic Auth	JDBC credentials	Token
Managed cloud	—	Any managed PostgreSQL	Zilliz Cloud

Create / Edit Form

Navigate to **Generative AI → Embedding Stores** to manage store instances.

General Information

FIELD	REQUIRED	DESCRIPTION
Title	Yes	Display name for this store instance
Description	Yes	Brief description of its purpose
Vendor	Yes	Select the backend: ChromaDB, PgVector, or Milvus. Selecting a vendor applies default values for Endpoint URL and Collection Name.

Connection

FIELD	REQUIRED	DESCRIPTION
Endpoint URL	Yes	Base URL for the store backend
Collection Name		Name of the collection or table (vendor-specific default applied)
Credential		Authentication token or password — stored encrypted. Leave blank when editing to keep the existing value.

Provider Options

Each vendor exposes additional configuration fields in the **Provider Options** section. These fields appear dynamically when a vendor is selected. A raw JSON editor is also available for advanced configurations.

Status

FIELD	DESCRIPTION
Enabled	Toggle to activate or deactivate this store. Disabled stores are not available for selection.

Vendor defaults applied on selection:

VENDOR	DEFAULT URL	DEFAULT COLLECTION
ChromaDB	`http://localhost:8000`	`turing`
PgVector	`jdbc:postgresql://localhost:5432/turing`	`vector_store`
Milvus	`http://localhost:19530`	`turing`

Collection Management

Each store instance provides a **Collections** page where you can view and manage vector collections.

The collections table shows:

COLUMN	DESCRIPTION
Collection Name	Name of the collection or table
ID	Internal identifier
Document Count	Number of distinct source documents embedded
Chunk Count	Total number of embedding chunks stored

Available actions per collection:

ACTION	DESCRIPTION
Create	Create a new empty collection in the store
Clear	Remove all embeddings from the collection while keeping the collection structure
Delete	Remove the entire collection and all its data

CLEARING OR DELETING COLLECTIONS

Clearing or deleting a collection permanently removes all stored embeddings. A full re-indexing of all content is required to rebuild the vectors. This action cannot be undone.

System Information

The **System Info** endpoint returns backend-specific metadata for monitoring and diagnostics:

BACKEND	INFORMATION RETURNED
ChromaDB	ChromaDB version
PgVector	PostgreSQL version, pgvector extension version
Milvus	Milvus server version

REST API

Store instances are managed via the REST API at `/api/store`.

Store Instance Endpoints

METHOD	ENDPOINT	DESCRIPTION
`GET`	`/api/store`	List all store instances (ordered by title)
`GET`	`/api/store/structure`	Get the structure template for a new instance
`GET`	`/api/store/{id}`	Get a specific store instance
`POST`	`/api/store`	Create a new store instance
`PUT`	`/api/store/{id}`	Update an existing store instance
`DELETE`	`/api/store/{id}`	Delete a store instance

Collection Endpoints

METHOD	ENDPOINT	DESCRIPTION
`GET`	`/api/store/{id}/collections`	List all collections in a store
`POST`	`/api/store/{id}/collections/{name}`	Create a new collection
`DELETE`	`/api/store/{id}/collections/{name}`	Delete a collection
`DELETE`	`/api/store/{id}/collections/{name}/clear`	Clear all embeddings from a collection

System Info Endpoint

METHOD	ENDPOINT	DESCRIPTION
`GET`	`/api/store/{id}/system-info`	Get store backend version and status

Store Vendor Endpoints

METHOD	ENDPOINT	DESCRIPTION
`GET`	`/api/store/vendor`	List all available vendors
`GET`	`/api/store/vendor/{id}`	Get a specific vendor

Global Configuration

Set the default embedding store in **Administration → Settings**:

SETTING	DESCRIPTION
Default Embedding Store	Which vector database backend to use (ChromaDB, PgVector, or Milvus)

Individual Semantic Navigation Sites can override this setting in their **Generative AI** tab. The Knowledge Base always uses the global default.

Security

Credentials are handled with care at every layer:

- **Stored encrypted** — credentials are encrypted via `TurSecretCryptoService` before being persisted in the `credentialEncrypted` column
- **Never returned** — the `credential` field is transient and write-only. It flows in on save but never comes back in API responses
- **Edit safely** — leaving the credential field blank when editing preserves the existing encrypted value

Per-vendor auth — ChromaDB supports Bearer token or Basic Auth; Milvus uses token-based auth; PgVector uses JDBC credentials with connection pooling

Caching

Store instance and vendor data is cached at the repository layer to avoid repeated database reads:

- `turStoreInstancefindAll` — caches the full list of instances
- `turStoreInstancefindById` — caches individual instance lookups
- `turStoreVendorfindAll` / `turStoreVendorfindById` — caches vendor metadata

Cache entries are invalidated automatically on create, update, or delete.

Related Pages

PAGE	DESCRIPTION
Embedding Models	Configure the models that generate vectors stored here
What is RAG?	How embedding stores fit into the RAG pipeline
GenAI & LLM Configuration	RAG architecture, RAG sources, and system overview
LLM Instances	Configure the LLM providers that supply embedding APIs
Assets	Knowledge Base files that are indexed into the Embedding Store
Semantic Navigation	Generative AI tab: per-site embedding overrides

Embedding Models

An **Embedding Model** converts text into a high-dimensional numerical vector that captures its semantic meaning. These vectors enable similarity search — finding documents conceptually related to a query even when they share no common keywords.

Turing ES uses embedding models in two phases:

Indexing — document chunks are vectorized and stored in the Embedding Store

Querying — the user's question is vectorized with the same model and compared against stored vectors

> **SAME MODEL FOR INDEXING AND QUERYING**
>
> The embedding model must remain consistent across both phases. Changing the model after documents have been indexed causes dimension mismatches and incorrect similarity results. A **full re-indexing** of all content is required whenever the embedding model changes.

Supported Providers

Embedding model support depends on the LLM vendor configured in the LLM Instance. Not all vendors provide an embedding API.

PROVIDER	EMBEDDING SUPPORT	EXAMPLE MODELS	DEFAULT MODEL
OpenAI	Yes	`text-embedding-3-small`, `text-embedding-3-large`, `text-embedding-ada-002`	`text-embedding-3-small`
Azure OpenAI	Yes	Deployment name of an embedding model in your Azure resource	`text-embedding-ada-002`
Ollama	Yes	`nomic-embed-text`, `mxbai-embed-large`, `all-minilm`, `stella-v5`	*(configurable)*
Anthropic	No	—	—
Gemini	No	—	—
Gemini (OpenAI-compatible)	No	—	—

OpenAI

Connects to the OpenAI API (default: `https://api.openai.com`) using your API key. OpenAI offers three embedding model families:

MODEL	DIMENSIONS	NOTES
`text-embedding-3-small`	1,536	Best cost-performance balance for most deployments
`text-embedding-3-large`	3,072	Higher quality, larger storage footprint
`text-embedding-ada-002`	1,536	Legacy model — use `3-small` for new deployments

Azure OpenAI

Uses the same OpenAI embedding models, hosted on your Azure tenant. Configuration requires:

Endpoint — your Azure OpenAI resource URL (e.g., `https://my-resource.openai.azure.com`)

Embedding Deployment Name — the deployment name created in the Azure portal

API Key — Azure API key (stored encrypted)

Ollama (Local)

Runs embedding models locally via Ollama. No API key required for local deployments — ideal for air-gapped environments or development.

MODEL	DIMENSIONS	NOTES
`nomic-embed-text`	768	Good general-purpose model, lightweight
`mxbai-embed-large`	1,024	Higher quality, more resource-intensive
`all-minilm`	384	Very lightweight, fast inference

Pull a model before using it:

```
ollama pull nomic-embed-text
```

Local Transformers (ONNX)

Turing ES also supports running embedding models locally via ONNX Runtime, without an external LLM provider. This is useful for deploying custom or fine-tuned models.

SETTING	DESCRIPTION
Model Path	Absolute path to the `.onnx` model file
Tokenizer Path	Absolute path to `tokenizer.json`
Enable GPU	Toggle GPU acceleration via ONNX Runtime
Batch Size	Number of texts to embed per batch

Create / Edit Form

Navigate to **Generative AI → Embedding Models** to manage embedding model configurations.

General Information

FIELD	REQUIRED	DESCRIPTION
Model Name	Yes	Display name for this embedding model
Description		Free-text notes about the model's purpose

Provider

FIELD	REQUIRED	DESCRIPTION
LLM Instance	Yes*	Select the LLM Instance that provides the embedding API. Not required for Local Transformers.
Model Reference	Yes	Technical model identifier (e.g., `text-embedding-3-large`, `nomic-embed-text`)

Local Transformers Options

These fields appear only when **Transformers (Local)** is selected as the provider type:

FIELD	DESCRIPTION
Model Path	Path to the ONNX model file (`.onnx` extension)
Tokenizer Path	Path to the tokenizer file (`tokenizer.json`)
Batch Size	Number of texts processed per inference batch
Enable GPU	Toggle hardware acceleration

Status

FIELD	DESCRIPTION
Enabled	Toggle to activate or deactivate this model. Disabled models are not available for selection.

Choosing a Model

The embedding model determines two things:

Dimensionality — the number of dimensions in the vector (e.g., 384, 768, 1,536, 3,072). Higher dimensions capture more nuance but require more storage.

Semantic quality — how well the model captures meaning. Larger models generally produce better similarity results at the cost of slower indexing.

Recommendations

SCENARIO	RECOMMENDED MODEL	PROVIDER
General production	`text-embedding-3-small`	OpenAI
Maximum quality	`text-embedding-3-large`	OpenAI
Local / air-gapped	`nomic-embed-text`	Ollama
Resource-constrained	`all-minilm`	Ollama
Azure enterprise	`text-embedding-ada-002` deployment	Azure OpenAI
Custom fine-tuned	Your `.onnx` model	Local Transformers

For most deployments, a mid-sized model such as `text-embedding-3-small` (OpenAI) or `nomic-embed-text` (Ollama) provides a good balance between quality and performance.

Global Configuration

Set the default embedding model in **Administration → Settings**:

SETTING	DESCRIPTION
Default Embedding Model	The embedding model used to generate vectors at indexing and query time

Individual Semantic Navigation Sites can override this setting in their **Generative AI** tab.

The Knowledge Base always uses the global default.

REST API

Embedding models are managed via the REST API at `/api/embedding-model`.

METHOD	ENDPOINT	DESCRIPTION
`GET`	`/api/embedding-model`	List all embedding models
`GET`	`/api/embedding-model/structure`	Get the structure template for a new model
`GET`	`/api/embedding-model/{id}`	Get a specific embedding model
`POST`	`/api/embedding-model`	Create a new embedding model
`PUT`	`/api/embedding-model/{id}`	Update an existing embedding model
`DELETE`	`/api/embedding-model/{id}`	Delete an embedding model

Per-Site Override

Each Semantic Navigation Site can override the global embedding model in its **Generative AI** tab. This allows different sites to use different models — for example, a multilingual site might use a model optimized for cross-language embeddings while a technical site uses a domain-specific model.

The site-level configuration includes:

SETTING	DESCRIPTION
Embedding Model	Overrides the global default for this site
Embedding Store	Overrides the global store backend for this site
LLM Instance	The chat/reasoning model for this site's GenAI features

Caching

Embedding model data is cached at the repository layer to avoid repeated database reads during high-throughput indexing:

- `turEmbeddingModelfindAll` — caches the full list of models
- `turEmbeddingModelfindById` — caches individual model lookups

Cache entries are invalidated automatically on create, update, or delete.

Related Pages

PAGE	DESCRIPTION
Embedding Stores	Vector database backends (ChromaDB, PgVector, Milvus)
What is RAG?	How embedding models fit into the RAG pipeline
LLM Instances	Configure the LLM providers that supply embedding APIs
Assets	Knowledge Base files indexed using the embedding model
Semantic Navigation	Per-site GenAI and embedding overrides
GenAI & LLM Configuration	Global settings and architecture overview

Assets

Assets is the file manager that feeds the RAG Knowledge Base in Turing ES. Content managers and administrators use it to upload documents (PDFs, Word files, spreadsheets, and more) that are automatically indexed as vector embeddings and made available to AI Agents for semantic search. It is the bridge between your files and the Knowledge Base — everything uploaded here becomes part of what the AI can reference when answering questions.

The **Assets** section (`/console/asset`) is a file manager with built-in RAG training capabilities. It is available in the **Management** section of the sidebar and is only visible when **MinIO is enabled**.

Assets serves as the Knowledge Base for AI Agents — every file uploaded here can be indexed as vector embeddings and queried by the LLM via tool calling. For the conceptual overview of how this fits into the GenAI architecture, see Generative AI & LLM Configuration.

> **MINIO REQUIRED**
> Assets and all RAG Knowledge Base features require MinIO to be configured. See MinIO Configuration at the bottom of this page.

Layout

The interface uses a **resizable dual-panel layout**:

- **Left panel** — file and folder listing with the action toolbar
- **Right panel** — inline preview of the selected file

A **breadcrumb** at the top of the left panel shows the current folder path and allows navigation to any parent level. A **Root** button returns to the top-level folder instantly.

File Table

The file listing displays the following columns:

COLUMN	DESCRIPTION
Name	File or folder name
Size	File size in human-readable format
Type	MIME type or folder indicator
Last Modified	Date and time of the last modification
AI	Training status — a checkmark indicates the file has been indexed as embeddings, with a tooltip showing the training timestamp
Actions	Per-row download and delete buttons

File Management

Upload Files

Files are uploaded to the **current folder** via a file picker dialog (click **Upload Files**). Multiple files can be selected in one operation. Uploads are sent to:

```
POST /api/asset
```

After upload, an **asynchronous event automatically triggers individual AI indexing** for each uploaded file — no manual training step is needed for new uploads.

Create Folder

A dialog prompts for a folder name. Folders can be nested to any depth and are navigated via the breadcrumb.

Download

Each file has a dedicated download button that preserves the original filename.

Delete

Files and folders can be deleted via an inline button. A **toast notification** confirms completion. When a file is deleted, its **embeddings are automatically removed from the vector store**.

> **NO RENAME**
>
> Renaming files or folders is not currently supported. To rename a file, download it, delete the original, and re-upload with the new name.

Supported File Formats for Training

Apache Tika is the text-extraction engine used during AI Training. It supports a broad range of document types:

CATEGORY	FORMATS
Documents	PDF, DOCX, DOC, ODT, RTF, EPUB
Spreadsheets	XLSX, XLS, ODS, CSV
Presentations	PPTX, PPT, ODP
Web / Markup	HTML, XHTML, XML
Plain text	TXT, LOG, Markdown
Email	EML, MSG, MBOX
Images (with OCR)	PNG, JPEG, TIFF, BMP, GIF (requires Tesseract)

Files that Tika cannot extract text from (e.g. binary executables, ZIP archives without textual content) are **silently skipped** during training with no error counted.

Preview Panel

Selecting a file opens an inline preview in the right panel without leaving the page. Supported formats:

CATEGORY	FORMATS
Images	PNG, JPEG, GIF, WebP, SVG, BMP
PDFs	Rendered via iframe
Video	MP4, WebM, OGG (with player controls)
Audio	MP3, OGG, WAV, WebM (with player controls)
Text	TXT, CSV, HTML, CSS, JS, JSON, XML

Panel actions:

- **Maximise** — opens fullscreen view (press `Esc` to close)
- **Download** — downloads the file directly from the preview panel
- **Close** — collapses the preview panel

The panel footer displays the file size, content type, modification date, and file extension.

AI Training (RAG)

The AI training features are only available when `ragEnabled=true` **and** an embedding model and embedding store are configured in **Administration → Global Settings → RAG Settings**.

Training Status per File

The **AI column** in the file table shows the indexing state of each file:

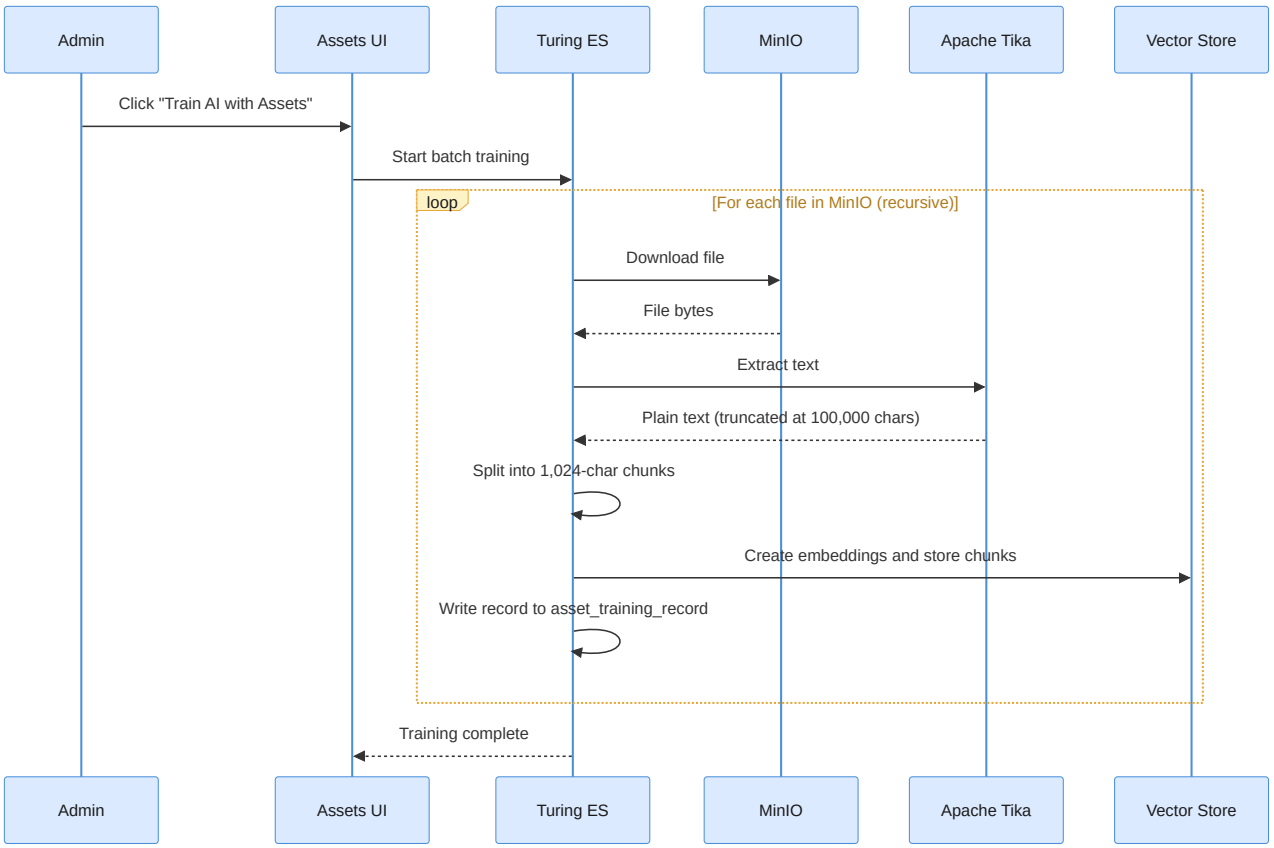
✅ **Checkmark** — file has been indexed; hover to see the training timestamp
(empty) — file has not yet been indexed

Automatic Training on Upload

When a file is uploaded, Turing ES dispatches an **asynchronous event** that indexes the file individually without any user action required. Similarly, when a file is deleted, its embeddings are automatically purged from the vector store.

Batch Training

To index all existing files at once — useful after enabling RAG on an existing installation, or after changing the embedding model — use the **"Train AI with Assets"** button.



Batch training steps for each file:

Download file bytes from MinIO

Extract plain text via **Apache Tika** — supports PDF, DOCX, XLSX, PPTX, HTML, TXT, and images (with OCR)

Truncate text to **100,000 characters**

Split into **chunks of 1,024 characters**

Generate embeddings and store in the configured vector store

Write a record to `asset_training_record` with timestamp

Progress monitoring — while the batch is running, the UI polls every **3 seconds** and displays:

```
X / Y files processed, Z errors
```

Training states: `IDLE` → `RUNNING` → `COMPLETED` / `FAILED`

> **⚠ RE-TRAINING AFTER EMBEDDING MODEL CHANGE**
>
> If you change the Default Embedding Model in **Administration → Global Settings → RAG Settings**, all existing embeddings become invalid. Run "Train AI with Assets" again to re-index all files with the new model.

Embedding Metadata

Each chunk stored in the vector store carries the following metadata:

FIELD	VALUE
`source`	`"minio-asset"`
`objectName`	Full object path in MinIO
`objectPath`	Folder path within the bucket
`fileName`	Original filename
`contentType`	MIME type of the source file
`size`	File size in bytes

This metadata is used by AI Agents when returning search results, so the LLM can cite the source file and provide context about where the information came from.

Text Extraction Limits

There is **no hard file size limit** for uploads — MinIO accepts files of any size. However, during AI Training the extracted text is **truncated to 100,000 characters** before chunking. For very large documents this means only the first portion of the content is indexed. The truncation limit is defined by `TurRagUtils.MAX_TEXT_LENGTH`.

How AI Agents Use the Knowledge Base

Once files are indexed, AI Agents can query the knowledge base via four built-in tools:

TOOL	DESCRIPTION
`search_knowledge_base`	Semantic similarity search across all indexed chunks
`knowledge_base_stats`	Returns total files, chunks, and storage size
`list_knowledge_base_files`	Lists all indexed files, with optional keyword filter
`get_file_from_knowledge_base`	Retrieves the full indexed content of a specific file

For details on configuring AI Agents and tools, see AI Agents and Tool Calling.

MinIO Configuration

MinIO must be enabled and configured before Assets becomes available:

```
turing.minio.enabled=true
turing.minio.endpoint=http://minio:9000
turing.minio.accessKey=minioadmin
turing.minio.secretKey=minioadmin
turing.minio.bucket=turing-assets
```

The bucket (`turing-assets` by default) is **created automatically on startup** if it does not exist.

With Docker Compose, add the MinIO service alongside Turing ES:

```
minio:
  image: minio/minio
  ports:
    - "9000:9000"
    - "9001:9001"
  environment:
    MINIO_ROOT_USER: minioadmin
    MINIO_ROOT_PASSWORD: minioadmin
  command: server /data --console-address ":9001"
  volumes:
    - minio_data:/data
```

TIP

The MinIO web console is available at `http://localhost:9001` when running locally via Docker Compose. Use it to inspect buckets and verify that files are being stored correctly.

REST API Reference

All endpoints are under `/api/asset` and require authentication.

METHOD	ENDPOINT	DESCRIPTION
GET	/api/asset?prefix=	List files and folders in a prefix (folder)
POST	/api/asset	Upload one or more files (multipart/form-data)
GET	/api/asset/download?objectName=	Download a file (Content-Disposition: attachment)
GET	/api/asset/preview?objectName=	Preview a file inline (Content-Disposition: inline)
GET	/api/asset/metadata?objectName=	Retrieve file metadata (size, type, date)
POST	/api/asset/folder?path=	Create a new folder
DELETE	/api/asset?objectName=	Delete a file or folder
POST	/api/asset/train	Start batch AI Training
GET	/api/asset/train/status	Poll current training status
GET	/api/asset/train/records?objectNames=	Get training timestamps for specific files

Related Pages

PAGE	DESCRIPTION
Embedding Stores & Models	Configure vector database backends and embedding models used by AI Training
Tool Calling	All 27 native tools, including the four Knowledge Base tools
AI Agents	Compose agents that query the Knowledge Base via tool calling
Generative AI & LLM Configuration	Platform-wide GenAI settings and RAG architecture overview

Tool Calling

A **Tool Calling** is a function that the LLM can invoke autonomously during a conversation to retrieve information or perform an action. Instead of relying purely on training data, the LLM calls tools to fetch live data, search indexed content, execute code, and more — then incorporates the results into its reasoning.

Turing ES includes **27 native tools** organized into 7 categories, plus support for external tools via MCP Servers. Tools are enabled per AI Agent — each agent selects only the tools it needs.

Semantic Navigation — 15 tools

These tools allow the LLM to interact with any Semantic Navigation Site as a structured knowledge source, enabling rich search-based reasoning over your indexed content.

TOOL	DESCRIPTION
`list_sites`	Lists all available SN Sites with their locales
`get_site_fields`	Returns valid facet fields for filtering a specific site
`get_valid_filter_values`	Returns valid values for a specific filter or facet
`search_site`	Searches a site and returns compact results (ID, title, URL, snippet)
`get_document_details`	Retrieves full text and metadata of a document by ID
`get_search_suggestions`	Autocomplete and spelling corrections for a search query
`find_similar_documents`	Finds semantically similar documents (More Like This)
`get_aggregated_stats`	Calculates totals and distributions by category via facet aggregation
`get_document_highlights`	Extracts snippets from a document where search terms appear
`compare_items`	Compares specific fields of two or more documents side by side
`search_recent_updates`	Retrieves the most recently updated content on a topic
`get_facet_summary`	Statistical summary of all available categories and attributes
`search_by_date_range`	Searches documents within a date range
`lookup_facet_value`	Searches a term across all facets to find exact values
`discover_facet_values`	Splits a phrase into words and searches across all facets

Prompt examples:

"Search for documents about authentication in the Sample site" → triggers `search_site`

"What fields can I filter by on the Sample site?" → triggers `get_site_fields`, then `get_valid_filter_values`

"Show me the full content of document ID abc-123" → triggers `get_document_details`

"Find documents similar to the article about Solr configuration" → triggers `find_similar_documents`

"How many documents do we have per content type?" → triggers `get_aggregated_stats` or `get_facet_summary`

"What articles were updated in the last 7 days about security?" → triggers `search_recent_updates` or `search_by_date_range`

"Compare the features of Product A and Product B" → triggers `search_site` then `compare_items`

RAG / Knowledge Base — 4 tools

These tools enable the LLM to query the **Knowledge Base** — files indexed from MinIO via the Assets section — using semantic similarity search.

TOOL	DESCRIPTION
`search_knowledge_base`	Searches for relevant documents by semantic similarity (top 10, threshold 0.7)
`knowledge_base_stats`	Returns statistics: total files, chunks, and storage size
`list_knowledge_base_files`	Lists all indexed files, with optional keyword filter
`get_file_from_knowledge_base`	Retrieves the full indexed content of a specific file

Prompt examples:

"What does our internal documentation say about deployment procedures?" → triggers `search_knowledge_base`

"How many files are indexed in the knowledge base?" → triggers `knowledge_base_stats`

"Show me the full content of the file onboarding-guide.pdf" → triggers `list_knowledge_base_files` then `get_file_from_knowledge_base`

Web Crawler — 2 tools

TOOL	DESCRIPTION
`fetch_webpage`	Fetches a web page by URL and returns its content as plain text
`extract_links`	Extracts all links from a web page, with optional keyword filter

Prompt examples:

"What does the Apache Solr documentation say about faceted search?" → triggers `fetch_webpage`

"List all links on our company blog homepage" → triggers `extract_links`

Finance — 2 tools

TOOL	DESCRIPTION
`get_stock_quote`	Current price and market data for a stock ticker symbol
`search_ticker`	Looks up a ticker symbol by company name or keyword

Prompt examples:

"What is the current stock price of Apple?" → triggers `search_ticker` then `get_stock_quote`

"Show me the market data for MSFT" → triggers `get_stock_quote`

Weather — 1 tool

TOOL	DESCRIPTION
`get_weather`	Current weather and forecast for a city (1–7 day range)

Prompt examples:

"What's the weather forecast for São Paulo this week?" → triggers `get_weather`

Image Search — 1 tool

TOOL	DESCRIPTION
`search_images`	Searches the web for images and returns URLs and descriptions

Prompt examples:

"Find images of enterprise search architecture diagrams" → triggers `search_images`

DateTime — 1 tool

TOOL	DESCRIPTION
`get_current_time`	Returns the current date and time for a given IANA timezone

Prompt examples:

"What time is it right now in Tokyo?" → triggers `get_current_time`

Code Interpreter — 1 tool

TOOL	DESCRIPTION
`execute_python`	Executes Python code in a sandboxed environment and returns stdout/stderr

The Code Interpreter runs Python in an isolated sandbox directory with a **30-second execution timeout**. It supports:

- Standard output capture
- Matplotlib chart generation (with `Agg` backend — no display required)
- Automatic `print()` wrapping for bare expressions

The Python executable path is configured in **Administration → Settings → Python Path**.

Prompt examples:

"Calculate the compound interest on $10,000 at 5% for 10 years" → triggers `execute_python`

"Generate a bar chart showing monthly sales: Jan=100, Feb=150, Mar=120" → triggers `execute_python` with matplotlib

External Tools via MCP Servers

Beyond the 27 native tools, AI Agents can access tools from any external server implementing the **Model Context Protocol (MCP)**. This covers company-internal systems, proprietary data APIs, and the growing ecosystem of public MCP servers.

See MCP Servers for configuration details.

Related Pages

PAGE	DESCRIPTION
AI Agents	How to compose agents with the tools they need
MCP Servers	Extend agents with external tools via MCP
Assets	Knowledge Base files queried by RAG tools
Semantic Navigation	The search experience powering SN tools

MCP Servers

The **Model Context Protocol (MCP)** is an open standard that lets AI models call tools hosted on external servers — turning any API, database, or internal system into a capability that an AI Agent can use autonomously. If you are an administrator looking to extend your agents beyond the 27 built-in tools, MCP Servers let you connect to virtually any external service without writing custom code inside Turing ES.

An **MCP Server** extends the capabilities of an AI Agent by connecting it to any external server that implements the **Model Context Protocol (MCP)**. This allows Turing ES agents to use tools defined completely outside the platform — a company-internal knowledge system, a proprietary data API, a database query interface, or any of the growing ecosystem of public MCP servers.

MCP Servers are configured in **Administration → MCP Servers** and then selected per AI Agent.

Configuration Form

FIELD	DESCRIPTION
Name	Display name for the MCP server — shown when selecting it in an AI Agent
Transport type	`HTTP` or `stdio` — how Turing ES communicates with the server
Command	For `stdio`: the executable to launch the MCP server process (e.g., `npx`)
Arguments	For `stdio`: command-line arguments passed to the process (e.g., `@modelcontextprotocol/server-filesystem /data`)
URL	For `HTTP`: the MCP server endpoint URL
Execution mode	`Synchronous` or `Asynchronous`

Transport Types

stdio

Launches the MCP server as a **local subprocess**. The MCP client communicates with it via standard input/output streams. This is the standard mode for locally installed MCP servers.

```
# Example: filesystem MCP server
npx @modelcontextprotocol/server-filesystem /path/to/allowed/directory
```

Use stdio when the MCP server is installed locally (via npm, pip, or a system binary) and does not expose an HTTP endpoint.

HTTP

Connects to a **remote MCP server** over HTTP. This mode supports both public MCP services and internally hosted servers accessible over the network.

```
https://mcp.example.com/api/tools
```

Use HTTP when the MCP server is a hosted service or when it runs on a different machine or container.

Execution Modes

Synchronous

Waits for the MCP tool result before continuing the agent's reasoning chain. The LLM receives the tool result and can incorporate it into its next reasoning step before deciding whether to call another tool or generate a final response.

Use synchronous mode when the tool result is needed as context for subsequent steps.

Asynchronous

Dispatches the tool call without blocking. Useful for long-running operations or when the result does not need to be incorporated into the immediate reasoning chain.

MCP Server Examples

Internal ticketing system (stdio):

```
Name:      JIRA Internal
Transport: stdio
Command:   node
Arguments: /opt/mcp-jira/index.js --token <TOKEN>
Mode:      Synchronous
```

Company knowledge API (HTTP):

```
Name:      Internal Knowledge Base API
Transport: HTTP
URL:       https://internal-api.example.com/mcp
Mode:      Synchronous
```

Public MCP service (HTTP):

```
Name:      Brave Search
Transport: HTTP
URL:       https://api.search.brave.com/mcp
Mode:      Asynchronous
```

Assigning MCP Servers to Agents

Once configured, MCP servers appear in the **MCP Servers** multi-select field of the AI Agent configuration form. Each agent independently selects which MCP servers it can use. The tools exposed by each selected MCP server become available to the LLM during that agent's conversations.

Related Pages

PAGE	DESCRIPTION
AI Agents	Compose agents with native tools + MCP servers
Tool Calling	The 27 native tools built into Turing ES
Chat	Front-end chat interface where agents are used

AI Agents

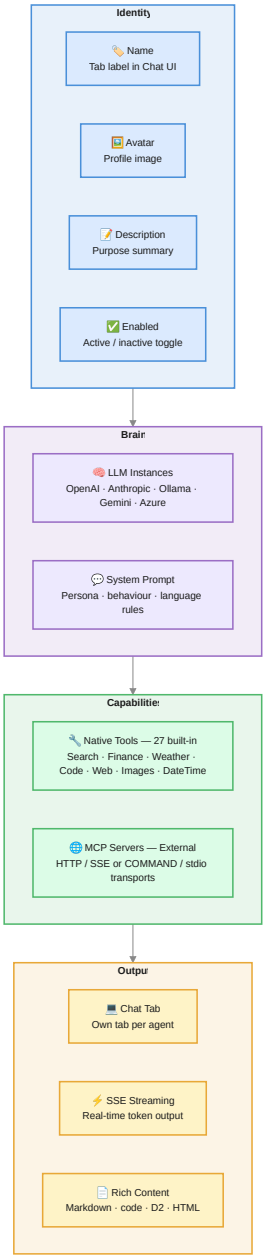
An **AI Agent** is the central composition object in Turing ES's GenAI system. It combines a specific LLM Instance, a selected set of tools, and a set of MCP Servers into a single, named, deployable assistant.

Each agent has its own personality, capability set, and visual identity. In the **Chat** interface, every configured agent appears as a separate tab — users choose which agent to interact with based on its name and description.

AI Agents are configured in **Administration → AI Agents**.

Agent Composition

An AI Agent is built from four layers: **Identity**, **Brain**, **Capabilities**, and **Output**.



Configuration Form

The agent form is organized into **four tabs** accessible from the agent detail page.

Settings

FIELD	REQUIRED	DESCRIPTION
Name	Yes	Display name shown as the tab label in the Chat interface
Avatar		Profile image representing the agent in the chat UI — supports upload and removal
Description		Brief explanation of the agent's purpose and specialization
System Prompt		Instructions sent as a system message before every conversation. Defines persona, behaviour, and language rules.
Enabled		Toggle to activate or deactivate the agent. Disabled agents do not appear in the Chat interface.

DEFAULT SYSTEM PROMPT

If the system prompt is left blank, the agent uses a built-in default:

"You are an AI assistant. Answer the user's questions using the tools available to you. If you have access to MCP server tools, use them when relevant to fulfill the user's request. If the user asks in a specific language, respond in that same language."

LLM

Select one or more LLM Instances that this agent can use for inference. The list shows each instance's title, description, vendor, and model name. At chat time, the user (or frontend) specifies which instance to use from the agent's allowed set.

See LLM Instances for configuration details.

Tools

Select which of the **27 native tools** (across 7 categories) are available to this agent. Tools are displayed grouped by service — each group has a select-all checkbox for quick configuration.

CATEGORY	EXAMPLES
Semantic Navigation	`list_sites`, `search_site`, `get_document_details`, `find_similar_documents`, `search_by_date_range`
RAG / Knowledge Base	`search_knowledge_base`, `list_knowledge_base_files`, `get_file_content`, `knowledge_base_stats`
Web Crawler	`fetch_webpage`, `extract_links`
Finance	`get_stock_quote`, `search_ticker`
Weather	`get_weather`
Image Search	`search_images`
Date / Time	`get_current_time`
Code Interpreter	`execute_python`

A lean tool list reduces prompt length and helps the LLM make more precise tool choices.

See Tool Calling for the full tool reference.

MCP Servers

Select which external MCP servers this agent can call. The list shows each server's title, description, and connection type badge:

BADGE	TRANSPORT	DESCRIPTION
HTTP (blue)	SSE over HTTP	Web-based MCP servers
COMMAND (amber)	stdio	Local process-based MCP servers

See MCP Servers for configuration details.

Composing Agents for Specific Roles

Because each agent independently selects its LLM Instance, tools, and MCP servers, it is straightforward to build purpose-specific assistants.

Enterprise Search Agent

An agent that helps users find and explore indexed content across the organization.

FIELD	VALUE
LLM Instance	Anthropic Claude Sonnet
Tools	`list_sites`, `search_site`, `get_document_details`, `find_similar_documents`, `search_by_date_range`
MCP Servers	—

Data Research Agent

A multi-purpose agent that can browse the web, query financial data, and run data analysis scripts.

FIELD	VALUE
LLM Instance	OpenAI GPT-4o
Tools	`fetch_webpage`, `extract_links`, `get_stock_quote`, `get_weather`, `execute_python`, `search_knowledge_base`
MCP Servers	Internal data API (HTTP MCP)

IT Operations Agent

A local agent for internal IT queries — runs fully on-premise using a local LLM.

FIELD	VALUE
LLM Instance	Ollama (local Llama 3)
Tools	`execute_python`, `get_current_time`, `search_knowledge_base`
MCP Servers	Internal ticketing system (stdio MCP)

How an Agent Executes

When a user sends a message to an AI Agent, the following loop runs:

User Input — the user sends a message (text, file attachments, or follow-up) via the agent's Chat tab.

Prompt Construction — Turing ES builds the prompt from the agent's system prompt, tool definitions (native + MCP), and the full message history.

LLM Inference — the LLM Instance processes the prompt and decides whether to respond directly or call tools.

Tool Execution — if tools are needed, the LLM requests a tool call (name + arguments). Turing ES executes it (native tool or MCP server) and returns the result.

Multi-step Reasoning — the LLM analyses the tool results and may request additional tool calls in a reasoning chain, looping back to step 4 until satisfied.

Final Response — the LLM generates the final answer, grounded in the tool results and conversation context.

Chat Rendering — the response is streamed to the user via SSE with full rich content rendering (Markdown, code blocks, D2 diagrams, HTML, download links).

All tool invocations are wrapped with logging that records the tool name, input, execution time, and response length for debugging.

REST API

Agent Management

METHOD	ENDPOINT	DESCRIPTION
`GET`	`/api/ai-agent`	List all agents (ordered by title)
`GET`	`/api/ai-agent/structure`	Get empty structure template for a new agent
`GET`	`/api/ai-agent/{id}`	Get a specific agent
`POST`	`/api/ai-agent`	Create a new agent
`PUT`	`/api/ai-agent/{id}`	Update an existing agent
`DELETE`	`/api/ai-agent/{id}`	Delete an agent

Agent Chat

METHOD	ENDPOINT	DESCRIPTION
`POST`	`/api/v2/ai-agent/{agentId}/chat`	Stream chat response (SSE). Request body: `{ llmInstanceId, messages[] }`
`GET`	`/api/v2/ai-agent/{agentId}/chat/context-info`	Get LLM context window size. Query param: `llmInstanceId`

Native Tools

METHOD	ENDPOINT	DESCRIPTION
`GET`	`/api/native-tool`	List all available tool groups with tool names and descriptions

Caching

Agent data is cached at the repository layer to avoid repeated database reads:

- `turAIAgentfindAll` — caches the full list of agents
- `turAIAgentfindById` — caches individual agent lookups

Cache entries are invalidated automatically on create, update, or delete.

Related Pages

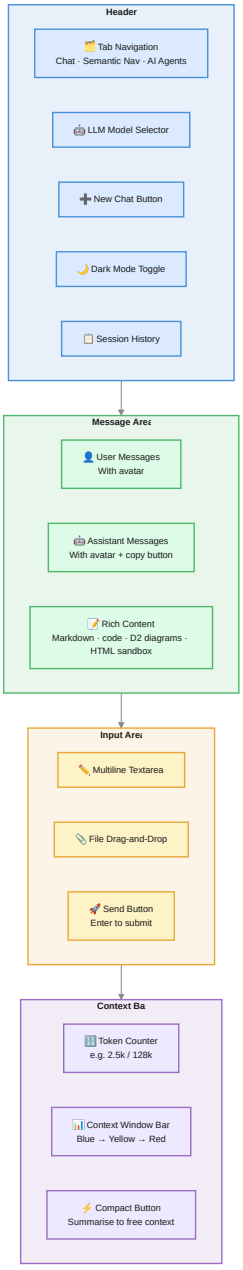
PAGE	DESCRIPTION
LLM Instances	Configure the LLM providers available as agent backends
Tool Calling	Full reference of all 27 native tools
MCP Servers	Connect agents to external tools via MCP
Chat	Front-end where agents are used — AI Agents tab
GenAI & LLM Configuration	RAG architecture overview

Chat

The **Chat** interface is the primary way users interact with the AI capabilities of Turing ES. It is organized into three sections: a direct **LLM chat**, a **Semantic Navigation** chat for searching indexed sites, and dynamic **AI Agent** views — one per configured and enabled agent.

> **LLM REQUIRED**
>
> The Chat interface is only available when at least one LLM Instance is configured and enabled. See LLM Instances to set one up.

Layout



Header controls:

CONTROL	DESCRIPTION
Tab navigation	Switch between Chat, Semantic Navigation, and AI Agent views
LLM model selector	Choose which configured LLM instance to use for the session
New Chat	Start a fresh session (saves the current one to history automatically)
Dark mode toggle	Switch between light and dark themes — code highlighting adapts accordingly
Session history	Opens the sessions sidebar to browse, restore, or delete previous conversations

Context Bar — displayed below the message area:

INDICATOR	BEHAVIOUR
Token counter	Shows `current/max` tokens (e.g., `2.5k/128k`) — estimated at ~4 chars per token
Progress bar	Visual fill showing context usage; **blue** (normal), **yellow** (60%+), **red** (80%+)
Compact button	Summarises the conversation via the LLM to free up context space

Chat (Direct LLM)

A general-purpose chat with the selected LLM. This tab provides the most direct access to the underlying model, with a set of optional tools the user can enable per conversation.

File Attachments

Files can be added to the conversation via drag-and-drop or the file picker:

FILE TYPE	HOW IT'S HANDLED
Documents (PDF, DOCX, XLSX, PPTX, HTML, TXT, ...)	Text extracted via **Apache Tika** and included in the prompt as context
Images (PNG, JPEG, WebP, GIF, ...)	Sent directly as media to models with vision capability

Attached files are displayed as badges on the message they are sent with.

Streaming

Responses are streamed in real time using **Server-Sent Events (SSE)**, so content appears progressively as the model generates it — no waiting for the full response.

Available Tools

The following tools can be enabled for the Chat section. The LLM invokes them autonomously during a conversation when it determines they are needed:

TOOL	DESCRIPTION
Code Interpreter	Executes Python code in a sandboxed environment. Supports Matplotlib for charts. Timeout: 30 seconds. Generated files (e.g., charts, CSVs) are returned as download links.
Web Crawler	Fetches and extracts content from a web page. Max 12,000 characters per page, up to 30 links extracted.
Image Search	Searches for images via DuckDuckGo / Bing. Returns up to 8 results.
Weather	Returns weather forecasts for 1–7 days using Open-Meteo.
Finance	Retrieves stock quotes and historical price data via Yahoo Finance.
Date / Time	Returns the current date and time for any given timezone.
RAG Search	Searches the Knowledge Base (vector store) by semantic similarity. Also provides: knowledge base statistics, file listing with optional keyword filter, and full file content retrieval.

Semantic Navigation

A chat interface backed by the indexed content of **Semantic Navigation Sites**. Instead of querying the LLM's parametric knowledge, this section sends the user's question through site-specific search tools.

The system prompt includes locale instructions and available facets for each configured site.

Tools available:

TOOL	DESCRIPTION
`list_sites`	Lists all available SN Sites and their locales
`get_site_fields`	Returns available fields and facets for a specific site
`get_valid_filter_values`	Returns valid values for a filter or facet field
`search_site`	Performs a semantic search within a site and returns results

Any **MCP Servers** configured in Administration are also available in this tab, extending the tool set with external capabilities.

> **TIP**
> Use this tab when you want answers grounded exclusively in your indexed enterprise content, rather than the LLM's general knowledge.

AI Agents

Each **AI Agent** configured and enabled in **Administration → AI Agents** appears as its own view in the Chat interface. Agents are completely independent from each other — each has its own LLM, system prompt, tool set, and visual identity.

PER-AGENT SETTING	DESCRIPTION
Name	The tab label and agent display name
Avatar	Image shown in the chat alongside agent messages
System Prompt	The agent's persona, purpose, and behavioural instructions
LLM Instance	The specific language model powering this agent (must be valid and enabled)
Native Tools	A selection from the 27 native tools (code interpreter, search, weather, finance, etc.)
MCP Servers	External tool servers connected specifically to this agent

For full configuration details — composing agents, tool selection, and MCP Server registration — see AI Agents.

Rich Content Rendering

Chat responses are rendered with full media-type awareness:

CONTENT TYPE	RENDERING
Markdown	Full GitHub Flavored Markdown — tables, strikethrough, task lists, inline code, blockquotes
Code blocks	Syntax highlighting via highlight.js with automatic light/dark theme switching
D2 diagrams	Rendered to SVG via WASM; falls back to a dev server in development mode
HTML	Sandboxed preview in an isolated iframe — toggle between rendered view and source, with fullscreen option
Generated files	Files created by the Code Interpreter (charts, processed data, etc.) are shown as download links inline in the response

Session History

Chat sessions are stored locally in the browser's **IndexedDB** — they are not sent to the server. This means:

- Sessions are **per browser and per device** — clearing browser data removes them
- No user authentication is required to access past sessions
- Session data never leaves the user's machine

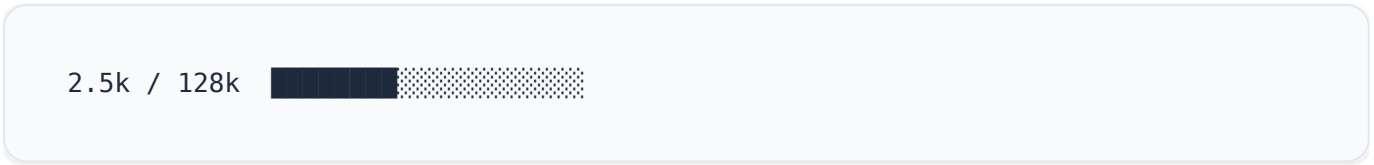
Session sidebar features:

FEATURE	DESCRIPTION
Auto-title	A short title is generated by the LLM from the first exchange; falls back to the first message text if generation fails
Model badge	Shows which LLM model was used for the session
Message count	Number of messages in the session
Timestamp	Date and time of the last message
Restore	Click to resume a previous session
Delete	Remove a session from history

Sessions are saved automatically after each complete response.

Context Window Management

A context bar at the bottom of the chat input area shows token usage in real time.



The bar displays the **token count** (e.g., `2.5k/128k`) alongside a progress bar. Tokens are estimated client-side at **~4 characters per token** (`Math.ceil(text.length / 4)`), counting the full message history.

Context window size

The context window size is resolved with a three-tier fallback:

Backend API (highest priority) — calls `GET /v2/llm/{llmInstanceId}/chat/context-info` and caches the result

LLM Instance configuration — the `contextWindow` property set on the instance

Default — 128,000 tokens if neither of the above is available

Progress bar colours

USAGE	COLOUR	MEANING
Below 60%	**Blue**	Normal — plenty of context remaining
60% - 79%	**Yellow**	Warning — consider compacting soon
80%+	**Red**	Critical — compact before the limit is reached

Compact

The **Compact** button (lightning bolt icon) is available when the conversation has at least **4 messages** and the model is not generating a response. Compacting:

Sends the full conversation history to the same LLM with a summarisation prompt

The LLM generates a concise summary preserving key facts, decisions, and context

The entire conversation is replaced with a single `**[Context compacted]**` block followed by the summary

The conversation continues from the compacted state with significantly reduced token usage

Compacting is available in all three chat modes: direct LLM, Semantic Navigation, and AI Agents.

> **WHEN TO COMPACT**
>
> When the bar turns **yellow** (60%+), consider compacting proactively. Don't wait until it turns **red** (80%+) — at that point the model may start losing context from earlier messages. The tooltip on the Compact button shows how much context remains (e.g., "58% of context remaining").

Files & Attachments Reference

CAPABILITY	DETAIL
Upload method	Drag-and-drop onto the chat window, or click the attachment button
Transfer format	Multipart form (files sent together with the message)
Document processing	Apache Tika extracts text from PDF, DOCX, XLSX, PPTX, HTML, TXT, and more
Image processing	Passed directly as media bytes to vision-capable models
Display	Shown as file badges on the sent message bubble

API Endpoints

The chat features are accessible programmatically via the REST API. See REST API Reference → GenAI API for endpoint details and examples.

Token Usage

If you are an administrator responsible for monitoring AI spend, this page is for you. Token Usage helps you track how much each LLM model is costing your organization by showing consumption broken down by model, day, and month — so you can identify heavy usage patterns, set budgets, and avoid unexpected bills.

The **Token Usage** page (`/token-usage`) gives administrators a clear view of how many tokens are being consumed by the LLM integrations — broken down by model, day, and month. This is the primary tool for tracking AI costs and identifying heavy usage patterns.

> **AVAILABILITY**
> This page only appears in the sidebar when **at least one LLM instance is enabled**. If no LLM is configured, the Generative AI section will not show Token Usage.

Period Selector

At the top of the page, a month navigator lets you move forward and backward through calendar months. All cards and tables update instantly when the period changes.

```
◄  January 2025  ►
```

Summary Cards

Four cards provide an at-a-glance snapshot of the selected month:

CARD	DESCRIPTION
Total Requests	Number of LLM API calls made during the period
Input Tokens	Total tokens sent to the model (prompts, context, system messages)
Output Tokens	Total tokens received from the model (generated responses)
Total Tokens	Sum of input + output tokens

Values are displayed in a human-readable format — for example `1.2M`, `45K`, or `8.3K` — rather than raw numbers.

Summary by Model

A monthly aggregation table grouping consumption by LLM instance:

COLUMN	DESCRIPTION
Instance	The LLM instance name as configured in Administration
Vendor	Provider (e.g., OpenAI, Anthropic, Google, Azure, Ollama)
Model	Specific model identifier (e.g., `gpt-4o`, `claude-3-5-sonnet`)
Requests	Total requests to this instance in the period
Input	Total input tokens for this instance
Output	Total output tokens for this instance
Total	Combined token count for this instance

Use this table to compare token consumption across different providers and models, and to identify which instances account for the largest share of usage.

Daily Breakdown

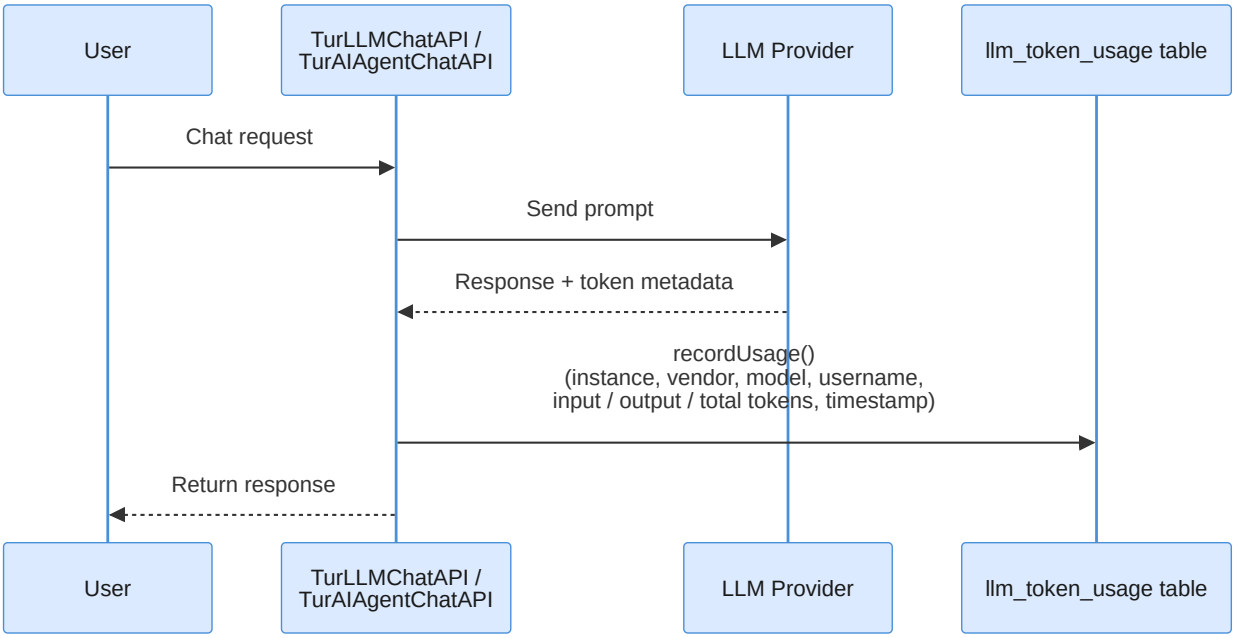
A day-by-day table showing consumption per model across the selected month:

COLUMN	DESCRIPTION
Date	Calendar day
Instance	LLM instance name
Vendor	Provider
Model	Model identifier
Input Tokens	Tokens sent that day
Output Tokens	Tokens received that day
Total Tokens	Combined token count for that day
Requests	Number of requests on that day

Days with no LLM activity do not appear in the table. This breakdown helps spot daily spikes — for example, a batch re-indexing job or a high-traffic event.

How Token Recording Works

Turing ES records token usage automatically on every LLM call — no extra configuration is needed.



What gets recorded on each call:

FIELD	DESCRIPTION
Instance	Which LLM instance handled the request
Vendor	Provider name
Model	Model identifier from the response metadata
Username	The authenticated user who triggered the call
Input Tokens	Extracted from the LLM response metadata
Output Tokens	Extracted from the LLM response metadata
Total Tokens	Input + output
Timestamp	Date and time of the request

> **NOTE**
>
> Responses where all token counts are zero are **not recorded**. This filters out failed or incomplete LLM calls that would skew usage statistics.

API Endpoint

Token usage data is also available via the REST API. See REST API Reference → Token Usage API for endpoint details and examples.

Developer Guide

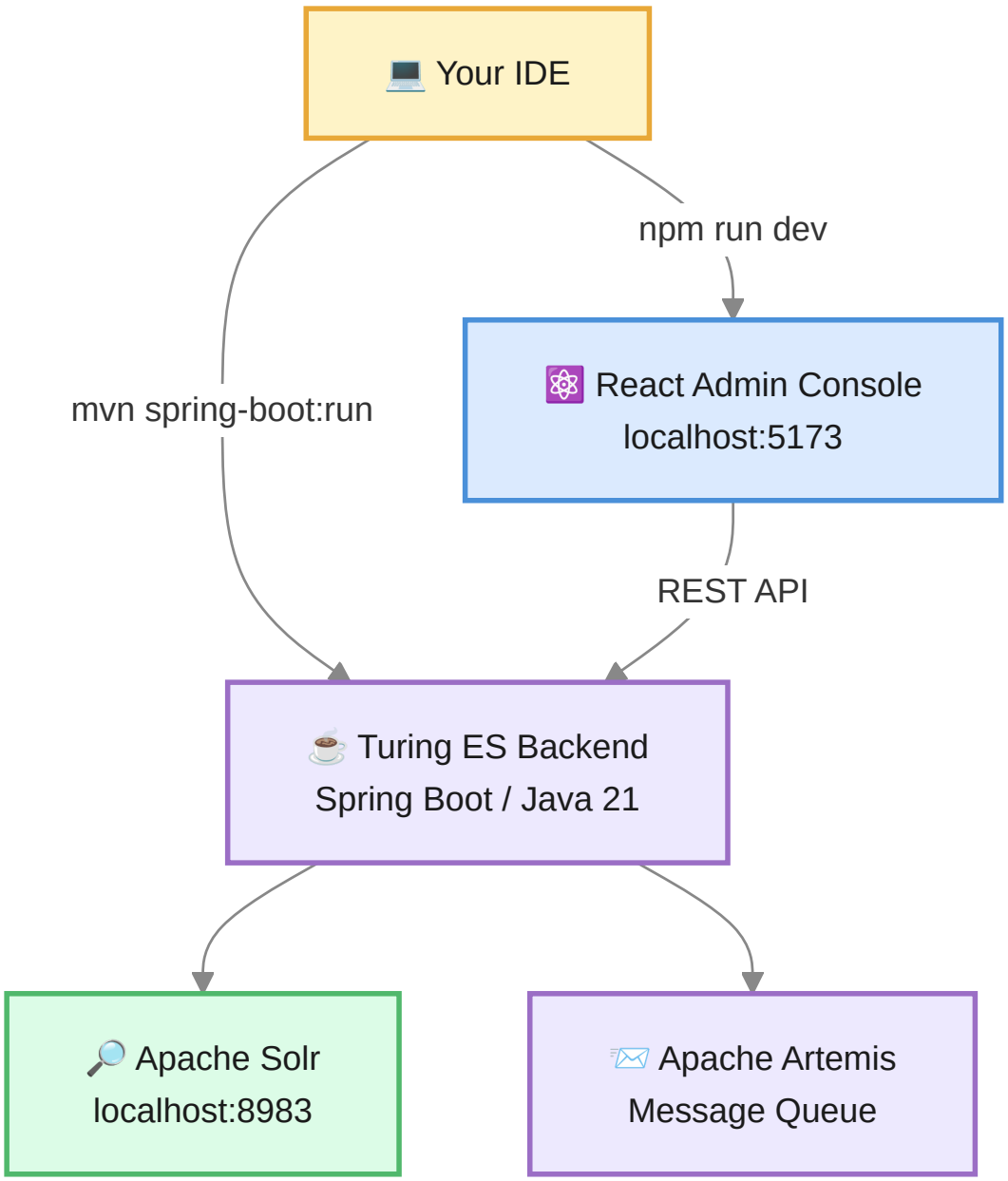
Whether you're **building a search experience** on top of Turing ES or **contributing to the project itself**, this guide has everything you need to get up and running quickly.

Turing ES is a fully open-source enterprise search platform with semantic navigation and GenAI capabilities. The source code lives at github.com/openviglet/turing and all contributions are welcome.

Tech Stack

Understanding the stack helps you navigate the codebase and decide where to plug in.

LAYER	TECHNOLOGY
Backend	Java 21 · Spring Boot · Spring AI
Search Engine	Apache Solr (primary) · Elasticsearch · Lucene
Message Queue	Apache Artemis
Database	H2 (dev) · PostgreSQL / MySQL (prod)
Frontend	React · TypeScript · shadcn/ui · Vite
AI / GenAI	Spring AI · ChromaDB · PgVector · Milvus
Build	Maven (backend) · npm (frontend)
CI/CD	GitHub Actions



Setting Up Your Dev Environment

Prerequisites

Before you begin, make sure you have these installed:

- Java 21 (Temurin recommended)
- Maven 3.9+
- Node.js 20+ and npm
- Git

Docker Desktop (for running Solr and other external services locally)

Clone the Repository

```
git clone https://github.com/openviglet/turing.git
cd turing
```

Start Services with Docker Compose

Apache Solr is the only external service required to run Turing ES. Apache Artemis runs **embedded** inside the application and requires no separate installation. The easiest way to start Solr locally is with Docker Compose:

```
docker-compose up -d
```

This starts:

Apache Solr at `http://localhost:8983`

> **TIP**
> Wait for Solr to be fully ready before starting the backend. You can check at `http://localhost:8983/solr/#/`.

Running Turing ES

Backend (Spring Boot)

Run the full backend including the bundled React console:

```
cd turing
mvn spring-boot:run -pl turing-app
```

Or skip the npm build step if you haven't changed the frontend:

```
mvn spring-boot:run -pl turing-app -Dskip.npm
```

The backend starts at `http://localhost:2700`.

Frontend — React Admin Console

For active frontend development, run the React dev server separately. First start the backend in headless mode:

```
mvn spring-boot:run -pl turing-app -Dskip.npm -Dspring-boot.run.profiles=dev-ui
```

Then launch the React app:

```
cd turing/turing-ui
npm install
npm run dev
```

The Vite dev server starts at `http://localhost:5173` with hot-reload enabled.

Production Build

```
cd turing
mvn clean install
mvn package -pl turing-app
```

The resulting JAR in `turing-app/target/` bundles both the backend and the compiled React assets.

Development URLs

SERVICE	URL	NOTES
Admin Console	`http://localhost:2700`	Backend-served
React Dev Server	`http://localhost:5173`	Vite hot-reload
SN Search Sample	`http://localhost:2700/sn/Sample`	
Swagger UI	`http://localhost:2700/swagger-ui.html`	Interactive API docs
OpenAPI Spec	`http://localhost:2700/v3/api-docs`	JSON spec
Solr	`http://localhost:8983`	Docker Compose

DEFAULT CREDENTIALS

On first startup, set the admin password via the environment variable `TURING_ADMIN_PASSWORD`. See the Installation Guide for details.

Java SDK

The Turing Java SDK lets you integrate semantic search into any JVM application.

Run the Sample

```
cd turing-java-sdk
mvn package
java -cp build/libs/turing-java-sdk-all.jar
com.viglet.turing.client.sn.sample.TurSNClientSample
```

Add to Your Project via Maven Central

Add to your `pom.xml`:

```
<dependency>
  <groupId>com.viglet.turing</groupId>
  <artifactId>turing-java-sdk</artifactId>
  <version>2026.1.17</version>
</dependency>
```

The SDK and `turing-commons` are published to Maven Central. No custom repository configuration is needed.

Build the SDK

```
cd turing-java-sdk
mvn package
```

Database Migrations with Liquibase

Turing ES uses **Liquibase** (v5.0.2) for database schema management. All schema changes — tables, columns, indexes, foreign keys — are tracked as versioned changelogs and applied automatically on startup. This ensures every environment (dev, staging, production) runs the exact same schema, regardless of the database platform.

How it works

When Turing ES starts, Spring Boot runs Liquibase before the application context is fully initialized. Liquibase reads the master changelog, compares it against the `DATABASECHANGELOG` tracking table, and applies any pending changesets. This means:

- **First startup** — all migrations run, creating the full schema from scratch
- **Upgrades** — only new changesets (added since the last startup) are applied
- **Rollbacks** — changesets already applied are skipped (`DATABASECHANGELOG` tracks what has run)

Configuration

Liquibase is configured in `application.yaml`:

```
spring:
  liquibase:
    change-log: classpath:db/changelog/db.changelog-master.yaml
    enabled: true
    default-schema: PUBLIC
```

Changelog structure

All changelogs are in **YAML format** under `turing-app/src/main/resources/db/changelog/`:

```
db/changelog/
├── db.changelog-master.yaml          ← master file, includes all others
├── v2025.3_baseline.yaml             ← initial schema (full baseline)
├── v2026.1_update.yaml               ← major version update
├── v2026.1_type_enum_fix.yaml        ← type/enum corrections
├── v2026.1.11.yaml                   ← incremental patch
├── v2026.1.11.1.yaml                 ← incremental patch
├── v2026.1.12.yaml                   ← incremental patch
├── v2026.1.14.0.yaml                 ← incremental patch
├── v2026.1.14.1.yaml                 ← ...
├── ...
└── v2026.1.14.8.yaml
```

The **master changelog** (`db.changelog-master.yaml`) includes all files in sequential order:

```yaml
databaseChangeLog:
  - include:
      file: db/changelog/v2025.3_baseline.yaml
  - include:
      file: db/changelog/v2026.1_update.yaml
  - include:
      file: db/changelog/v2026.1_type_enum_fix.yaml
  - include:
      file: db/changelog/v2026.1.11.yaml
  # ... incremental patches follow
```

Naming convention

PATTERN	EXAMPLE	USAGE
`v{version}_baseline.yaml`	`v2025.3_baseline.yaml`	Full schema baseline for a major version
`v{version}_update.yaml`	`v2026.1_update.yaml`	Generated diff between two versions
`v{version}.yaml`	`v2026.1.12.yaml`	Incremental patch for a specific release

Writing a new migration

When adding a new table, column, or index, create a new changelog file following the naming convention and add it to the master changelog.

Example — adding a column to an existing table:

```yaml
databaseChangeLog:
  - changeSet:
      id: v2026.1.15-01-add-priority-to-spotlight
      author: viglet-team
      preConditions:
        - onFail: MARK_RAN
        - tableExists:
            tableName: sn_site_spotlight
        - not:
            - columnExists:
                tableName: sn_site_spotlight
                columnName: priority
      changes:
        - addColumn:
            tableName: sn_site_spotlight
            columns:
              - column:
                  name: priority
                  type: INT
                  defaultValueNumeric: 0
```

Key rules:

Always use preconditions with `onFail: MARK_RAN` to make changesets idempotent — safe to re-run if the change was already applied manually

Changeset IDs use the format `v{version}-{sequence}-{description}` (e.g., `v2026.1.14-01-add-default-label-to-custom-facet`)

Authors use `viglet-team` for manual migrations or `{name} (generated)` for auto-generated diffs

Never modify an already-released changeset — always create a new one

Generating a diff changelog

The Liquibase Maven plugin can auto-generate a diff between two database versions:

```
cd turing-app
mvn liquibase:diff
```

This compares the reference database (current code) against the target database (previous version) and writes a changelog to `src/main/resources/db/changelog/v2026.1_update.yaml`. The diff tracks: tables, columns, indexes, foreign keys, and primary keys.

The plugin configuration is in `turing-app/pom.xml` and uses `liquibase.properties` for connection settings.

Database platform support

Liquibase abstracts SQL differences across platforms. The same changelogs work on all supported databases:

DATABASE	DRIVER	NOTES
H2	`org.h2.Driver`	Default for development — embedded, zero setup
MariaDB / MySQL	`org.mariadb.jdbc.Driver`	Production recommended
PostgreSQL	`org.postgresql.Driver`	Production recommended
Oracle	`oracle.jdbc.OracleDriver`	Enterprise environments
SQL Server	`com.microsoft.sqlserver.jdbc.SQLServerDriver`	Enterprise environments

Liquibase handles type mappings (e.g., `VARCHAR(255)` on MySQL vs `VARCHAR2(255)` on Oracle) transparently.

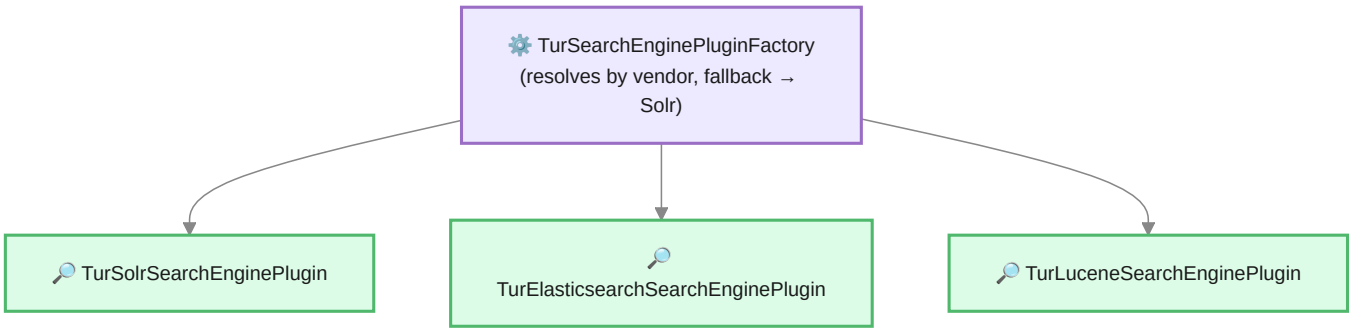
Code Quality

Turing ES maintains high code quality standards. You can check the project health at any time:

TOOL	LINK
SonarCloud	sonarcloud.io/organizations/viglet-turing
GitHub Actions	openviglet/turing/actions
GitHub Security	openviglet/turing/security
Codecov	app.codecov.io/gh/openviglet/turing

Search Engine Plugin Architecture

Turing ES uses a **plugin architecture** to support multiple search backends behind a unified interface. This abstraction allows the same application code to work with Apache Solr, Elasticsearch, or Lucene — the active plugin is resolved at runtime based on the vendor configured per Search Engine instance.



The `TurSearchEnginePlugin` interface

All backend operations are defined in a single interface with four categories:

Search

METHOD	DESCRIPTION
`retrieveSearchResults()`	Execute a full-text query and return results with facets
`retrieveFacetResults()`	Return facet counts for a query

Index management

METHOD	DESCRIPTION
`createIndex()`	Create a new core / index
`deleteIndex()`	Delete a core / index
`clearIndex()`	Remove all documents from a core without deleting it
`listIndexes()`	List all cores / indices on the backend

Schema management

METHOD	DESCRIPTION
`addOrUpdateField()`	Add or update a field definition in the schema
`deleteField()`	Remove a field from the schema
`fieldExists()`	Check whether a field exists in the schema

Documents & monitoring

METHOD	DESCRIPTION
`indexDocument()`	Index a single document
`deIndex()`	Remove a document from the index
`commit()`	Flush pending writes (for backends that require explicit commits)
`getDocumentTotal()`	Return the total number of indexed documents
`getSystemInfo()`	Return backend version, OS, JVM, and memory information

Implementing a new backend

To add support for a new search engine:

1. Create a class implementing `TurSearchEnginePlugin`
2. Implement all methods in the four categories above
3. Register the plugin in `TurSearchEnginePluginFactory` with a vendor identifier
4. The factory resolves the correct plugin based on the `vendor` field of the `TurSEInstance` entity

> **FALLBACK BEHAVIOUR**
>
> If a vendor is unrecognised or unavailable, `TurSearchEnginePluginFactory` falls back to the **Solr** plugin. Apache Solr is the primary supported backend with the most complete feature set.

REST API

Turing ES exposes a REST API for integrating search and AI capabilities into any application. All endpoints use **JSON**. Authentication uses the `Key` header with an API Token created in **Administration → API Tokens**.

For the full endpoint reference — search, autocomplete, spell check, latest searches, GenAI chat, and token usage — see **REST API Reference**.

For interactive exploration, use the built-in Swagger UI at `http://localhost:2700/swagger-ui.html` or the OpenAPI spec at `http://localhost:2700/v3/api-docs`.

Contributing

We'd love your help making Turing ES better. Here's how to get involved:

Fork the openviglet/turing repository.

Create a branch for your feature or fix: `git checkout -b feature/my-improvement`

Commit your changes with clear, descriptive messages.

Open a Pull Request — describe what you changed and why.

For larger contributions, open an issue first to discuss the approach before writing code.

> **TIP**
> Check the open GitHub Issues for good first issues tagged with `good first issue` or `help wanted`.

REST API Reference

Turing ES exposes a REST API for integrating search and AI capabilities into any application. All endpoints use **JSON**. Public endpoints (search, chat, autocomplete) require no authentication. Administrative endpoints require a `Key` header.

For authentication details, see Authentication. For the interactive API explorer, visit `http://localhost:2700/swagger-ui.html`.

Authentication

Protected endpoints require an **API Token** in the `Key` request header:

```
Key: <YOUR_API_TOKEN>
```

Create tokens in **Administration → API Tokens**. See Authentication for step-by-step instructions.

Example — authenticated API call:

```
curl "http://localhost:2700/api/sn/Sample/search?q=cloud&_setlocale=en_US" \
  -H "Key: <YOUR_API_TOKEN>" \
  -H "Accept: application/json"
```

Public Endpoints (no authentication required)

Certain endpoints are publicly accessible, allowing client applications to perform searches and chat without managing sessions:

ENDPOINT	PURPOSE
`GET /api/sn/*/search`	Semantic Navigation search
`GET /api/sn/*/chat`	GenAI chat on an SN Site
`GET /api/sn/*/ac`	Autocomplete
`POST /api/genai/chat`	Direct GenAI chat
`POST /api/ocr/**`	OCR text extraction
`POST /api/v2/integration/**`	External integration endpoints
`GET /api/v2/guest/**`	Guest access endpoints
`POST /graphql`	GraphQL queries
`GET /api/login`	Login endpoint

All other endpoints require authentication — including the full administration API, user management, site configuration, and AI Agent management.

OpenAPI & Swagger

Explore and test every endpoint interactively:

INTERFACE	URL
Swagger UI	`http://localhost:2700/swagger-ui.html`
OpenAPI 3.0 spec	`http://localhost:2700/v3/api-docs`

Semantic Navigation API

Search

The core search endpoint. Returns results, facets, spotlights, spell-check suggestions, and pagination.

```
GET  http://localhost:2700/api/sn/{siteName}/search
POST http://localhost:2700/api/sn/{siteName}/search
```

Query Parameters (GET):

PARAMETER	REQUIRED	DESCRIPTION
`q`	✅	Search query
`_setlocale`	✅	Locale code (e.g., `en_US`, `pt_BR`)
`p`		Page number (default: `1`)
`rows`		Results per page (overrides site default)
`sort`		Sort field and direction (e.g., `date desc`)
`fq[]`		Filter query — apply a facet filter (e.g., `fq[]=type:news`)
`group`		Group results by a field value

Example (GET):

```
curl "http://localhost:2700/api/sn/Sample/search?
q=enterprise+search&p=1&_setlocale=en_US&rows=10" \
  -H "Accept: application/json"
```

Search Response Structure

The search response is a **self-describing navigational JSON**: every link a user might follow — apply a filter, remove a filter, paginate, filter by metadata — is pre-built and ready to use. The front-end does not need to construct query strings or manage filter state.

```
{
  "pagination": [
    { "type": "CURRENT", "text": "1", "href": "...", "page": 1 },
    { "type": "NEXT",    "text": "Next", "href": "...", "page": 2 }
  ],
  "queryContext": {
    "count": 42,
    "index": "Sample",
    "limit": 10,
    "offset": 0,
    "page": 1,
    "pageCount": 5,
    "pageEnd": 10,
    "pageStart": 1,
    "responseTime": 23,
    "query": { ... },
    "defaultFields": { ... },
    "facetType": "AND",
    "facetItemType": "AND"
  },
  "results": {
    "document": [
      {
        "source": "TURING",
        "elevate": false,
        "fields": { "title": "...", "url": "...", ... },
        "metadata": [
          { "name": "type", "values": [ { "label": "article", "link": "..." }
] }
        ]
      }
    ]
  },
  "widget": {
    "facet": [ ... ],
    "secondaryFacet": [ ... ],
    "facetToRemove": { ... },
    "similar": [ ... ],
    "spellCheck": { ... },
    "spotlights": [ ... ],
    "locales": [ ... ],
    "cleanUpFacets": "...",
    "selectedFilterQueries": [ ... ]
  },
  "groups": [ ... ]
}
```

Top-level sections:

SECTION	DESCRIPTION
pagination	Pre-built page links (`FIRST` , `PREVIOUS` , `CURRENT` , `NEXT` , `LAST`) — the front-end follows them directly
queryContext	Result count, page info, response time, active facet operators, and default field mappings
results.document	Array of matched documents. Each contains `fields` (all indexed values), `metadata` (facet values with pre-built filter links), `source` , and `elevate` flag
widget.facet	Primary facet groups with counts and pre-built filter/clear links per value. Enabled by `Facet = true`
widget.secondaryFacet	Separate facet groups for fields promoted to Secondary Facet — independent from `facet` , for different UI treatment (e.g., tabs)
widget.similar	More Like This results. Enabled by `MLT = true`
widget.spellCheck	Spelling correction with `original` , `corrected` text, and `usingCorrectedText` flag. Enabled by `Spell Check = true`
widget.spotlights	Curated documents injected at configured positions when the query matches a spotlight term
widget.locales	All configured locales with pre-built links — use as valid values for `_setlocale`
widget.cleanUpFacets	Pre-built link to clear all active facet filters
groups	Grouped results when the `group` parameter is used

Self-describing navigation operates at three levels:

Facet navigation — Each facet value carries pre-built links for applying, removing, or clearing filters

Document metadata navigation — Each result's metadata values carry filter links (e.g., click a `type:article` tag to filter all articles)

Pagination — Next/previous/specific page links are included; no offset calculation needed

A Turing ES search UI can be built as a **pure rendering layer** — render results, facets, tags, pagination, and locale switchers directly from the response. Adding a new facet to the SN Site configuration propagates automatically, with no client code changes.

Locale selection:

```
GET /api/sn/{siteName}/search?q=annual+report&_setlocale=pt_BR
```

POST Body Parameters

The POST endpoint accepts the same query parameters plus additional fields in the JSON body:

FIELD	TYPE	DESCRIPTION
`userId`	`string`	User identifier (for metrics and latest searches)
`query`	`string`	Search query (alternative to `q` query param)
`locale`	`string`	Locale code (alternative to `_setlocale` query param)
`page`	`integer`	Page number
`rows`	`integer`	Results per page
`sort`	`string`	Sort field and direction
`group`	`string`	Group results by field
`fq`	`string[]`	Filter queries
`fqAnd`	`string[]`	AND filter queries
`fqOr`	`string[]`	OR filter queries
`fqOperator`	`string`	Filter query operator between facets
`fqItemOperator`	`string`	Filter query operator within facet values
`fieldList`	`string[]`	Restrict which fields are returned
`disableAutoComplete`	`boolean`	Disable autocomplete suggestions (default: `false`)
`populateMetrics`	`boolean`	Enable search metrics recording (default: `true`)
`targetingRules`	`string[]`	Targeting rules — see Targeting Rules

FIELD	TYPE	DESCRIPTION
`targetingRulesWithCondition`	`map`	Targeting rules with conditions
`targetingRulesWithConditionAND`	`map`	Targeting rules with AND conditions
`targetingRulesWithConditionOR`	`map`	Targeting rules with OR conditions

Example (POST with targeting rules):

```
curl -X POST "http://localhost:2700/api/sn/Sample/search" \
  -H "Content-Type: application/json" \
  -d '{
    "query": "benefits",
    "locale": "en_US",
    "rows": 10,
    "targetingRules": ["department:HR", "department:Finance"]
  }'
```

For the full Targeting Rules reference — rule types, Solr query generation, fallback clause, and practical examples — see Semantic Navigation → Targeting Rules.

Auto Complete

Returns suggestions for the given prefix. Ideal for search-as-you-type UIs.

```
GET http://localhost:2700/api/sn/{siteName}/ac
```

PARAMETER	REQUIRED	DESCRIPTION
`q`	✅	Prefix to complete
`_setlocale`	✅	Locale
`rows`		Maximum number of suggestions

Example:

```
curl "http://localhost:2700/api/sn/Sample/ac?q=enter&_setlocale=en_US"
```

Response:

```
["enterprise", "enterprise search", "enterprise AI", "entries"]
```

Latest Searches

Returns the most recent search terms for a given user — useful for personalised search history UIs.

```
POST http://localhost:2700/api/sn/{siteName}/search/latest
```

PARAMETER	LOCATION	DESCRIPTION
`q`	Query	Current query
`rows`	Query	Max results (default: `5`)
`_setlocale`	Query	Locale
`userId`	Body (JSON)	User identifier

Example:

```
curl -X POST "http://localhost:2700/api/sn/Sample/search/latest?
q=cloud&rows=5&_setlocale=en_US" \
  -H "Key: <API_TOKEN>" \
  -H "Content-Type: application/json" \
  -d '{ "userId": "user123" }'
```

Response:

```
["cloud computing", "cloud storage", "cloud security"]
```

Search Locales

Lists all configured locales for a Semantic Navigation Site.

```
GET http://localhost:2700/api/sn/{siteName}/search/locales
```

Response:

```
[
  { "locale": "en_US", "link": "/api/sn/Sample/search?_setlocale=en_US" },
  { "locale": "pt_BR", "link": "/api/sn/Sample/search?_setlocale=pt_BR" }
]
```

Spell Check

Corrects a query against the site's indexed vocabulary.

```
GET http://localhost:2700/api/sn/{siteName}/{locale}/spell-check
```

PARAMETER	REQUIRED	DESCRIPTION
`q`	✅	Text to check

Example:

```
curl "http://localhost:2700/api/sn/Sample/en_US/spell-check?q=entirprise"
```

GenAI API

> **STREAMING RESPONSES**
>
> Both chat endpoints return **Server-Sent Events (SSE)** streams — content is delivered progressively as the model generates it. Clients should consume the response as an event stream rather than waiting for a single JSON payload.

RAG Chat (SN Site)

Sends a question to the GenAI engine of a Semantic Navigation Site. Returns a streaming SSE response grounded in the site's indexed content.

```
GET http://localhost:2700/api/sn/{siteName}/chat?q={question}
```

Example:

```
curl "http://localhost:2700/api/sn/Sample/chat?q=What+are+the+main+features?"
\
  -H "Key: <YOUR_API_TOKEN>"
```

AI Agent Chat

Sends a message to a specific AI Agent. Returns a streaming SSE response.

```
POST http://localhost:2700/api/v2/llm/agent/{agentId}/chat
```

Example:

```
curl -X POST "http://localhost:2700/api/v2/llm/agent/my-agent/chat" \
  -H "Key: <YOUR_API_TOKEN>" \
  -H "Content-Type: application/json" \
  -d '{ "message": "Summarise the latest quarterly report" }'
```

See Chat for the full chat interface documentation.

Token Usage API

Returns token consumption statistics for a given month.

```
GET http://localhost:2700/api/v2/llm/token-usage?month=YYYY-MM
```

Example:

```
curl "http://localhost:2700/api/v2/llm/token-usage?month=2025-01" \
  -H "Key: <YOUR_API_TOKEN>"
```

Authentication required. See Token Usage for details.

Integration API

The Integration API provides a **reverse-proxy** endpoint that forwards requests to a configured external integration instance (e.g., an AEM connector or Web Crawler). All HTTP methods are supported. The proxy validates the target host and path to prevent SSRF attacks.

Proxy Endpoint

```
GET|POST|PUT|DELETE
http://localhost:2700/api/v2/integration/{integrationId}/**
```

The `{integrationId}` maps to an Integration Instance configured in Administration. The remainder of the path (`**`) is forwarded to the instance's endpoint URL.

Example:

```
curl -X POST "http://localhost:2700/api/v2/integration/my-aem-instance/index"
\
  -H "Content-Type: application/json" \
  -d '{ "url": "https://example.com/content/page.html" }'
```

This endpoint is **public** (no authentication required).

Integration Instance CRUD (Admin)

Manage integration instances. Authentication required.

METHOD	ENDPOINT	DESCRIPTION
`GET`	`/api/integration`	List all integration instances
`GET`	`/api/integration/{id}`	Get an integration instance
`POST`	`/api/integration`	Create an integration instance
`PUT`	`/api/integration/{id}`	Update an integration instance
`DELETE`	`/api/integration/{id}`	Delete an integration instance
`GET`	`/api/integration/vendor`	List available integration vendors

Administration API

The following endpoints manage SN Sites, fields, and spotlights. All require authentication via the `Key` header. For full details, explore the Swagger UI at `http://localhost:2700/swagger-ui.html`.

SN Site Management

METHOD	ENDPOINT	DESCRIPTION
`GET`	`/api/sn`	List all SN Sites
`GET`	`/api/sn/{id}`	Get an SN Site
`POST`	`/api/sn`	Create an SN Site
`PUT`	`/api/sn/{id}`	Update an SN Site
`DELETE`	`/api/sn/{id}`	Delete an SN Site (removes index cores)
`GET`	`/api/sn/{id}/monitoring`	Site monitoring status (document count, queue size)
`GET`	`/api/sn/{id}/export`	Export an SN Site configuration
`GET`	`/api/sn/export`	Export all SN Site configurations

Example — list sites:

```
curl "http://localhost:2700/api/sn" \
  -H "Key: <YOUR_API_TOKEN>"
```

Field Management

METHOD	ENDPOINT	DESCRIPTION
GET	/api/sn/{snSiteId}/field	List fields for a site
GET	/api/sn/{snSiteId}/field/{id}	Get a field
POST	/api/sn/{snSiteId}/field	Create a field
PUT	/api/sn/{snSiteId}/field/{id}	Update a field
DELETE	/api/sn/{snSiteId}/field/{id}	Delete a field

Spotlight Management

METHOD	ENDPOINT	DESCRIPTION
GET	/api/sn/{snSiteId}/spotlight	List spotlights for a site
GET	/api/sn/{snSiteId}/spotlight/{id}	Get a spotlight
POST	/api/sn/{snSiteId}/spotlight	Create a spotlight
PUT	/api/sn/{snSiteId}/spotlight/{id}	Update a spotlight
DELETE	/api/sn/{snSiteId}/spotlight/{id}	Delete a spotlight

GraphQL API

Turing exposes a GraphQL endpoint that provides the same Semantic Navigation search capabilities as the REST search API but using GraphQL queries. The endpoint is **public** (no authentication required).

```
POST http://localhost:2700/graphql
```

Available Queries

QUERY	DESCRIPTION
`siteNames`	Returns all configured SN Site names
`siteSearch(siteName, searchParams, locale)`	Performs a search against an SN Site

The `SearchParamsInput` accepts: `q`, `rows`, `p`, `sort`, `group`, `fq`, `fqAnd`, `fqOr`, `fqOp`, `fqiOp`, `locale`, and `fl` (field list).

Example:

```
curl -X POST "http://localhost:2700/graphql" \
  -H "Content-Type: application/json" \
  -d '{
    "query": "query { siteSearch(siteName: SAMPLE, searchParams: { q:
\"enterprise search\", rows: 10 }, locale: \"en_US\") { queryContext { count }
results { document { fields { title url } } } } }"
  }'
```

The response mirrors the REST search structure (pagination, queryContext, results, widget, groups) — see Search Response Structure above.

OCR API

Extracts text from documents via OCR. Two endpoints are available — one for file uploads and one for URLs. Both are **public** (no authentication required).

Extract Text from File

```
POST http://localhost:2700/api/ocr/file
```

Accepts a `multipart/form-data` request with a `file` parameter.

Example:

```
curl -X POST "http://localhost:2700/api/ocr/file" \
  -F "file=@/path/to/document.pdf"
```

Extract Text from URL

```
POST http://localhost:2700/api/ocr/url
```

Accepts a JSON body with a `url` field pointing to a remote document.

Example:

```
curl -X POST "http://localhost:2700/api/ocr/url" \
  -H "Content-Type: application/json" \
  -d '{ "url": "https://example.com/report.pdf" }'
```

Both endpoints return a `TurFileAttributes` JSON object containing the extracted text and file metadata.

Related Pages

PAGE	DESCRIPTION
Authentication	How to create and use API Tokens
Semantic Navigation	SN Site configuration and the search response structure
Chat	Front-end chat interface and GenAI API
Token Usage	Token consumption reporting

Authentication

Turing ES supports two authentication modes. **Native authentication** is the default and requires no external dependencies. **Keycloak OAuth2 / OpenID Connect** is the recommended mode for production SSO environments.

This page covers **native authentication** — the session-based admin console login and the API Key mechanism for REST API access. For the full Keycloak production setup, see Security & Keycloak.

Native Authentication (default)

When Keycloak is not enabled, Turing ES uses its own user store:

Admin Console

Users log in via a form at `http://localhost:2700/login` and receive a **Java HTTP session**. Sessions are maintained server-side and expire after inactivity. No special configuration is required — this mode works out of the box.

Users, groups, and roles are managed in **Administration → Users**, **Groups**, and **Roles**.

REST API — API Key

All REST API requests authenticate via an **API Key** passed in the `Key` request header. API Tokens are not tied to user sessions — they are created in **Administration → API Tokens** and remain valid until explicitly deleted.

Creating an API Token:

1. Sign in to the Administration Console.
2. Navigate to **Administration → API Tokens**.
3. Click **New**, fill in a name and description.
4. Copy the generated token immediately — it will not be shown again.

For the full endpoint reference, authentication examples, and the list of public endpoints that require no authentication, see REST API Reference → Authentication.

TOKENS ARE ENCRYPTED AT REST

API Token values are encrypted in the database using `TurSecretCryptoService`. The encryption key is set via `turing.ai.crypto.key` in `application.yaml`. Change this key in production — see Configuration Reference.

Switching to Keycloak SSO

For production environments with SSO requirements, Turing ES can delegate authentication to Keycloak via OAuth2 / OpenID Connect. Enable it with:

```
-Dturing.keycloak=true
```

See Security & Keycloak for the full 6-step production setup — database, Keycloak installation, realm configuration, SSL certificates, JVM properties, and Apache reverse proxy.

Related Pages

PAGE	DESCRIPTION
Security & Keycloak	Full production setup with Keycloak OAuth2/OIDC
Administration Guide	Managing users, groups, roles, and API tokens
REST API	Full REST API reference including authentication examples
Configuration Reference	`turing.ai.crypto.key` and other security-related settings

Security & Keycloak

This page covers the full **Keycloak OAuth2 / OpenID Connect** production setup for Turing ES, enabling SSO integration with corporate identity providers and centralized user management.

For day-to-day authentication — native session-based login and REST API Key usage — see **Authentication**.

Why Use Keycloak with Turing ES?

The primary reason to integrate Keycloak is to connect Turing ES to your organization's **existing Single Sign-On (SSO) infrastructure**. Instead of managing a separate set of credentials just for Turing ES, users sign in with the same corporate identity they already use for email, cloud tools, and internal applications — no additional password, no extra login step.

Keycloak acts as an **identity broker** between Turing ES and your identity provider. Turing ES never sees or stores user passwords — it receives a signed JWT token from Keycloak confirming who the user is and what roles they have.

Supported Identity Providers

Because Keycloak supports the **SAML 2.0**, **OAuth 2.0**, and **OpenID Connect** standards, it can federate identity from virtually any enterprise identity provider:

IDENTITY PROVIDER	PROTOCOL	USE CASE
Microsoft Entra ID (Azure AD)	SAML 2.0 / OIDC	Microsoft 365, Azure, corporate Windows environments
Google Workspace	OIDC	Organizations on Google Workspace / Gmail
Okta	SAML 2.0 / OIDC	Multi-cloud enterprises, workforce identity
Auth0	OIDC	Developer-focused platforms, B2C applications
Ping Identity	SAML 2.0 / OIDC	Large enterprises, government, regulated industries
OneLogin	SAML 2.0 / OIDC	HR-integrated identity management
ADFS (Active Directory Federation Services)	SAML 2.0	On-premise Windows Server / Active Directory
LDAP / Active Directory	LDAP	Direct directory sync without SAML — Keycloak federates LDAP users natively
GitHub / GitLab	OAuth 2.0	Developer-focused organizations
Any SAML 2.0 IdP	SAML 2.0	Shibboleth, SimpleSAMLphp, custom SAML providers
Any OIDC provider	OIDC	Any standards-compliant OpenID Connect provider

What This Enables

Single Sign-On (SSO): Users click "Sign in" on Turing ES, are redirected to their corporate login (e.g., Microsoft 365 login page), authenticate once, and are immediately signed in to Turing ES. If they're already signed in to another corporate application, they skip the login form entirely — the SSO session is shared.

Single Sign-Out: Logging out of Turing ES triggers a Keycloak OIDC logout that propagates to the corporate IdP. The user is signed out of Turing ES and optionally out of all applications sharing the same SSO session.

Centralized User Management: User accounts, groups, and roles are managed in Keycloak (or synchronized from the corporate directory). No need to create or maintain users in Turing ES manually — Keycloak provides the identity, roles, and group memberships via JWT claims.

Multi-Factor Authentication (MFA): Keycloak supports TOTP (Google Authenticator, Microsoft Authenticator), WebAuthn/FIDO2 (hardware security keys, biometrics), SMS OTP, and email verification. MFA policies are configured in Keycloak and enforced transparently — Turing ES requires no changes.

Role-Based Access Control (RBAC): Keycloak roles and realm roles are mapped to Turing ES permissions. Define roles like `turing-admin`, `turing-editor`, `turing-viewer` in Keycloak, assign them to users or groups, and Turing ES enforces access accordingly.

Social Login: For public-facing or partner-facing deployments, Keycloak can enable login via Google, GitHub, Facebook, Apple, Twitter, and other social providers alongside or instead of corporate identity.

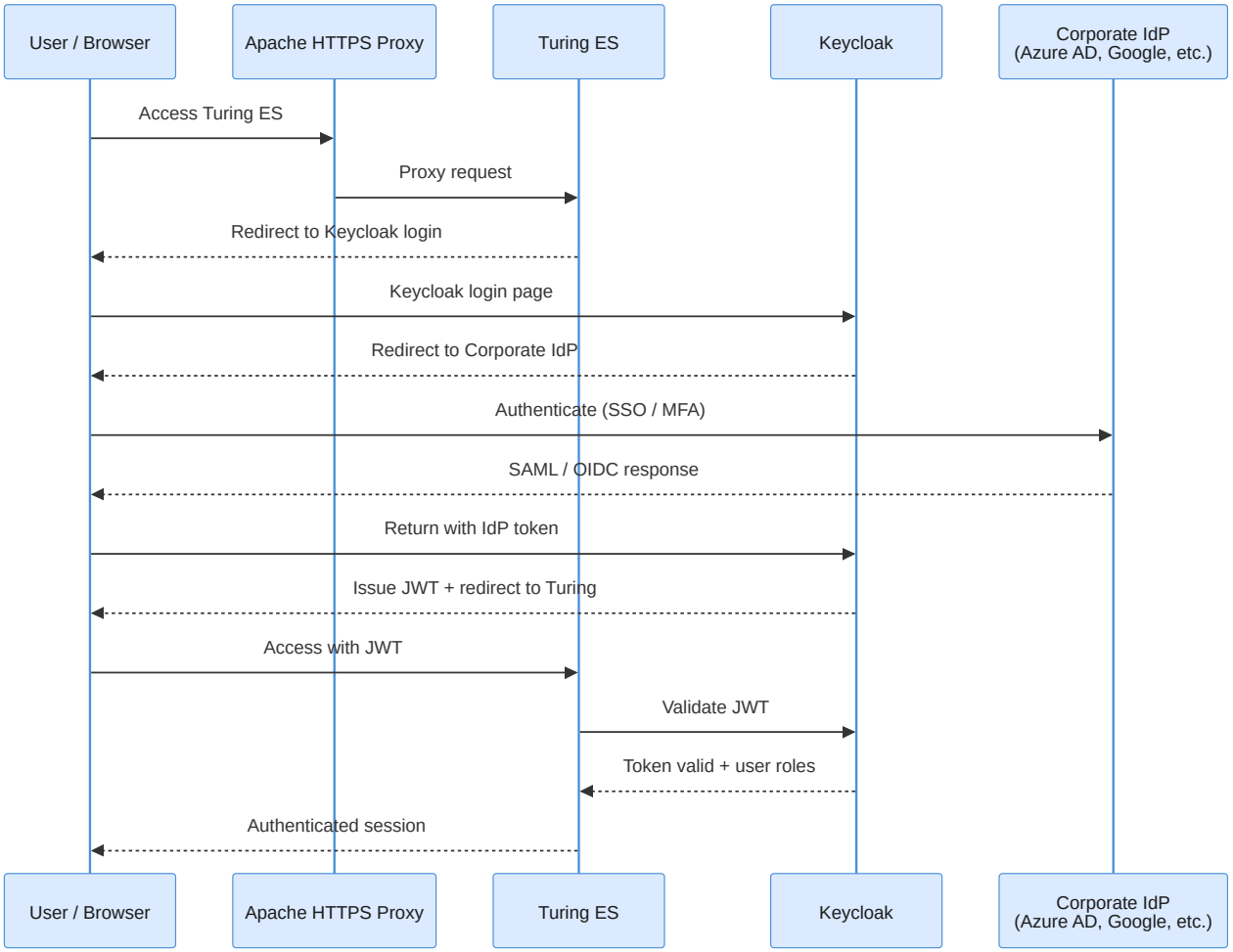
User Federation: Keycloak can synchronize users from **LDAP** or **Active Directory** without requiring SAML. Users are imported (or federated on-demand) from the directory, with password validation delegated to the directory server. This is the simplest path for organizations running on-premise AD without ADFS.

Token-Based API Access: When Keycloak is enabled, REST API clients can authenticate using **OAuth2 access tokens** in addition to API Keys. Obtain a token via Keycloak's token endpoint and pass it as a Bearer token — useful for service-to-service integrations where API Keys are not appropriate.

Audit and Compliance: Keycloak logs all authentication events — successful logins, failed attempts, token refreshes, logouts. These events can be exported to SIEM systems for compliance and security monitoring.

Architecture

When `turing.keycloak=true` is set in the JVM properties, Turing ES delegates all authentication to Keycloak using the Authorization Code flow (OAuth2) and validates access tokens as JWTs (OpenID Connect). Keycloak can be dedicated to Turing ES or shared with other applications — in both cases, Turing ES registers as a **client** within a Keycloak **realm**.


User / Browser
Apache HTTPS Proxy
Turing ES
Keycloak
Corporate IdP
(Azure AD, Google, etc.)
Access Turing ES
Proxy request
Redirect to Keycloak login
Keycloak login page
Redirect to Corporate IdP
Authenticate (SSO / MFA)
SAML / OIDC response
Return with IdP token
Issue JWT + redirect to Turing
Access with JWT
Validate JWT
Token valid + user roles
Authenticated session
User / Browser
Apache HTTPS Proxy
Turing ES
Keycloak
Corporate IdP
(Azure AD, Google, etc.)

Full Production Setup: Turing + Keycloak + Apache HTTP

The recommended production topology runs Keycloak and Turing ES on the same host, with Apache HTTP Server as the HTTPS reverse proxy in front. All three services are reachable through a single public hostname and port (443), with Apache routing by path prefix.

```
Browser / Client
      │  HTTPS :443
      ▼
Apache HTTP Server (reverse proxy)
  ├── /kc/   → Keycloak   (https://localhost:8443)
  ├── /solr/ → Solr       (http://localhost:8983)
  └── /      → Turing ES  (https://localhost:2700)
```

Step 1 — Database Setup

Both Turing ES and Keycloak require their own databases. Create them before starting either service.

```
-- Turing ES database and user
CREATE USER 'turing'@'%' IDENTIFIED BY '<turing-db-password>';
CREATE DATABASE IF NOT EXISTS turing;
GRANT ALL PRIVILEGES ON turing.* TO 'turing'@'%' WITH GRANT OPTION;

-- Keycloak database and user
CREATE USER 'keycloak'@'%' IDENTIFIED BY '<keycloak-db-password>';
CREATE DATABASE IF NOT EXISTS keycloak;
GRANT ALL PRIVILEGES ON keycloak.* TO 'keycloak'@'%' WITH GRANT OPTION;
```

Tested with MariaDB and MySQL. When using MariaDB, the JDBC connection URL should include `allowPublicKeyRetrieval=true&useSSL=false` if SSL is not configured between the application and the database server.

USE STRONG, UNIQUE PASSWORDS

Replace all placeholder passwords (`<turing-db-password>`, `<keycloak-db-password>`) with strong, randomly generated values. Never reuse passwords between services.

Step 2 — Keycloak Installation and Build

Download and extract the Keycloak distribution. Before starting Keycloak for the first time, run the build step to configure the HTTP relative path. This is required when Keycloak runs behind a reverse proxy at a path prefix (e.g., `/kc/`).

```
./kc.sh build --http-relative-path=/kc
```

Generate a keystore for Keycloak's HTTPS listener:

```
keytool -genkeypair \
  -alias localhost \
  -keyalg RSA \
  -keysize 2048 \
  -validity 365 \
  -keystore server.keystore \
  -dname "CN=<your-hostname>, O=<your-org>, C=<country-code>" \
  -keypass <keystore-password> \
  -storepass <keystore-password>
```

Start Keycloak in production mode:

```
./kc.sh start \
  --optimized \
  --hostname-strict=false \
  --https-key-store-password=<keystore-password>
```

Keycloak will be available at `https://localhost:8443/kc/`.

Running Keycloak as a Linux service

Create a systemd unit file at `/etc/systemd/system/keycloak.service`:

```
[Unit]
Description=Keycloak
After=syslog.target network.target

[Service]
User=viglet
EnvironmentFile=/appl/viglet/keycloak/env
ExecStart=/appl/viglet/keycloak/bin/kc.sh start --optimized --hostname-
strict=false --https-key-store-password=<keystore-password>

[Install]
WantedBy=multi-user.target
```

Enable and start the service:

```
systemctl daemon-reload
systemctl enable keycloak
systemctl start keycloak
```

Step 3 — Keycloak Realm and Client Configuration

After starting Keycloak, open the admin console and perform the following setup:

Create a Realm

A realm is an isolated namespace for users, clients, and roles. Create a dedicated realm for Turing ES (e.g., `turing` or a name matching your organization). All subsequent configuration is done within this realm.

Create a Client

Within the realm, create a new client for Turing ES:

FIELD	VALUE
Client ID	`turing-app` (or your preferred identifier)
Client Protocol	`openid-connect`
Access Type	`confidential`
Standard Flow	Enabled
Direct Access Grants	Disabled (not needed)
Root URL	`https://<your-hostname>`
Valid Redirect URIs	`https://<your-hostname>/login/oauth2/code/turing-app`
Web Origins	`https://<your-hostname>`

After saving, go to the **Credentials** tab and copy the generated **Client Secret**.

Configure Scopes

Ensure the client requests the following scopes: `openid`, `profile`, `roles`. These are needed for Turing ES to receive the user's identity and role information from the JWT.

Step 4 — SSL Certificate for Turing ES

Turing ES can terminate HTTPS directly via its embedded Tomcat server. Generate a self-signed certificate (replace with a CA-signed certificate for production):

```
# Generate self-signed certificate and key
openssl req -x509 -nodes -newkey rsa:2048 \
  -keyout <hostname>.key \
  -out <hostname>.crt \
  -days 365

# Package as PKCS12 for Java
openssl pkcs12 -export \
  -out turing.p12 \
  -inkey <hostname>.key \
  -in <hostname>.crt \
  -name turing

# Alternatively, generate directly with keytool
keytool -genkey \
  -alias turing \
  -storetype PKCS12 \
  -keyalg RSA \
  -keysize 2048 \
  -keystore turing.p12 \
  -validity 3650
```

Place `turing.p12` in a secure path on the server (e.g., `/appl/viglet/turing/server/cert/`).

Import the Apache certificate into the Java truststore

Turing ES calls Keycloak over HTTPS to validate tokens. If Keycloak's certificate is self-signed, Turing ES will fail the SSL handshake unless you import the Apache (or Keycloak) certificate into the Java truststore:

```
keytool -import \
  -trustcacerts \
  -alias turing-apache \
  -file /path/to/<hostname>.crt \
  -keystore $JAVA_HOME/lib/security/cacerts
```

The default truststore password is `changeit`.

Step 5 — Turing ES JVM Configuration

All Turing ES configuration is passed via JVM system properties at startup. Set `JAVA_OPTS` before launching the JAR:

```
JAVA_OPTS="\
  -Xmx1g -Xms1g \
  \
  -Dspring.datasource.url=jdbc:mariadb://localhost:3306/turing?
allowPublicKeyRetrieval=true&useSSL=false \
  -Dspring.datasource.username=turing \
  -Dspring.datasource.password=<turing-db-password> \
  -Dspring.datasource.driver-class-name=org.mariadb.jdbc.Driver \
  -
Dspring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDB103Diale
ct \
  \
  -Dturing.keycloak=true \
  -Dspring.security.oauth2.client.registration.keycloak.client-id=turing-app \
  -Dspring.security.oauth2.client.registration.keycloak.client-secret=<client-
secret> \
  -Dspring.security.oauth2.client.registration.keycloak.scope=openid \
  -Dspring.security.oauth2.client.registration.keycloak.authorization-grant-
type=authorization_code \
  -Dspring.security.oauth2.client.registration.keycloak.redirect-
uri=https://<your-hostname>/login/oauth2/code/turing-app \
  -Dspring.security.oauth2.client.provider.keycloak.issuer-uri=https://<your-
hostname>/kc/realms/<your-realm> \
  -Dspring.security.oauth2.resourceserver.jwt.issuer-uri=https://<your-
hostname>/kc/realms/<your-realm> \
  \
  -Dserver.ssl.enabled=true \
  -Dserver.ssl.key-store=/appl/viglet/turing/server/cert/turing.p12 \
  -Dserver.ssl.key-store-password=<keystore-password> \
  -Dserver.ssl.keyStoreType=PKCS12 \
  -Dserver.ssl.keyAlias=turing \
  \
  -Dturing.url=https://<your-hostname>"

java $JAVA_OPTS -jar viglet-turing.jar
```

JVM property reference

PROPERTY	DESCRIPTION
`turing.keycloak`	Set to `true` to enable Keycloak authentication. Default: `false` (native session + API Key)
`spring.security.oauth2.client.registration.keycloak.client-id`	Client ID registered in the Keycloak realm
`spring.security.oauth2.client.registration.keycloak.client-secret`	Client secret from the Keycloak Credentials tab
`spring.security.oauth2.client.registration.keycloak.scope`	OAuth2 scopes to request. Minimum: `openid`
`spring.security.oauth2.client.registration.keycloak.authorization-grant-type`	Must be `authorization_code`
`spring.security.oauth2.client.registration.keycloak.redirect-uri`	Must match the redirect URI registered in Keycloak. Pattern: `https://<your-hostname>/login/oauth2/code/<client-id>`
`spring.security.oauth2.client.provider.keycloak.issuer-uri`	Keycloak realm issuer URI: `https://<hostname>/kc/realms/<realm-name>`
`spring.security.oauth2.resourceserver.jwt.issuer-uri`	Same as issuer-uri; used for JWT validation
`server.ssl.enabled`	Enables HTTPS on the embedded Tomcat server
`server.ssl.key-store`	Path to the PKCS12 keystore file
`server.ssl.key-store-password`	Password for the keystore
`server.ssl.keyStoreType`	Must be `PKCS12`
`server.ssl.keyAlias`	Alias of the certificate entry inside the keystore

PROPERTY	DESCRIPTION
`turing.url`	Public base URL of Turing ES (used for OAuth2 redirect URI construction and post-logout redirect)

Step 6 — Apache HTTP Server as Reverse Proxy

Build Apache HTTP Server with the required modules, or install a distribution package that includes `mod_ssl`, `mod_proxy`, and `mod_proxy_http`.

Compile from source (if needed):

```
apt install autoconf libtool-bin make libpcre* libssl-dev libnghttp2-dev
libexpat1-dev

./buildconf && ./configure \
  --prefix=/appl/apache/httpd \
  --enable-mods-shared=all \
  --enable-so \
  --enable-ssl=shared \
  --enable-http2=shared \
  --enable-proxy=shared \
  --enable-proxy-http=shared \
  --enable-proxy-ajp=shared \
  --enable-proxy-balancer=shared \
  --with-mpm=worker && \
make && make install
```

VirtualHost configuration for HTTPS reverse proxy:

```
<VirtualHost _default_:443>
    DocumentRoot "/appl/apache/httpd/htdocs"
    ServerName <your-hostname>:443

    # SSL
    SSLEngine on
    SSLCertificateFile    "/path/to/<hostname>.crt"
    SSLCertificateKeyFile "/path/to/<hostname>.key"

    # Proxy settings
    HostnameLookups Off
    UseCanonicalName Off
    ProxyPreserveHost On

    # Required headers for applications behind a proxy
    RequestHeader set X-Forwarded-Proto "https"
    RequestHeader set X-Forwarded-Port "443"

    # Allow SSL proxying to backend services with self-signed certs
    SSLProxyEngine on
    SSLProxyCheckPeerName off

    # Route /solr/ to Solr (plain HTTP, internal only)
    ProxyPass        "/solr/" "http://localhost:8983/solr/"
    ProxyPassReverse "/solr/" "http://localhost:8983/solr/"

    # Route /kc/ to Keycloak (HTTPS, internal)
    ProxyPass        "/kc/" "https://localhost:8443/kc/"
    ProxyPassReverse "/kc/" "https://localhost:8443/kc/"

    # Route everything else to Turing ES (HTTPS, internal)
    ProxyPass        "/" "https://localhost:2700/"
    ProxyPassReverse "/" "https://localhost:2700/"
</VirtualHost>
```

Key configuration notes:

`ProxyPreserveHost On` — passes the original `Host` header to the backend. Required for Keycloak and Turing ES to generate correct redirect URIs.

`RequestHeader set X-Forwarded-Proto "https"` — tells backend services that the original request was HTTPS, even though the internal connection may be unencrypted (in configurations where backends do not use SSL).

`SSLProxyCheckPeerName off` — disables hostname verification for the backend SSL connections. Required when using self-signed certificates on Keycloak and Turing ES. In production with CA-signed certificates, this should be set to `on`.

`/kc/` routes to Keycloak HTTPS because Keycloak was started with HTTPS. The path prefix matches the `--http-relative-path=/kc` configured in Step 2.

Logout flow

When Keycloak is enabled, logging out of Turing ES triggers a full OIDC logout. After the local session is cleared, the user is redirected to Keycloak's logout endpoint:

```
https://<your-hostname>/kc/realms/<your-realm>/protocol/openid-connect/logout
  ?client_id=turing-app
  &post_logout_redirect_uri=https://<your-hostname>
```

Keycloak invalidates the SSO session and redirects the user back to the Turing ES root URL.

Using an Existing Keycloak Instance

If your organization already runs a Keycloak instance, you do not need to install a dedicated one. Simply:

- Create a new realm (or use an existing realm, if your organization's policies allow it)
- Register Turing ES as a new client within that realm following the configuration in Step 3
- Configure the JVM properties in Step 5 to point to the existing Keycloak's issuer URI

The `issuer-uri` format is always `https://<keycloak-hostname>/kc/realms/<realm-name>` — adjust the path prefix if your existing Keycloak uses a different one (or no prefix).

Security Checklist

Before going to production, verify the following:

- [ ] Keycloak is only accessible via the Apache reverse proxy (port 8443 is not exposed externally)

- [ ] Solr is only accessible via the reverse proxy or not exposed externally at all
- [ ] Turing ES port (2700) is not exposed externally — all traffic goes through Apache
- [ ] All self-signed certificates are replaced with CA-signed certificates (or certificates from an internal PKI)
- [ ] `SSLProxyCheckPeerName` is set to `on` if using CA-signed backend certificates
- [ ] Keycloak admin credentials have been changed from defaults
- [ ] Database users have minimum required permissions (no superuser)
- [ ] JVM properties containing secrets (`client-secret`, `key-store-password`, `datasource.password`) are not logged or exposed in process listings — consider using an environment file or a secrets manager
- [ ] HTTP (port 80) redirects to HTTPS (port 443)
- [ ] Keycloak's `hostname-strict` mode is evaluated for your network topology

Related Pages

PAGE	DESCRIPTION
Authentication	Native session + API Key authentication (default mode)
REST API Reference	API endpoints and authentication examples
Configuration Reference	JVM properties and `application.yaml` settings
Installation Guide	Initial Turing ES installation and database setup