Funded by the European Union
NextGenerationEU

RECOVERY AND RESILIENCE PLAN

Architecture and Design Documentation

Deliverable D1

Reported by: Evolveum, s.r.o.

Milestone 1 06/2025

Evolveum

Table of Contents

Revision History

Revision	Date	Author	Description
0.1	2025-05-15	V.Kresnakova	Initial version
0.2	2025-05-21	V.Kresnakova	Model mapping recommendation system, Correlation recommendation system
0.3	2025-06-10	T.Chrapovic, V.Kresnakova	Baseline experiments for model mapping/matching and correlation recommendation system, Delineation
0.4	2025-06-17	A.Tkacik, Z.Kohut, V.Kresnakova	Connector code generator
0.5	2025-06-18	P.Malinak, D.Horvath, L.Skublik	User interface (MidPoint Studio plugin & Web UI)
0.6	2025-06-19	A.Zan	AI development & Testing tools
0.7	2025-06-20	M.Bielik, D.Horvath, A.Zan, L.Skublik, P.Mederly	Architecture and Design
0.8	2025-06-25	M.Bielik, P.Mederly, I.Farinic, K.Bolemant	Review
0.9	2025-06-27,	Whole team	Address review comments
1.0	2025-06-30,	I.Farinic, K.Bolemant	Final review

1. Introduction

1.1 Purpose

This document, submitted as Milestone 1 under the VAIA Projects (project code 09I05-03-V04-00101), defines the architecture and high-level design for our AI-driven extensions to midPoint. It establishes the conceptual framework and modular breakdown of the connector code generator, outlines the recommendation engines for schema mapping and identity correlation, and specifies the experimental methodology for validating embedding-based and LLM-assisted techniques. The document also describes our approach to automated delineation rule inference, details proposed UI/UX concepts (web-based wizards and an IntelliJ IDEA Studio plugin), and introduces the "AI Studio" MLOps toolchain for developing, testing, deploying, and monitoring the AI/ML pipelines that will drive subsequent project phases.

1.2 Document Overview

This document is organized into a series of chapters that collectively define the architecture, design, and research methodology for our AI-driven extensions to midPoint.

- **Chapter 1: Introduction** – presents the document's purpose, scope, organization, target audience, acronyms and abbreviations, and changed terms
- **Chapter 2: Executive Summary** – summarizes project scope and outcomes, and outlines project milestones and timeline
- **Chapter 3: System components with embedded AI research approach** – details the connector code generator, model-mapping recommendation system, correlation recommendation system, experimental pipeline, automated rule inference, UI concepts, and the AI Studio toolset
- **Chapter 4: Architecture and Design** – describes the high-level architecture via component-level diagrams, component descriptions, and API specifications for the smart integration layer, code-generation service, code validator, and integration catalog
- **Chapter 5: Infrastructure and Operations** – outlines workload organization and namespace strategy, API gateway and ingress management, workload standards and best practices, security operations, and backup and disaster recovery
- **Chapter 6: Compliance&Governance** – covers AI governance frameworks (including ISO/IEC42001), alignment with the EU AI Act, and GDPR considerations
- **Bibliography**: Compiles all referenced literature, standards, and online resources

1.3 Target Audience

This Architecture and Design document is intended for the following audiences:

- **Solution Architects and System Designers** – responsible for defining the overall identity-governance solution, its major components, and integration patterns.
- **IGA Engineers and midPoint Integrators** – hands-on implementers tasked with configuring, extending and deploying connectors and mapping recommendation systems.
- **DevOps and Infrastructure Teams** – in charge of provisioning, securing and operating the underlying infrastructure, CI/CD pipelines and API gateways.
- **Security and Compliance Officers** – overseeing GDPR, EU AI Act and other regulatory obligations, and ensuring that the proposed design meets organizational policies.
- **Project Managers and VAIA Stakeholders** – monitoring milestone progress (Milestone 1: Architecture & Design), budgets under the Recovery and Resilience Facility, and overall alignment with Component 9 (research funding) goals.
- **Research & Innovation Coordinators** – evaluating the use of generative AI for connector code generation, mapping recommendations and correlation logic, and guiding future R&D directions.

1.4 Acronyms and Abbreviations

AI

Artificial Intelligence

AI/ML

Artificial Intelligence/Machine Learning

AIMS

Artificial Intelligence Management Systems

API

Application Programming Interface

APU

Accelerated Processing Unit

CD

Continuous Delivery (or Continuous Deployment)

CI

Continuous Integration

CI/CD

Continuous Integration/Continuous Deployment

CIDR	Classless Inter-Domain Routing
CNCF	Cloud Native Computing Foundation
CRUD	Create, Read, Update, Delete
DoS	Denial of Service
DDoS	Distributed Denial of Service
DNS	Domain Name System
ELK	Elasticsearch, Logstash, Kibana
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IGA	Identity Governance and Administration
ISO	International Organization for Standardization

ISO/IEC 27001
: Information Security Management Systems standard

ISO/IEC 42001
: Artificial Intelligence Management Systems standard

IPMI
: Intelligent Platform Management Interface

JAR
: Java Archive

JDK
: Java Development Kit

JSON
: JavaScript Object Notation

LLM
: Large Language Model

ML
: Machine Learning

OIDC
: OpenID Connect

OCI
: Open Container Initiative

ORAS
: OCI Registry As Storage

PDF
: Portable Document Format

RAG
: Retrieval-Augmented Generation

RBAC
: Role-Based Access Control

RPO
: Recovery Point Objective

RTO

Recovery Time Objective

REST

Representational State Transfer

SCIM

System for Cross-domain Identity Management

SBOM

Software Bill of Materials

SSH

Secure Shell

TLS

Transport Layer Security

UI

User Interface

UX

User Experience

VLAN

Virtual Local Area Network

VM

Virtual Machine

XSD

XML Schema Definition

YAML

YAML Ain’t Markup Language

1.5 Changed Terms

Old term	New term
Marketplace	Integration catalog

2. Executive Summary

Modern organizations increasingly rely on a wide range of applications to support their daily operations. As the number of applications grows, so does the complexity of managing identities, permissions, and access in a secure and efficient manner. Identity Governance and Administration (IGA) platforms, such as midPoint, are essential for maintaining control over who has access to what and ensuring compliance with security and regulatory requirements.

While identity governance and administration (IGA) plays a critical role in securing modern organizations, one of the ongoing challenges is the slow, costly, and often incomplete process of onboarding applications into centralized IGA platforms. This bottleneck is largely driven by the need to develop custom integration code, commonly referred to as connectors, for each new application. Such development can take up to months in some cases, significantly delaying time-to-value. The applications that need to be integrated are different, typically involving a mix of cloud-based, on-premise, legacy, and even custom-built applications, each with its own unique interfaces, protocols, and data structures.

As a result, many organizations suffer from poor visibility across the IT environment, the emergence of shadow IT, higher operational costs, and an increased attack surface. With an estimated 80% of cyberattacks now exploiting identity-related vulnerabilities, there is a growing need for smarter, more efficient and scalable onboarding approaches.

At the same time, recent advances in artificial intelligence, particularly in generative models and machine learning, have opened new possibilities for automating complex tasks such as code generation, data mapping, and object correlation. These capabilities can significantly reduce the manual effort required for application onboarding and help organizations improve their identity governance posture in a scalable way.

2.1 Project Scope and Outcomes

The scope of the midPilot project includes addressing the critical technical and operational challenges related to onboarding applications into the midPoint Identity Governance and Administration (IGA) platform. The following areas are within the scope of the project:

Connector Development with Generative AI Support

The project aims to explore the use of generative AI technologies to reduce the time and manual effort required to develop connectors for midPoint. This includes support for various identity-related operations as well as support for different types of authentication and authorization, or object discovery.

Connectors need to target both cloud-based and on-premise applications, including those using legacy or proprietary integration protocols. Emphasis is placed on generating connector logic and configuration using AI models based on available documentation, metadata, and interface descriptions.

Model Mapping Recommendation System

Another key area of the project is the development of an AI-assisted mapping recommendation system. The scope covers:

- Suggesting attribute-level mappings between applications and midPoint,
- Supporting both simple and complex mapping scenarios, including transformations and scripted logic,
- Recommending schema extensions when no matching midPoint attribute exists,
- Supporting iterative refinement based on user feedback.

The mapping process is intended to be interactive, allowing users to review and confirm or adjust recommendations. The system is expected to use a combination of heuristics, machine learning, and LLMs to suggest mappings based on sample data and common patterns. Recommendation system should also support iterative refinement based on user feedback, allowing it to improve over time.

Correlation Recommendation System

The project also includes an assistance with correlating identity data between application and midPoint. This involves:

- Identifying correlation keys and generating correlation rules for objects such as users, accounts, roles, and groups,
- Recommending both simple and complex correlation logic,
- Simulating correlation outcomes before execution,
- Supporting iterative refinement based on user feedback.

The aim is to reduce reliance on manual correlation rule design, especially in large-scale or complex environments.

Consideration of Data Quality Issues

Given that data quality challenges are common during system integration, these issues must be carefully considered throughout the design and implementation phases. Typical data quality problems include:

- Typographical errors and inconsistent formatting,
- Missing values or incomplete datasets,
- Ambiguous or conflicting data representations.

The system must be designed to remain robust and effective even in real-world environments where data may be imperfect or inconsistent.

Integration Catalog

The scope also includes the creation of a dedicated Integration Catalog to support collaboration and reuse of the connectors and configurations. The catalog is intended to enable users to:

- Browse and use available connectors,
- Contribute or request development of a new connector,
- Share mapping configurations and correlation rules,
- Reuse proven integration logic across projects.

The catalog is intended to function as a central repository that can be browsed independently and is integrated with both midPoint and midPoint Studio.

User Interface and User Experience (UI/UX) Improvements

As part of the project scope, enhancements to the midPoint UI/UX are planned to support AI-assisted integration workflows. This includes the design and implementation of intuitive user interfaces for all key functionalities, such as a new wizard for connector code generation, and an improved wizard for model mapping and correlation configuration.

In addition, enhancements to the midPoint Studio plugin are within scope to support advanced users and integrators. These include integrated features for generating connector code, receiving AI-based mapping suggestions, and configuring correlation rules.

Support Tools for AI Development

The scope of the project includes the implementation of essential support tools for the development and maintenance of AI pipelines. The central component will be AI Studio, which includes tools for the following purposes:

- Designing and maintaining AI pipelines,
- Creating and maintaining automated test scenarios for verifying the overall AI solution and its modular components (e.g., suitability of LLM models),
- Executing and validating automated test scenarios,
- Tracing and observability platform to monitor and verify the correct operation of AI-based functionalities.

Delivery of the Solution

As part of this project, we will focus on a set of key activities to ensure that the final solution is not only fully functional but also easy to deploy, operate, and share with the wider community. Following will be key focus areas:

- DevOps and Infrastructure - development of DevOps artifacts required to build, deploy, and operate the solution in a containerized environment. This includes the implementation of

CI/CD pipelines and deployment automation scripts.

- Environment Setup and Component Integration - configuration and deployment of the development and production environments. This covers the integration of all necessary components into a cohesive and fully functional system.
- Documentation - preparation of comprehensive documentation covering system architecture, configuration, deployment procedures, and operational guidelines.
- Open Source Publication - release of the final solution under an open-source license (EUPL 1.2), including the publication of associated artifacts and documentation in public repositories.

2.2 Project Milestones and Timeline

The official length of the midPilot project is one year, starting April, 1st and ending March 30, 2026. During this period, there will be 3 milestones, as stated in the table below.

Date	**Milestone**	**Title**
06/2025	M1	Architecture and design documentation
12/2025	M2	Implemented end-to-end solution for rapid application onboarding
03/2026	M3	Tested and delivered solution. Evaluation of AI - how accurate and helpful it is for the problem.

3. Current State: Research, Analysis, and Prototypes

This chapter presents main components of the proposed solution: the code generator, mapping, correlation, and delineation recommenders, and development and testing tools. It provides the current state of the research, analysis of requirements, and evaluation of research prototypes that were created so far.

3.1 Connector Code Generator

Connectors are a key component of midPoint and other IGA solutions, responsible for communication and invoking operations on external applications, services, or systems. They enable midPoint to manage identities, entitlements, and other resources across diverse environments, such as cloud platforms, databases, and enterprise applications. The responsibility of connectors is to provide a consistent interface for midPoint to interact with these external systems, abstracting the underlying complexities of each system's API or protocol.

Creating a connector for midPoint typically involves the following steps:

1. Obtaining the correct documentation for the target application, which describes the API and its capabilities.
2. Processing the documentation to understand the API and its capabilities and to identify concepts and operations that are relevant for the connector.
3. Conceptualizing the connector, which involves defining object classes (e.g., user, group, or resource) and their attributes, which capture entities and concepts of the application that midPoint should manage.
4. Implementing basic support for the connector, which involves connecting to the target application, authenticating, and implementing basic operations such as reading objects.
5. Implementing support for relationship management, which involves defining relationships between objects and implementing logic to retrieve these relationships.
6. Implementing advanced support for the connector, which involves implementing complex operations such as creating, updating, and deleting objects.

To address the challenges of this process, in this part of the document the focus is on investigation of AI/ML research areas that could automate or assist key stages:

- Documentation discovery and web scraping solutions: automatically finding and fetching API documentation for target applications
- AI/ML-assisted understanding of API documentation: defining object classes, attributes, and operations
- AI/ML-assisted code generation

In parallel, existing applications, their APIs, and their documentation are analyzed to define requirements, identify common challenges in connector development and to gather examples for evaluation.

The final connector code generator will be designed and based on the experiments described, insights derived from their results and the requirements gathered from existing applications and their APIs.

3.1.1 Search & Scraping Solutions

When building a connector for midPoint, the very first challenge is discovering reliable, up-to-date API documentation for the target application. Often, a user only knows the application's name (for example, "AppX") and does not have a direct link to its developer guide or REST reference. In order to generate code that interacts correctly with that application's API, we must first find and fetch whatever official, or at least authoritative, documentation is publicly available.

This process naturally splits into two distinct activities:

1. **Documentation Discovery (Search)**: We automatically generate and submit queries (e.g., "AppX API documentation," "AppX developer guide PDF") to a search-engine API or repository search. By filtering and ranking the results, we aim to identify one or two URLs that point to the application's official API reference, PDF manual, or developer portal.
2. **Web Scraping (Fetch & Structure)**: Once we have candidate URLs (whether HTML pages or PDFs), we need to retrieve their contents, check their suitability for task at hand, and convert them into a structured, machine-readable format (typically JSON). If the candidate URLs do not contain relevant or sufficient information, we will navigate through the domain(s) until we arrive at sufficient documentation. The relevant documentation will then be forwarded to the next step of the pipeline (digesting).

Both steps can be implemented via various open-source libraries, commercial APIs, or turnkey services. Below is a detailed look at existing solutions for each activity, organized by category, along with their primary use cases, strengths, and limitations.

In addition, none of these steps are compulsory. For the users who have sufficient knowledge about the suitable candidate URLs, the Documentation discovery step is skipped. If the users are able to provide suitable documentation directly, Web scraping is skipped and the documentation will only be structured into a format suitable for the next pipeline step (digesting).

1. Search & Discovery Solutions

1. **Google Programmable Search / Custom Search API**: A hosted search-engine API that returns ranked results for any query you send. You can restrict the search to particular domains (for example, appx.com), prioritize official sites, and retrieve titles, snippets, and URLs. It leverages Google's relevance algorithms, makes it easy to apply domain or site

restrictions (for example, only look at docs.appx.com), and returns structured metadata (title, snippet) for quick filtering. However, it requires an API key and incurs usage costs beyond the free tier, and it only finds pages that Google has already crawled, so very new documentation might be missed.

2. **Bing Web Search API (Microsoft Azure)**: Similar to Google's offering, Bing's API provides web-search results with "fresh" filters to find recently published documentation. You can restrict results by domain and retrieve JSON payloads of titles, URLs, and snippets. It is often more cost-effective than Google for comparable quotas, and freshness filtering makes it easier to find the latest version of a rapidly evolving API. However, its index coverage can be slightly smaller in certain regions, and you still need to manually filter results to identify the official documentation.
3. **DuckDuckGo Instant Answer & API**: A privacy-focused search engine whose API returns unbiased, non-personalized search results. There is no cost for basic queries, and it provides a clear set of organic URLs without ads. However, it lacks domain-restriction features, you cannot easily limit the search to a specific site, and its rankings can be less consistent, so you may need to manually inspect more results.
4. **GitHub / GitLab Repository Search**: If the target application is open source, its repository often contains a "docs" folder, a README linking to an API spec (OpenAPI/Swagger), or a standalone openapi.json. GitHub's REST or GraphQL search endpoints let you query repository contents by file name or keyword. This provides direct access to raw markdown, JSON, or YAML files representing official specs, and you can pinpoint specific versions (branches or tags) to ensure consistency. However, it only applies to public or open-source projects, and forks or outdated branches may confuse automated logic unless carefully checked.
5. **Stack Overflow / Stack Exchange API**: While not a primary documentation source, many developers link to, summarize, or excerpt official docs in Q&A threads. By searching questions tagged with the application name (for example, "appx-api"), you may find links to the correct endpoint reference. It provides community-tested examples and hints at undocumented endpoints or edge cases. However, it is not authoritative, answers can be outdated or incomplete, and requires additional validation before trusting any discovered links.
6. **SearXNG**: An open-source metasearch engine that aggregates results from multiple search engines (Google, Bing, DuckDuckGo, etc.) and returns a unified set of links. It allows us to configure which sources to query and can be self-hosted for privacy. This provides a single API endpoint to query multiple search engines simultaneously, increasing the chances of finding relevant documentation. However, it requires setup and maintenance, and the quality of results depends on the configured sources.
7. **OpenAPI Directory**: A community-driven directory of OpenAPI specifications for various APIs. It allows searching by API name, category, or tags and provides direct links to the official specs. This is useful for quickly finding well-documented APIs that already have OpenAPI definitions available. However, it relies on community contributions, so not all APIs may be listed, and it does not cover non-RESTful APIs.

2. Web-scraping & Crawling Frameworks

1. **Scrapy (Python)**: An open-source, asynchronous crawling framework. You create “spiders” that start from seed URLs (like a docs homepage or sitemap) and automatically follow links under allowed domains or paths. It has built-in handling of robots.txt, throttling, retries, and concurrent requests, and is highly extensible so you can define pipelines to convert HTML fragments and tables into JSON. However, it has a steeper learning curve, you must design and maintain spider code, and can be overkill for very small or one-off scraping jobs.
2. **BeautifulSoup + Requests (Python)**: A lightweight combination for manually fetching and parsing a handful of HTML pages. You write simple scripts that use Requests to download a page and BeautifulSoup to select headings, code blocks, and tables. It is easy to get started with minimal setup and good for targeted extraction of a few pages, but it lacks concurrency or automatic link discovery and you must handle error retries, rate limiting, and broken links yourself.
3. **Selenium / Playwright / Puppeteer (Headless Browsers)**: These tools automate a real (or headless) browser to load pages, execute JavaScript, click through menus, and wait for dynamic content. They can also intercept network calls to fetch embedded JSON or OpenAPI specs directly, handling Single-Page Applications and JavaScript-driven docs (e.g., Swagger UI, Redoc), and extracting data that only appears after user interactions (e.g., expanding sections). However, they are resource-intensive (each instance is effectively a browser) and slower than pure HTML scraping, rendering and waiting for scripts can take several seconds per page.
4. **PDF Parsing Libraries (Apache Tika, PDFMiner, PyPDF2)**: When official documentation is only available as a PDF, these libraries extract text, detect font styles, and sometimes reconstruct tables. If pages are purely image-based, an OCR pass (e.g., Tesseract) may be needed. They automate offline/manual scraping of PDF guides and can batch-process multiple PDFs at once, but the structure (headings, tables) often degrades, manual review may be needed, and OCR can introduce errors in code examples or parameter listings.
5. **MarkItDown**: MarkItDown is a lightweight Python utility for converting various files to Markdown for use with LLMs and related text analysis pipelines. MarkItDown currently supports the conversion from PDF, PowerPoint, Word, Excel, Images (EXIF metadata and OCR), Audio (EXIF metadata and speech transcription), HTML, text-based formats (CSV, JSON, XML), ZIP files (iterates over contents), Youtube URLs, EPubs, and others.
6. **Import.io / Diffbot / Common Crawl (Commercial & Open-Source Services)**: Import.io is a paid, no-code platform where you visually select elements on a page and receive a generated API or extraction template; Diffbot is an AI-powered service that automatically identifies semantic objects (articles, tables, code blocks) on any website and returns structured JSON; and Common Crawl is a publicly available archive of billions of web pages where you can run big-data queries (via AWS) to find and extract pages for a given domain or pattern. These services offload maintenance (anti-scraping, selector updates) to the provider, Diffbot’s ML models can work on sites without writing per-site rules, and Common Crawl can provide historical snapshots if you need archived docs; however, Import.io and Diffbot

require paid subscriptions and Common Crawl is not real-time, you cannot fetch last-week's docs if they weren't in the archive.

7. **Point-and-Click Scrapers (Octoparse, WebHarvy)**: Desktop applications where a user visually clicks on page elements (headings, code blocks, table rows) to build an extraction workflow. These tools can handle pagination and basic data cleaning, making them extremely low barrier to entry, no coding required, and useful for quickly grabbing a few pages or tables without developer effort; however, they are not easily automated or integrated into a CI/CD pipeline, and licensing costs and limited flexibility make them unsuitable for complex or large-scale crawls.
8. **LLM-based Web Agents (AutoWebGLM, WebPilot)**: Tools developed for nagivating the web and executing web-related tasks using LLMs have been developed, although not with the specific purpose of web scraping in mind. These tools can navigate complex web aplications autonomously and thus could prove to be helpful for web crawling and scraping.

3. Summary Table of Existing Solutions

Category	Solution	Type	Primary Use Case	Key Strength	Notable Limitation
Search-API (Cloud)	Google Custom Search API [1]	Hosted Search API	Broad web search with domain restrictions	High relevancy ranking, easy domain filtering	Cost, rate limits
Search-API (Cloud)	Bing Web Search API [2]	Hosted Search API	Similar to Google, with "freshness" filters	Cheaper than Google, Azure integration	Slightly smaller index coverage
Search-API (Cloud)	DuckDuckGo API [3]	Hosted Search API	Privacy-focused, minimal bias	Free, no tracking	Limited filtering, less consistent results
Code Repo Search	GitHub / GitLab Search API [4]	Repository-based Search	Discover "docs/" folders, OpenAPI specs, or README links in open-source repos	Direct access to raw docs or JSON specs	Only works for public/open projects

Category	**Solution**	**Type**	**Primary Use Case**	**Key Strength**	**Notable Limitation**
Code Repo Search	Stack Exchange API [5]	Q&A Discovery	Find community answers that link to or reference documentation	Real-world examples, alternative endpoints	Not authoritative, may be outdated
Open-Source Scraping	Scrapy [6]	Python Framework	Large-scale crawling of multi-page documentation portals	High performance, built-in spider pipeline	Steep learning curve, code maintenance
Open-Source Scraping	BeautifulSoup + Requests [7]	Python Libraries	Quick, manual scraping of a few static pages	Very simple to set up	Lacks concurrency, no built-in error handling
Open-Source Scraping	Selenium / Playwright / Puppeteer [8]	Headless Browser	Dynamic or JavaScript-rendered docs (e.g., Swagger UI, SPAs)	Renders as a real browser, captures AJAX data	Resource heavy, slower scraping speed
Open-Source Scraping	PDFMiner / Apache Tika / PyPDF2 [9]	PDF Parsing Libraries	Extracting text and tables from PDF manuals	Handles offline PDFs, can process many files programmatically	Structure extraction can be inaccurate
Commercial Scraping Services	Import.io [10]	Visual Scraper / API	No-code extraction of tables or code blocks from any website	No-code configuration, automated maintenance	Subscription cost, limited customization

Category	Solution	Type	Primary Use Case	Key Strength	Notable Limitation
Commercial Scraping Services	Diffbot [11]	AI-powered Scraper	Automatically extract semantic objects (articles, tables, code blocks) from arbitrary sites	ML-driven, minimal per-site rules needed	Paid service, occasionally misclassifies data
Commercial Scraping Services	Common Crawl [12]	Public Web Archive / Data	Historical or large-scale searches over archived web content	Free access to vast archive, ideal for research	Requires big-data tooling, not real-time
Point-and-Click Scrapers	Octoparse / WebHarvy [13]	Desktop Application	Quick extraction of tables or code blocks from a handful of pages	No coding required, handles pagination easily	Best for small scopes; not easily scriptable
Point-and-Click Scrapers	Kofax / Kapow Katalyst [14]	Enterprise Data Extraction	Large corporate environments with complex workflow integration needs	Enterprise-grade security and orchestration	Very expensive, heavy setup/time to learn
LLM-based web agents	AutoWebGLM (Lai et al. 2024), WebPilot (Zhang et al. 2024)	LLM agentic tools	General web-oriented tasks such as web search	LLM-driven, autonomous	General shortcomings of LLMs, high complexity

Preprocessing Scraped Data

Once the raw search and scraping output has been collected and structured, the next step is to preprocess these data to ensure consistency, remove noise, and normalize formats before passing them to the AI-based digestor and code generation stages. Preprocessing typically involves cleaning HTML artifacts, removing duplicate or irrelevant sections, standardizing field names, validating that each endpoint description includes required metadata (method, path, parameters), and transforming any tables or code snippets into a uniform internal representation. This ensures that downstream modules receive high-quality, normalized input for accurate summarization and reliable connector code output.

3.1.2 Code Generator Theory and Existing Solutions

Generating connector code for midPoint requires translating a high-level specification, often reconstructed from scraped API documentation, into executable source in a target language (e.g., Java, Groovy). At a conceptual level, a code generator bridges two domains:

1. **Specification Domain**: A structured summary of available endpoints, parameter types, request/response schemas, authentication flows, and data models, typically represented as JSON objects or intermediate DSLs.
2. **Implementation Domain**: Concrete API-client code that invokes HTTP endpoints (for example, via Apache HttpClient or Spring RestTemplate), handles authentication tokens, marshals request payloads, and parses responses into Java objects.

Under the hood, modern code-generation approaches combine elements of:

1. **Program Synthesis**: Formal or heuristic search over a grammar or DSL to produce code that satisfies a declarative specification (for example, an OpenAPI-derived grammar) and passes validation constraints (type-checking, simple unit tests).
2. **Machine Learning**: Statistical or neural models that learn patterns from large corpora of existing "big code" (public GitHub repositories, connector templates) to predict likely code snippets given a natural-language or structured prompt.

Below is an overview of representative existing solutions, with links to their primary publications. For each, we summarize the core paradigm, typical use case, and any relevant caveats.

Program Synthesis and Symbolic Techniques

1. **AlphaCode (Li et al., 2022)**
   - **Paradigm**: Hybrid neural sampling, symbolic filtering. Pretrain a sequence-to-sequence Transformer on GitHub code, fine-tune on competitive programming problems, then sample millions of candidate solutions per problem and filter via example-based clustering and unit tests.
   - **Key Idea**: Large-scale sampling from a pretrained model reduces search complexity, and symbolic test-case filtering guarantees correctness.
   - **Note**: Primarily targeted at algorithmic tasks rather than API-client code, and requires large compute for sampling.
2. **Ellis Et Al, 2018**
   - **Paradigm**: Neural proposal of drawing primitives, symbolic synthesis of loops,

conditionals. In "Learning to Infer Graphics Programs from Hand, Drawn Images," a small neural network proposes low-level instructions, then a bottom-up synthesizer builds higher-level constructs.

- **Key Idea**: Decouple perception (neural) from synthesis (symbolic) to efficiently search in a domain-specific grammar.
- **Note**: Requires a well-defined DSL and domain-specific grammars; not directly applicable to REST connectors without a custom grammar.

Pretrained Models and Neural Code Generation

1. **CodeBERT (Feng et al., 2020)**
   - **Paradigm**: Bimodal masked-language model (Transformer encoder) pretrained on parallel "comment, function" pairs and raw code. Fine-tuned for code completion, summarization, or generation tasks.
   - **Key Idea**: Joint NL, PL pretraining fosters alignment between natural language and code representations.
   - **Note**: Encoder-only architecture, requiring a separate decoder or template to generate full code blocks; primarily excels at understanding rather than freeform generation.
2. **GraphCodeBERT (Guo et al., 2020)**
   - **Paradigm**: Extends CodeBERT by incorporating code structure graphs (data flow edges) into pretraining.
   - **Key Idea**: Leverage code structure information (AST, data flow) to improve semantic consistency and generate syntactically valid code.
   - **Note**: Graph construction adds preprocessing overhead; still an encoder-only model.
3. **PLBART (Ahmad et al., 2021)**
   - **Paradigm**: Denoising sequence-to-sequence pretraining on unlabeled NL and PL ("Program and Language BART"), supporting both code understanding and code generation.
   - **Key Idea**: Unified encoder-decoder architecture learns to reconstruct corrupted code snippets (and NL descriptions) and can generate new code from NL prompts.
   - **Note**: Requires large-scale pretraining on code, NL; decoder output may still need fine-tuning for specific API patterns.
4. **OpenAI Codex (Chen et al., 2021)**
   - **Paradigm**: Large autoregressive Transformer (GPT-3 fine-tuned on public GitHub) that generates code given a docstring or natural-language prompt.
   - **Key Idea**: Massive pretrained parameters allow few-shot or zero-shot code synthesis across languages; can generate full function bodies from a brief description.
   - **Note**: Model is closed-source, may hallucinate incorrect API calls without explicit

grounding; generation quality varies with prompt design.

5. **Commercial LLM-Based Coding Tools**
    - **GitHub Copilot** [15] – Context-aware code suggestions in IDE, powered by Codex/GPT.
    - **Salesforce CodeGen (Nijkamp et al., 2022)** – Generate code conditioned on NL and repository context, pretrained on large code corpus.
    - **CodeT5 (Wang et al. 2021)** – Seq2seq Transformer framework pre-trained on code NL pairs, used for tasks such as code generation, summarization, and translation between natural language and programming languages.
    - **Amazon Q Developer** [16] – AI-driven code search and generation focused on AWS services.
    - **Tabnine** [17] – Autocompletion for multiple languages, powered by LLM backends (e.g., GPT).
    - **Replit Agent** [18] – AI assistant in Replit IDE for snippet generation based on prompts.
    - **Polycoder (Xu et al., 2022)** – Open-source autoregressive Transformer trained on C code for snippet synthesis.
6. **StarCoder - BigCode** (Li et al., 2023)
    - **Paradigm**: Autoregressive code LLM with infilling capabilities, trained on The Stack dataset.
    - **Key Idea**: Supports code completion and infilling with a 15.5B parameter model, achieving ~40% pass@1 on HumanEval.
7. **StarCoder2 - BigCode** (Lozhkov et al., 2024)
    - **Paradigm**: Autoregressive code LLM trained on The Stack v2 corpus, with extended context window (up to 16K tokens).
    - **Key Idea**: Improved performance on code generation tasks, especially for long-context scenarios, across model sizes from 3B to 16B parameters.
8. **Code Llama - Meta** (Roziere et al., 2023/2024)
    - **Paradigm**: Instruction-tuned variants of Llama 2 optimized for code, available in 7B, 34B, and 70B parameter sizes.
    - **Key Idea**: Free-to-use model family offering strong performance on a variety of code tasks without licensing fees.
9. **WizardCoder** (Luo et al., 2024)
    - **Paradigm**: Instruction-fine-tuned code LLM using Evol-Instruct to generate complex code tasks.
    - **Key Idea**: Outperforms prior open models on standard benchmarks through curriculum-like instruction tuning.
10. **CodeGen2** (Nijkamp et al., 2023)

  - **Paradigm**: Unified encoder-decoder prefix-LM combining span corruption and infilling.
  - **Key Idea**: Flexible generation modes supporting both complete and partial code synthesis across 1B–16B parameter variants.

11. **USCD** (Wang et al., 2024)
  - **Paradigm**: Uncertainty-aware selective contrastive decoding as an inference-only plugin to reduce hallucinations.
  - **Key Idea**: Improves pass@1 and reduces output noise for models such as InCoder, CodeLlama, WizardCoder, and StarCoder.

Probabilistic “Big Code” and Embedding Based Methods

1. **Code2Vec (Alon et al., 2019)**
  - **Paradigm**: Embed code snippets by summarizing AST paths; originally designed for function name prediction and code understanding, but embeddings can feed into generation pipelines.
  - **Key Idea**: AST path embeddings capture semantic, syntactic structure in a low-dimensional space.
  - **Note**: Not a standalone generator; requires additional decoder logic to map embeddings back to code tokens.
2. **Allamanis Et Al, 2018**
  - **Paradigm**: Survey of probabilistic models for “big code,” including n-gram, RNN, Transformer-based models trained on large open-source corpora to predict next tokens or suggest completions.
  - **Key Idea**: Code exhibits “naturalness” enabling statistical language models to perform tasks like autocompletion and code recommendation.
  - **Note**: Early models lacked the contextual capacity of modern LLMs, often limited to short code fragments.

3.1.3 Summary Of Existing Solutions

Approach	Representative Work	Key Idea
Symbolic Synthesis	AlphaCode (Li Et Al, 2022)	Large-scale neural sampling, symbolic filtering on test cases
Neural Pretraining, Bimodal	CodeBERT (Feng Et Al, 2020)	Pretrain on paired code, NL data, fine-tune for NL, code
Neural Pretraining, Graph	GraphCodeBERT (Guo Et Al, 2021)	Incorporate data-flow graphs into pretraining objectives

Approach	**Representative Work**	**Key Idea**
Neural Pretraining, Seq2Seq	PLBART (Ahmad Et Al, 2021)	Denoising autoencoding on code, NL pairs for generation, translation
Autoregressive LLM, Fine Tuning	Codex (Chen Et Al, 2021)	Finetune GPT on code, sample multiple candidates, filter by tests
Embedding Based	Code2Vec (Alon Et Al, 2019)	Embed AST paths to represent code semantics, used for naming, generation
Probabilistic Big Code	ML for Big Code Survey (Allamanis Et Al, 2018)	Models code with n-grams, RNNs, Transformers to capture statistical patterns
Program Synthesis, Graphics	Ellis Et Al (2018)	Infer drawing primitives with neural nets, then synthesize loops, conditionals
LLM Tool, IDE Integration	GitHub Copilot [19]	Context-aware code suggestions in IDE, powered by Codex
LLM Tool, API Service	OpenAI Codex (Chen et al., 2021)	Generate standalone functions from docstrings or NL prompts
LLM Tool, Code API	Salesforce CodeGen (Nijkamp Et Al, 2022)	Generate code conditioned on NL and repository context
LLM Tool, Text-To-Text Model	CodeT5 (Wang Et Al, 2021)	Seq2Seq framework for code generation, summarization, translation from NL□PL
LLM Tool, Commercial Assistant	Amazon Q Developer [20]	AI-driven code search and generation focused on AWS services
LLM Tool, Autocompletion	Tabnine [21]	Autocompletion for multiple languages, powered by LLM backends (e.g., GPT)
LLM Tool, IDE Agent	Replit Agent [22]	AI assistant in Replit IDE for snippet generation based on prompts
LLM Tool, Open-Source Model	Polycoder (Xu Et Al, 2022)	Open-source autoregressive Transformer trained on C code for snippet synthesis

Approach	**Representative Work**	**Key Idea**
Autoregressive Code LLM	StarCoder (Li et al., 2023)	Supports code completion and infilling with a 15.5B parameter model, achieving ~40% pass@1 on HumanEval
Autoregressive Code LLM	StarCoder2 (Lozhkov et al., 2024)	Improved performance on code generation tasks, especially for long-context scenarios, across model sizes from 3B to 16B parameters
Instruction-Tuned LLM	Code Llama (Roziere et al., 2023/2024)	Instruction-tuned variants of Llama 2 optimized for code, available in 7B, 34B, and 70B parameter sizes
Instruction-Tuned LLM	WizardCoder (Luo et al., 2024)	Outperforms prior open models on standard benchmarks through curriculum-like instruction tuning
Unified Seq2Seq LLM	CodeGen2 (Nijkamp et al., 2023)	Flexible generation modes supporting both complete and partial code synthesis across 1B–16B parameter variants
Contrastive Decoding Plugin	USCD (Wang et al., 2024)	Improves pass@1 and reduces output noise for models such as InCoder, CodeLlama, WizardCoder, and StarCoder

3.1.4 Open Challenges and Future Directions

1. **Handling complex specifications**: Real-world connectors often require multi-step workflows (authentication, pagination, error handling). Existing LLMs can produce syntactically valid code but may miss edge cases or usage conventions.
2. **Combining symbolic guarantees with statistical power**: Methods like AlphaCode show promise, but research is needed to more tightly integrate constraint solvers with LLMs, ensuring correctness without exhaustive sampling.
3. **Reducing hallucinations**: Retrieval, augmented generation and unit-test filtering mitigate incorrect outputs, but more robust grounding mechanisms (type checkers, static analysis) are still necessary.
4. **Domain adaptation and few-shot learning**: Connector code often involves proprietary or

less-common APIs; transfer learning and few-shot prompting remain active research areas.

5. **Efficiency and latency**: Large Transformer models incur high inference costs; lightweight distilled models or on-device generation could enable real-time connector generation in production systems.

3.1.5 State of the Art in Evaluation Metrics for Code Generation

Evaluating code produced by large language models remains an open problem: traditional NLP metrics often fail to reflect functional correctness or deeper semantic quality, and code-specific metrics still lack extensive validation against human judgment (Evtikhiev et al., 2023; Ren et al., 2020; Tran et al., 2019).

1. Lexical and Structural Similarity–Based Metrics

Traditional metrics such as BLEU (Papineni et al., 2002), ROUGE (Lin, 2004), METEOR (Banerjee & Lavie, 2005; Denkowski & Lavie, 2014), ChrF (Popović, 2015), edit distance, and exact match measure n-gram or character overlap and raw edit operations. They are rapid and easy to compute, making them useful for quick sanity checks and early-stage comparisons, but they correlate poorly with functional correctness and cannot account for semantically equivalent code that is lexically different (Papineni et al., 2002; Lin, 2004; Banerjee & Lavie, 2005; Denkowski & Lavie, 2014; Popović, 2015).

Code-specific metrics like CodeBLEU (Ren et al., 2020) and RUBY (Tran et al., 2019) extend n-gram scoring with Abstract Syntax Tree and data-flow matching or program-dependency-graph comparison. These methods improve on surface-level matching and show better correlation with human judgments in some studies, but they still struggle to predict execution errors or fully recognize semantic equivalence. They also introduce additional computational overhead and have not been proven to consistently outperform traditional metrics across diverse tasks (Ren et al., 2020; Tran et al., 2019).

2. Execution-Based Metrics

Execution-based evaluation begins by ensuring syntactic validity through compilation or interpretation success (Austin et al., 2021). This check is very fast and guarantees that the generated code adheres to language rules, but it provides no insight into runtime behavior or logical correctness.

The unit-test pass rate, and in particular the pass@k metric, validates generated code against predefined test suites (Chen et al., 2021). Pass@k measures the probability that at least one out of k sampled code outputs successfully passes all tests, directly assessing whether the code fulfills its specification. Benchmarks such as HumanEval report pass@1, pass@10, and so on. However, pass@k depends heavily on the quality and coverage of the test suite and may overestimate a model’s practical usefulness if users sample fewer than k times.

Performance and efficiency metrics-measuring runtime, memory usage, or energy consumption offer insight into real-world resource utilization (Austin et al., 2021; Chen et al., 2021). Frameworks such as EffiBench and Mercury enable these evaluations, but they require carefully controlled benchmarks, instrumentation, and management of variability across execution environments.

3. Human-Centric Evaluation

Human-centric evaluation relies on expert reviewers to judge aspects like readability, maintainability, and usability of generated code. Methods include blind peer reviews and real-world integration tests, which reveal issues automated metrics miss (Yu et al., 2024; Wu et al., 2024; Dibia et al., 2023). While this approach provides deep insights into code quality, it is time-consuming, expensive, and can vary due to reviewer subjectivity.

4. LLM-as-a-judge

An emerging trend is to enlist a separate language model to act as a proxy evaluator. By crafting prompts that guide the model to critique or score generated code examples include the ICE-Score framework researchers aim for scalable, explainable evaluation (Jiang et al., 2024; Zhuo, 2024; Zheng et al., 2024). Early results suggest that LLM-based evaluation can provide useful feedback at scale, but this technique remains experimental. Its reliability depends heavily on prompt design, the judge model's inherent biases, and its ability to reason about complex code constructs (Jiang et al., 2024; Zhuo, 2024; Zheng et al., 2024).

Open Challenges and Trends

Despite these developments, no single metric consistently aligns with human judgment across languages and domains. There is a pressing need to move beyond isolated-function benchmarks toward whole repository or system-level evaluation, capturing cross-file dependencies and architectural concerns. Improving test-suite adequacy to detect logical and security flaws remains critical. Future directions include exploring multimodal and context-aware evaluation pipelines, integrating with CI/CD workflows, and fostering human-AI collaboration in assessment.

3.1.6 Baseline Experiments for Connector Code Generator

The experiments conducted focused on exploring and evaluating three key areas: scraping, digesting, and code generation. These areas were selected based on prior research and analysis of the typical steps involved in developing connectors for midPoint. They are expected to represent the core functionality required in most real-world scenarios.

The Search and Discovery phase, while part of the broader conceptual framework, was considered supplementary. Its primary purpose is to improve usability for non-expert users with minimal input. Due to its supportive role, it was excluded from the first phase of prototyping. The following sections summarize the results and insights gained from the experimental evaluation of the three core components.

Experiment: Scraper Component - General Approach and Current Development

General Approach

The role of Scraper within the pipeline is to use the links provided by either the user or from automated Search and Discovery, to scrape and navigate across a provided domain or domains, in order to find and download the documentation relevant and necessary for the Digestion and Code Generation. Specifically, the target of Scraper component is to provide the inputs for Digester.

Scraper is intended to principally perform the following three high-level tasks:

1. Scraping - to scrape the content of the web pages provided
2. Sufficiency Evaluation - to evaluate whether the content scraped so far is sufficient as an input for the Digester. If it is evaluated as sufficient, the search is over and no further scraping is required. If the opposite is the case, the third task is performed.
3. Reference Selection - to select further references for scraping that might require further evaluation. The steps 1. and 2. are then repeated.

The three-step process described above is repeated until it 'closes the loop', i.e., until the Sufficiency Evaluation results in a positive answer or the process ends by different means (e.g., reaches the maximum number of loop repeats, amount of time, has nothing else to scrape further, etc.). In addition, as a link between the Scraper and Digester, another LLM tool might be required for pruning of the scraped documentation of the irrelevant content which might accumulate in the process of repeated scraping. This would effectively be a post-processing step outside of the main Scraper loop.

Current Development

The current prototype of Scraper addresses the three main steps described in the General approach section, with the main focus being on the LLM-related steps (i.e., Sufficiency evaluation and Reference Selection).

1. Scraping

   The provided urls are scraped using RecursiveUrlLoader, a class native to the LangChain python package. The scraped content is then parsed using the BeautifulSoup package. This is roughly equivalent to the BeautifulSoup + Requests (Python) combination described previously, as the RecursiveUrlLoader wraps the Requests library. The upside of RecursiveUrlLoader is that it allows to set scrape depth (and thus extending the search by scraping subdomains within set depth), and although we do not use this feature in the current prototype for simplicity reasons we see a potential for this to be of potential benefit in the more advanced prototype scenarios.

   The scraped content of each page is converted to markdown format, and chunked into smaller pieces, which are then stored in a vector storage (LangChain's implementation of

Chroma). These steps are performed because LLMs have limited context size (i.e., the amount of input they can ingest as part of a prompt is limited) and using a chunked vector storage allow them to pull only the relevant parts from the storage and use those (an approach called Retrieval Augmented Generation - RAG). In both following steps we are currently using RAG.

2. Sufficiency Evaluation

   In this step, the LLM is tasked with the task of provided a simple yes or no answer to a question of whether the documentation scraped so far is sufficient for the goal of writing a connector. The 'yes' answer ends the scraping process, while the 'no' answer triggers the nex step.

   The current design of this step uses a single LLM call with the access to the vector storage (RAG). The potential issue is the failure to provide the 'yes' answer even under the circumstances where the process is sufficient.

3. Reference Selection

   In this step, the LLM is tasked with the selection of the urls to select for further scraping. This steps employs RAG, just like the previous step, with an addition to the context where the LLM is provided with the list or urls to choose from. Based on the parameters of a test, we supply it with either a) only the urls selected from the pages scraped in the current iteration of the loop (url_method = 'current'), or b) with all urls selected from all scraped pages (url_method = 'all').

   The explicit provision of urls to the LLM as part of the context is not be a fixed choice as we might opt to let the LLM to choose the url based on its own context parsing. Currently, it is an operational choice, which allows us to keep track about whether the process is going in the expected direction through the expected search path and will be a subject to further experimentation.

Prototype Evaluation

The evaluation process in the case of Scraper seeks to evaluate two principal aspects of the tool, which are operationalised into two binary metrics:

- Target Identification - this metric establishes whether the target documentation was found or not
- Loop Closure - this metric establishes whether the scraped content was evaluated as sufficient

Note that there is a degree of conditionality, between the two metrics in the sense that Loop closing is relevant only after Target Identification was successful. In the scenario where the loop is closed before the target is identified, this is considered a failure. Thus, for the purpose of our evaluation, we establish a Composite Score metric out of the two metrics above where we assign

100% if a model succeeds in both tasks, 50% if it succeeds in the Target Identification only, and 0% if loop was closed but no target was identified or if it succeeds in none. This Composite Score is calculated per application and then averaged, resulting in a single value per model and per url_method.

The 50% score in the cases where the target is correctly located and scraped but is not recognised as such, models an expected real-life scenario, where the user will be able to override the machine-generated judgement about the scraped content insufficiency, and to proceed to the next steps based on their own evaluation and judgement.

Thus far, these tool were tested on five different LLMs, with three coming from the LLama family 'meta-llama/llama-3.3-70b-instruct:free', 'meta-llama/llama-4-scout:free', 'meta-llama/llama-4-maverick:free'. While the first one is a rather large model with dense architecture (70b in total), the other two rely on mixture-of-experts architecture (both 17b per expert), differing in the number or experts (scout: 16E vs. maverick: 128E) as well as other important parameters such as context length. The other two LLMs we have used are from mistral family, 'mistralai/magistral-small-2506' and 'mistralai/magistral-medium-2506' which are much lighter in terms of the amount of parameters (24b). We included these smaller models because such models are more suitable for running in environments with limited hardware resources.

The tests were performed over two applications, Forgejo and Authress. The results of the llama-family show mixed performance, with 'meta-llama/llama-4-maverick:free' showing the worst performance. This is not surprising since this model is not only smaller in terms of total parameter size but is also limited in terms of context size, compared to the other two llama models. Quite surprisingly, the mistral-family models have show solid performance with 'mistralai/magistral-medium-2506' being the top performer with url_method = 'all'. The downside of these smaller models was their tendency to close the loop before having scraped sufficient documentation. Regardless, this provides a motivation to not narrow our focus on large LLMs only but to also further explore the suitability of smaller LLMs for the current purpose.

Model Performance Summary with url_method = 'all'

Model	Composite Score (average)
'meta-llama/llama-3.3-70b-instruct:free '	25%
'meta-llama/llama-4-maverick:free'	0%
'meta-llama/llama-4-scout:free'	50%
'mistralai/magistral-medium-2506'	75%
'mistralai/magistral-small-2506'	25%

Model Performance Summary with url_method = 'current'

Model	**Composite Score (average)**
'meta-llama/llama-3.3-70b-instruct:free '	50%
'meta-llama/llama-4-maverick:free'	0%
'meta-llama/llama-4-scout:free'	25%
'mistralai/magistral-medium-2506'	0%
'mistralai/magistral-small-2506'	50%

Further Considerations and Challenges

1. Addressing the url Selection Method

   The parameter url_method described above appears to be a significant factor of success, we will therefore seek to implement more dynamic strategies and fall-back mechanisms, potentially not intervening in the process of selection by simply providing no url_method to the LLM.

2. Extending Toward Dynamic Web Scraping Abilities

   For the purposes of prototyping, we have chosen a simpler scraping approach (RecursiveUrlLoader + BeautifulSoup), which limits our testing abilities to the extent that we are not able to properly test dynamic web sites which is becoming a limiting element. Tools such as Selenium will be further considered and implemented in the in order to cover a wider scope of test cases.

3. Extending the Test Application Set

   Relating to the previous challenge, we aim to extend our set of test applications, since in the current prototype we limited the test scope to two example applications. This will enable us to test the used models and methods more reliably with respect to a wider range of cases.

4. Parameter Testing and Optimization

   There are various parameters that will be attended to in order to maximise the success of the tool. These are, for example, the RAG-related parameters such as the search type and the particular type's parameters, the size of the text chunks stored in the vector store, the embedding model and its interaction with the main LLM. Other parameters include more operational characteristics such as the maximum number of rounds or other stopping rules if the search is not succesful or needs to fall back to a different method.

Experiment: Digester Component - General Approach and Current Development

General Approach

The purpose of digesting in the current pipeline, as mentioned above, is to extract definitions and specifications from the provided or scraped documentation. For this task it is assumed use of LLM, which should help with identification of types of objects in the application, as well as determining whether or not a given object type is relevant for IDM/IGA. In addition to object type identification, the LLM should be able to identify the corresponding API endpoints and the object type schema, i.e. a list of object attributes and their types. Moreover, the LLM is also responsible for identifying and describing object relationships, which includes detecting which objects are related to each other (e.g., users and groups) and providing definitional information about these relationships (e.g. who is the owner of the relationship).

The digesting procedure is as follows:

1. Identification of the type (technical documentation in natural language or machine-readable structured documentation) and format of the documentation (PDF, HTML, MD, JSON, YAML, …)
2. Loading documentation from provided URL(s) into memory.
3. Extracting the list of endpoints
4. Identification of object types, and relationships between them
5. Extracting object types schemas
6. Extracting user documentation for endpoints (query parameters, request type, etc…)

Current Development

The current prototype of the digester component focuses on extracting information from JSON and YAML files according to the OpenAPI specification.

In the first step, we detect if the supplied URL points to a JSON or YAML file and then, using json and PyYAML libraries, we load the file into a python dictionary.

Next, we extract the list of endpoints and their descriptions from the dictionary.

We provide this list to the LLM, which we ask to identify Object Classes, fetch-all and fetch-by-id endpoints and whether it is an IDM/IGA related object. To implement the LLM we use the langchain framework and specifically its structured output capabilities.

It should be noted that some models sometimes fail to provide their responses in the correct structured form, so we repeat the call to LLM on a failed attempt until we get the output in the requested form, but no more than 5 times.

Prototype Evaluation

In evaluating the digester prototype, we evaluate the model's ability to correctly identify Object Classes, fetch-all and fetch-by-id endpoints, as well as endpoint complexity.

For evaluation purposes, we have manually prepared a set of test cases against which we compare the results of the model. This set contains test data for the following applications:

- Forgejo (swagger json)
- Github (openapi yaml)
- Gitlab (swagger yaml)
- OpenProject (openapi json)
- Zoom (openapi json)

The actual evaluation for the chosen model and application is performed in the following steps:

1. Comparison of the list of Object Classes predicted by LLM and the expected list
2. Comparison of the list of endpoints predicted by LLM and the expected list
3. Comparison of the complexity of the endpoints predicted by the LLM and the expected values

It should be noted that these comparisons are interdependent. Therefore, in the final evaluation of the model, the metrics from the individual steps will be multiplied together.

We have tested the prototype performance of several state of the art models as seen in the table below:

Model	**Precision**	**Recall**	**F1**
google/gemini-2.0-flash-001	0.717	0.643	0.664
openai/gpt-4.1	0.446	0.850	0.558
mistralai/devstral-small	0.497	0.783	0.520
deepseek/deepseek-chat:free	0.460	0.817	0.494
mistralai/devstral-small:free	0.480	0.517	0.468
deepseek/deepseek-chat	0.344	0.817	0.454
qwen/qwq-32b:free	0.269	0.717	0.367
qwen/qwen3-235b-a22b:free	0.269	0.550	0.339
qwen/qwen3-32b:free	0.181	0.663	0.240
qwen/qwen3-30b-a3b:free	0.050	0.150	0.075
qwen/qwen3-14b	0.043	0.150	0.067

Model	Precision	Recall	F1
qwen/qwen3-14b:free	0.036	0.283	0.063

Further Considerations and Challenges

1. Addressing Endpoint Complexity Identification

   None of the models evaluated were consistently able to correctly identify endpoint complexity. It will therefore be necessary to find a good formulation of the LLM prompt that leads to better results.

2. Implementing Natural Language Documentation Digesting

   For the purposes of the prototype, we implemented the documentation digesting in a machine-readable form that conforms to the OpenAPI (or swagger) specification. Since such documentation is not always available, it will be necessary to extend the implementation to include natural language documentation digesting.

3. Implementation of the Extraction of Information about Relations between Class Objects

   In the prototype we did not focus on the implementation of the extraction of relationship information between Class Objects. However, this aspect is crucial for understanding the interactions and dependencies in the system and will therefore need to be included in the next phases of the implementation.

Experiment: Code Generator

The role of the code generator is to generate scripts using LLM, from the definitions and specifications extracted by the digester component, which will be linked with the skeleton code. These scripts contain the definitions of the native schema of the application, the mapping of the native schema to the ConnId schema, the definitions of the relationships between objects and the implementation of specific parts of the operations with the application. The main goal in the initial phases of the project is to focus on the implementation of fetch-all and fetch-by-id operations of individual objects from the application. In addition, the generated code will be validated using the code validator service, in order to ensure quality and security.

The code generation process generally follows these steps:

1. Preparation of the context for the generator (documentation for the code skeleton, object specifications, endpoint definitions, files generated so far, etc.)
2. Defining the task for LLM
3. Generating code using LLM
4. Validation of the generated code using the validation service
5. In case of a negative feedback from the validation service, define a new task for LLM, which

will contain the message from the validation service and proceed back to step 3.

This general procedure is adapted to specific requirements for generating specific scripts (e.g.: User.native.schema.groovy, Team.connid.schema.groovy, Organization.search.groovy, etc.).

3.1.7 Analysis of Current Connector Frameworks and APIs

In order to determine the best approach for code generation, existing open-source connector base frameworks for ConnId connectors and API documentation were evaluated. The analysis was done based on the list of popular cloud applications, used in enterprise space, their European counterparts and open-source alternatives.

Observations from Existing Cloud Applications

The key observations from API documentation analysis are:

- **Authentication & Authorization**: Most applications use OAuth 2.0 or API keys for authentication, with some supporting JWTs.
- **Protocol, Data Schema & API Design**: More applications are starting to using SCIM 2.0 for identity management, while RESTful APIs still remain dominant, usually accompanied by OpenAPI specifications.
    - **SCIM 2.0** - is a standard for identity management, providing a common schema for user and group management, allows programatic schema discovery and management of users, groups, and other identity resources.
        - Most of the applications do not support full SCIM 2.0, but rather a subset of it.
        - Naming of attributes in application differs from the SCIM 2.0 standard. Usually they provide mapping table in their documentation to ease manual integration and user understanding.
        - Some applications do not provide enough information using SCIM 2.0, so fallback to RESTful APIs seems necessary in such cases.
    - **REST** - is the most common protocol for API communication, used for any general purpose, not specific only to identity management.
        - Most applications provide **OpenAPI** specification for their REST APIs.
        - Most applications use common patterns for REST API design, which differs only in naming conventions and some specific parameters.
    - Few applications do not provide any common machine-readable documentation.
- **Data Formats**: JSON is the most common data format for request and response payloads, with some applications supporting only YAML.

On-Premise Applications

Similar observations apply to modern on-premise (non-cloud) applications, which we will need to support in the future, as they are still widely used in enterprise space.

Older on-premise applications require support for other approaches. We are planning to support these application using connectors with direct SQL database connections with multi-table support.

Observations from Existing ConnId Connectors

Existing open-source ConnId connectors follow two distinct approaches:

- **Pure ConnId Implementation**: Connectors are manually implemented using only ConnId framework and its API. This approach requires deep knowledge of the ConnId framework and the target application API.
- **Connector Framework Based Implementation**: Connectors are implemented using a common framework that provides a set of base classes and methods to simplify the implementation. This approach allows for faster development and easier maintenance, but no solution supporting mix of SCIM & REST was observed.

Base Connector Framework Requirements

Based on key observations, existing ConnId connectors, their complexity, and software development best practices we have identified the following requirements for the connector framework:

- **Support for Authorization and Authentication**: The framework should provide a way to implement various authentication and authorization mechanisms, such as OAuth 2.0, API keys, and JWTs.
- **Support for SCIM 2.0**: The framework should provide out-of-the box support allowing for easy implementation of SCIM 2.0 connectors.
    - SCIM 2.0 support should be optional, as not all applications provide full SCIM 2.0 support.
    - Schema could be discovered using SCIM 2.0 protocol and its translation to ConnId should be supported by built-in algorithm to ensure consistency.
        - Schema attribute name mapping should be customizable, as some applications use different naming conventions, in order to minimize manual mapping effort. This allows also for interoperability with REST part, by using common names.
    - Several strategies for searching, retrieving, creating, updating and modifying users, groups and custom resources.
        - Some analyzed applications do not support relative updates for all attributes, only for some of them, so framework should provide a way to configure which update strategy should be used.

- **Support for REST APIs**: The framework should provide a way to implement operations using REST APIs, allowing for easy integration with applications that do not support SCIM 2.0 or have limited SCIM 2.0 support.
  - REST API support should be optional, as some applications provide only SCIM 2.0 support.
  - Framework should support custom mapping of REST API endpoints to ConnId operations, allowing for flexibility in implementation.
  - Framework should provide common strategies for retrieving data, such as pagination, filtering, and sorting.
  - Framework should provide easy way to implement filter mapping, allowing for easy mapping of ConnId filters to REST API calls.
- Framework should provide a way to implement custom operations, allowing for easy implementation of custom logic that is not covered by the framework.
- Framework design should minimize boilerplate code, need to write custom code, and allow for easy customization of the connector implementation.
- Framework design should prefer declarative approach over imperative, allowing for easy configuration of the connector behavior, minimizing the need for custom code.
- Optionally framework should be easily extended to support new features, such as additional authentication mechanisms, new data formats, or new protocols.
  - Based on our research, modern on-premises applications should be covered by existing requirements.
  - Framework should be extendable to support other protocols, such as SQL (multitable database connector) for support of older on-premise applications, which is planned for the future.

Based on analysis of existing ConnId connector frameworks and the requirements we have identified, we have concluded that there is no existing framework that meets all of the requirements outlined above. Therefore, we have decided to design and implement our own base connector framework that will support SCIM 2.0, REST APIs and SQL connections and will be designed to meet the identified requirements outlined above, providing a solid foundation for generating connectors for various applications.

3.2 Model Mapping Recommendation System

The system should help users by suggesting how to map attributes from connected application to their counterparts in midPoint.

This section presents the theoretical foundations and provides an overview of existing solutions, highlighting their key advantages and limitations. The final recommendation pipeline will be composed based on the experiments described and the insights derived from their results.

3.2.1 Survey of Schema Matching and Mapping Algorithms

Schema matching is the problem of finding potential associations between elements (most often attributes or relations) of two schemas. Given two schemas S1 and S2, a solution to the schema matching problem, called a schema matching (or more often a matching), is a set of matches. A match associates a schema element (or a set of schema elements) in S1 to (a set of) schema elements in S2. Research in this area focuses primarily on the development of algorithms for the discovery of matchings. Existing algorithms are often distinguished by the information they use during this discovery. Common types of information used include the schema dictionaries and structures, the corresponding schema instances (if available), external tools like thesauri or ontologies, or combinations of these techniques. Matchings can be used as input to schema mappings algorithms, which discover the semantic relationship between two schemas (Kementsietsidis, 2009).

While schema matching finds correspondences between elements of two schemas, a **schema mapping** goes one step further by describing how data conforming to one schema (the source) can be transformed into data conforming to another (the target). The problem of establishing associations between data structured under different schemas is at the core of many data integration and data sharing tasks. Research on schema mapping has focused on the formal specification of schema mappings, the semantics of mappings, along with techniques for creating schema mappings (Fuxman and Miller, 2009).

Based on the provided sources, none of the papers specifically focus on schema matching or data integration techniques for Identity Governance and Administration (IGA), identity directories, or access management systems. All papers discuss schema matching and data integration within a general context. Therefore, we will categorize the papers solely based on whether they represent traditional or advanced approaches within this general domain.

Summary of Key Algorithms and Approaches

The sources describe a variety of key algorithms and approaches for schema matching and data integration:

- **Similarity Flooding (SF):** This is a **structural graph matching algorithm** designed to match diverse data structures by converting them into directed labeled graphs. It uses an iterative fixpoint computation to determine the similarity between nodes in the graphs. SF can be used as a generic matching algorithm across application areas and is implemented as an operator in schema manipulation tools. It produces a mapping between corresponding nodes, which is then filtered (Melnik et. al, 2002).
- **Cupid:** Described as a **hybrid schema matcher**, Cupid discovers mappings based on element names, data types, constraints, and schema structure. It integrates **linguistic and structural matching**, biases towards leaf structures, and uses context-dependent matching. It parses compound names, categorizes elements by data type and linguistic content, and calculates linguistic similarity. Structural similarity is computed bottom-up based on schema trees (Madhavan et. al, 2001).

- **COMA:** A **system designed for the flexible combination of schema matching approaches**. It provides an extensible library of individual and hybrid matchers and a framework for combining their results. COMA supports matchers based on element names, data types, structural properties, data instances, and auxiliary sources. It distinguishes between hybrid and composite combination approaches, focusing on the composite approach for flexibility. COMA also includes a **reuse-oriented approach** for leveraging previous match results (Do and Rahm, 2002).
- **iMAP:** This system focuses on the challenging problem of discovering **complex semantic matches** between database schemas, which are often not one-to-one (1:1). iMAP is semi-automatic and combines searching through candidate matches with methods for evaluating them. It utilizes domain knowledge, such as overlap data and external data mining, and provides explanations for its decisions (Dhamankar et. al, 2004).
- **Un-interpreted Matching (Opaque Data):** This technique addresses schema matching when element names and data values are "opaque" or difficult to interpret. It does not rely on the interpretation of data elements or schema elements. The approach involves two steps: measuring pair-wise attribute correlations to construct a dependency graph and then using a graph matching algorithm to find corresponding nodes (Kang and Naughton, 2003).
- **Machine Learning (ML) / Deep Learning (DL) Methods:** ML techniques have been explored for schema matching, sometimes using instance-level information. ADnEV (Shraga et. al, 2020), DITTO (Li et al., 2020) and SMAT (Zhang et. al, 2021) are examples of deep learning models used for schema-level matching. SMAT is noted for capturing rich text information.
- **Large Language Model (LLM) Methods:** Recent work leverages LLMs like GPT-4 for schema matching (Feng et. al, 2024). KARMA is an example of a **multi-agent LLM framework** applied to knowledge graph enrichment, which involves schema alignment. Its Schema Alignment Agents are tasked with aligning entities and relations to KG schemas (Lu and Wang, 2025).

Taxonomy of Approaches

Based on the surveys and descriptions in the sources, schema matching approaches can be classified along several dimensions (Sutanta, 2016; Rahm and Bernstein, 2001; Kang and Naughton, 2003):

1. Level of Information Exploited:

- **Schema-level:** Uses only schema information (names, descriptions, data types, constraints, structure). Examples: Linguistic matchers, Constraint-based matchers, structural matchers like SF (when applied to schema graphs), Cupid (schema-based aspects), SMAT.
- **Instance-level:** Uses actual data values within the schema attributes (e.g., data patterns, distributions).
- **Auxiliary Information:** Uses external resources such as dictionaries, thesauri, taxonomies,

or ontologies.

2. Granularity of Matching:

- **Element-level:** Matches individual schema elements (e.g., attributes, columns, classes).
- **Structure-level:** Matches based on the relationships, context, or structure of schema elements (e.g., hierarchical structure, relational links, dependency graphs). Examples: Similarity Flooding, Opaque Matching (dependency graphs), structural aspects of Cupid.

3. Matching Criteria/Technique:

- **Linguistic/name-based:** Compares names and textual descriptions of schema elements.
- **Constraint-based:** Compares elements based on schema constraints like data types, keys (primary/foreign), uniqueness, and cardinality.
- **Graph-based:** Converts schemas into graphs and applies graph matching algorithms. Examples: Similarity Flooding, Opaque Matching.
- **Learning-based (ML/DL/LLM):** Uses machine learning, deep learning, or Large Language Models to learn matching patterns or generate matches. Examples: SMAT, ADnEV, KARMA (for schema alignment).
- **Statistical/Dependency-based:** Analyzes statistical relationships or dependencies between attributes, particularly useful for opaque data. Example: Opaque Matching.
- **Semantic:** Focuses on the meaning of schema elements and relationships, often leveraging ontologies or external knowledge. Example: iMAP (for complex semantic matches).
- **Reuse-oriented:** Leverages results from previous matching operations.

4. Combination Strategy:

- **Individual/Stand-alone:** A single matching algorithm is used.
- **Hybrid:** Multiple matching criteria or techniques are combined within a single algorithm. Example: Cupid.
- **Composite:** The results from several independently executed match algorithms are combined. Example: COMA supports this approach.

5. Automation Level:

- **Manual:** Matching is performed entirely by human experts.
- **Semi-automatic:** Tools assist human users by proposing candidate matches, which are then reviewed and adjusted by the user. Example: iMAP.
- **Automatic:** Algorithms attempt to find matches without human intervention. Often requires subsequent human verification and refinement.

6. Complexity of Matches:

- **Simple (1:1):** Matches are typically one-to-one correspondences between elements.
- **Complex (m:n):** Matches can involve combinations or transformations of multiple elements (e.g., concatenation, arithmetic). Example: iMAP.

7. Evolutionary Aspect:

- **Traditional:** Earlier heuristic, rule-based, or simple statistical methods.
- **Advanced:** Methods employing machine learning, deep learning, complex statistics, or LLMs.

Comparative Analysis of Characteristics

Comparing the main characteristics of traditional and advanced schema matching approaches based on the sources:

1. Traditional Methods (e.g., SF, Cupid, COMA framework, Linguistic, Structural, Constraint-based):

Strengths:

- Well-understood and foundational, providing the basis for more complex systems.
- Effective for common matching scenarios where names, structures, or constraints are consistent and informative.
- COMA provides a flexible framework for combining various matchers.
- SF is versatile for matching graph-like structures.

Weaknesses:

- Often struggle with significant schematic heterogeneity, including differing names, structures, and constraints across schemas.
- May fail when schema element names or data values are opaque or uninterpretable.
- Generally limited in their ability to discover complex, non-1:1 matches.
- Performance can be highly dependent on careful parameter tuning.

Degree of automation: Varying, often providing semi-automatic support by suggesting candidates. Purely automatic execution exists but results require verification.

Need for human feedback: Significant. Human experts are crucial for evaluating, verifying, and adjusting the proposed matches. This manual effort can be time-consuming.

2. Advanced Methods (e.g., ML/DL, LLM, Dependency-based):

Strengths:

- Can potentially capture more complex patterns and semantic relationships between schema elements and data.
- Capable of handling opaque or semi-opaque data where traditional linguistic methods fail.
- Show promise in tackling complex match discovery.
- Modern techniques like Deep Learning and LLMs have demonstrated strong performance on various text and data-related tasks. LLMs can potentially interpret complex natural language descriptions in schemas.

Weaknesses:

- ML/DL methods often require substantial training data, which may not always be available.
- Can be computationally expensive and require specialized hardware or expertise.
- LLMs require careful prompt engineering and can also be resource-intensive.
- Interpretability of results from complex models can be challenging.
- Some techniques are specialized for particular problems (e.g., opaque data, complex matches).

Degree of Automation: Generally higher in the process of generating matches.

Need for Human Feedback: Still present, though potentially shifted. Human feedback is essential for evaluating the quality of results, resolving ambiguities, refining the models or prompts, and handling cases beyond the model's capabilities. Improving user interaction mechanisms remains important.

Research Gaps and Challenges

The sources highlight several research gaps and challenges in schema matching (Rahm and Bernstein, 2001; Feng et. al, 2024, Kang and Naughton, 2003):

- **Lack of Consistent Superiority:** There is no single schema matching method that consistently performs better than others across all datasets and scenarios.
- **Need for Parameter Tuning:** Many algorithms require complex parameterization, and real-world performance suffers when parameters are not finely tuned. Auto-tuning remains an open problem.
- **Handling Complex Matches:** Most previous work has focused solely on 1:1 matches, neglecting the important class of complex matches that are prevalent in practice.
- **Combining Matchers:** How to best combine different match algorithms (hybrid and composite approaches) still requires further work. Fully utilizing the flexibility of composite approaches is a challenge.
- **Underutilization of Information:** More attention should be given to effectively utilizing instance-level information and opportunities for reusing previous match results. Exploring

semantic relationships at entities or instances as a basis for mapping is also a direction.

- **Opaque Data:** Schema matching with opaque column names and data values is a specific problem class that traditional methods do not address.
- **Improved User Interaction:** While human feedback is crucial, current approaches have limited means of effectively capturing user interaction. Designing comprehensive ways to incorporate user feedback is needed.
- **Generalizability and Domain Adaptation:** Developing generic matchers usable across application areas and adapting techniques to specific domains (like bioinformatics mentioned for COMA) or handling cross-domain matching remain relevant. The source mentions evaluating generalizability using benchmark datasets.
- **Addressing Schematic Heterogeneity:** Overcoming the problems caused by different naming, types, formats, precision, structures, and semantic interpretations in schemas is fundamental.
- **Handling Multilingual/Opaque Labels:** Schema elements or data values may be expressed in different natural languages (e.g., “ime” vs. “name” vs. “nombre”). Effective matching requires integrating translation dictionaries, domain-specific glossaries, or language models to bridge lexical gaps and recognize semantically equivalent labels across languages.
- **LLM Integration Challenges:** While promising, integrating LLMs requires considering costs, prompt engineering, and handling potential inconsistencies or errors inherent in generative models.

3.2.2 Theoretical Framework for Applying Schema Mapping Techniques

This section outlines a simplified, four-step cascade for *schema mapping*. Each step represents a core mapping strategy, moving from the simplest identity checks to full code generation. In practice, we will experiment with combinations of these steps and use initial tests to guide our direction.

1. Direct Mapping

If the values of the attribute match exactly, we can assume trivial "as-is" mapping between these attributes.

2. Static Lookup (Mapping Table)

A static lookup table provides a simple dictionary-based mapping. It is often used for value translations (e.g., code→description).

Description Maintain a predefined table (CSV, spreadsheet, or database) with two primary columns:

Source value	Target value
M	Male
F	Female

Advantages

- Very fast, constant-time lookup.
- High precision for known, stable mappings.
- Easy to review, update, and version-control.

Limitations

- Cannot handle unseen values.
- Requires ongoing maintenance as code lists evolve.
- Does not perform dynamic transformations (e.g., string concatenation or date-formatting).
- Requires mapping of a single application and a single midPoint attribute.

3. Heuristic Transformations

When identity mapping and static lookup both fail, we can apply a series of deterministic heuristics. These heuristics include a **normalization** step plus various string-, date-, and value-level functions. The examples below draw from built-in functions typically found in integration platforms such as Adeptia AIMap. If application and midPoint attribute values still don't match after identity or lookup, first normalize them by lowercasing, stripping diacritics, removing punctuation, and collapsing spaces; next, compute fuzzy-matching scores (e.g., Levenshtein distance or Jaccard token overlap) to catch near-misses; and finally, chain targeted transformations—string operations (substring, replace, normalize-space, case conversion), date functions (date-format, date-difference, current-date), math functions (add, multiply, ceiling/floor), aggregation functions (sum, count, average), conditional logic (if/when/selectQuery), axis/navigation functions for nested JSON/XML, boolean predicates, and context functions for metadata tagging—to systematically align even the most stubborn fields and values. In Adeptia AIMap, for example, this exact sequence is implemented as a best-practice reference and can serve as an inspiration.

Procedure

1. Normalize the attribute values.
2. If there is an exact match, or a fuzzy string similarity threshold is met, consider the mapping.
3. Otherwise, attempt to chain built-in functions to transform the values.
4. If nothing is found, escalate to LLM-assisted code generation (Step 4).

Advantages

- Built-in functions cover a wide range of common data transformations without writing custom code.
- Deterministic and transparent when pipelines are simple.
- Handles cases where minor variations exist (typos, abbreviations, punctuation).

Limitations

- Complex multi-input transformations can become unwieldy.
- Requires careful ordering and parameter tuning.
- Some transforms (e.g., joining multiple disparate fields) may still be out of reach.

4. LLM-Assisted Code Generation

When direct mapping, static lookups, and heuristic transformations all fail to produce a satisfactory mapping, a Large Language Model can be invoked to generate custom transformation code. This is particularly useful for scenarios with complex logic, multiple attributes at the source side, or domain-specific rules.

In this approach, the LLM receives column values for the attributes and produces code that maps between them.

Advantages

- Can express arbitrarily complex transformations (joins, lookups, loops, nested conditions).
- Adaptable to various output programming languages.

Limitations

- Potential for hallucinations—code may have subtle bugs or missing edge-case handling.
- Safety concerns: generated code needs to be checked that it's harmless.
- May introduce additional dependencies (third-party libraries).

3.2.3 Overview of Open-Source and Commercial Tools

In addition to the literature review presented above, we conducted a comprehensive survey of both open-source and commercial schema-matching tools. Below, we summarize our findings and provide an overview of these tools.

Table 1. Comparison of selected tools

Aspect	**FlexMatcher** [23]	**COMA++** [24]	**Adeptia AIMap** [25]	**KARMA** [26]	**FlatFile** [27]	**PSM** [28]	**PMFSM** [29]
Automation	Semi-automatic ML; needs labeled data	Semi-automatic, user-in-loop	AI/ML with LLM-assisted suggestions	ML + multi-agent LLM alignment	AI/ML with LLM-augmented NL interface	ML (embeddings + classifier)	Primarily LLM-driven
Scalability	Small–medium schemas (training-bound)	Large match jobs via workflows	Enterprise/cloud scale	Distributed LLM pipelines (high scale)	Enterprise/cloud scale	Moderate (local compute/embeddings)	Scales with LLM service limits
Data Requirements	50–100 labeled pairs	Schema metadata + optional instances	Historical mappings + sample files	Ontologies + sample data for LLM prompts	Billions of historical examples	Labeled pairs + embedding corpora	Few-shot/zero-shot with retrieval
Semantics	String & instance features (KNN, LR)	String + structural + ontology reuse	NLP + metadata + LLM inference	Ontology + embeddings + LLM reasoning	Embeddings + NLP + LLM for NL rules	Embeddings (sentence transformers, XGBoost)	Pure LLM semantic reasoning

Short Summary of Findings:

- **Commercial Platforms** (Adeptia AIMap, FlatFile) blend ML and LLM techniques to offer intuitive UIs, confidence scoring, and wide integrations, but are tied to proprietary licenses and historical-data quality.
- **Emerging Open-Source** projects (KARMA, PSM, PMFSM) experiment with transformer embeddings and LLM-driven mappings; they are innovative yet still maturing in popularity and documentation.
- **Classic Frameworks** (COMA/COMA++, FlexMatcher) remain the backbone for academic benchmarking, providing modular, extensible matchers based on string- and statistic-based heuristics, but require manual tuning and training data.

The **industry trend** is clearly toward richer AI-driven mapping workflows—leveraging embeddings, generative models, and multi-agent architectures to enhance automation, semantic depth, and scalability. We expect traditional methods to struggle with following the context and semantics of our tasks so we will focus on the newer methods (transformer based models).

3.2.4 Metrics

We will evaluate the recommendation system using the following metrics:

- **Confusion Matrix**: A contingency table reporting true positives (TP), false positives (FP), false negatives (FN) and true negatives (TN) for the predicted vs. actual matches.
- **Precision**: $\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$ the proportion of predicted matches that are correct.
- **Recall**: $\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$ the proportion of true matches that are recovered.
- **F-score**: The harmonic mean of precision and recall: $F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$.
- **Matthews Correlation Coefficient (MCC)**:

  $$\mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}$$

  a balanced measure that accounts for all four cells of the confusion matrix.
- **Accuracy**: $\frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{FP}+\mathrm{FN}+\mathrm{TN}}$ the fraction of correct predictions overall.
- **LLM Hallucination Count**:: The number of LLM-generated match proposals that do not correspond to any valid ground-truth mapping, indicating over-confident or spurious suggestions.

3.3 Correlation Recommendation

Correlation in midPoint is a mechanism to correlate identity data in an application to existing identities in midPoint. It works in two steps:

- First, data from the application are mapped (transformed) into midPoint format. For example, `first_name` in the application is mapped into `givenName` in midPoint. Or, `DOB` in the application is mapped into `extension/dateOfBirth` in midPoint, including e.g. date format conversion.
- Second, a set of *correlation rules* is applied with the goal of finding an identity in midPoint that matches the (transformed) application data. For example, a rule can state that "if the family name, date of birth, and the national-wide ID all match, then the identity is matching". Another rule can state that "if (only) the national-wide ID matches, then the identity is matching with the confidence level of 0.7". Rules can compare attribute values exactly, using some normalization (like ignoring case, diacritics, or whitespaces), or by using fuzzy matching.

For more information, please see the midPoint documentation on correlation.

The goal of this section is to research potential mechanisms for identifying correlator attributes and assessing their value comparison strategies. In this stage, we describe various theoretical approaches - ranging from predefined domain knowledge lists and lightweight pattern based heuristics to LLM assisted suggestions - and outline exact, normalized, string similarity, and phonetic□matching matching strategies. These descriptions serve purely as a conceptual design: actual implementations will be validated and refined later.

3.3.1 Finding Correlation Attributes

The first issue is to identify which attributes will serve for correlation. Let’s call them correlation attributes.

Predefined Attribute List (Domain Knowledge)

In many organizations, certain attributes are known a priori to serve as reliable correlation ones. These are attributes like `emailAddress`, `uid`, `employeeNumber`, or `cn`. One approach is to maintain a small, configurable lookup (CSV or database table) that lists common attribute names (and any synonyms) that should automatically be considered as correlation ones.

Advantages:

- Low false□positive rate when domain knowledge is correct.
- Extremely fast to check (constant□time lookup).

Limitations:

- Fails to detect less obvious correlators (e.g., `telephoneNumber`).
- Requires maintenance as new use cases arise.

Heuristic□Based Selection (Pattern Matching)

When an attribute name is not in the predefined list, we will consider lightweight heuristics to flag other potential correlation attributes. Examples include:

- Name patterns:: Check for substrings like `email`, `id`, `uid`, `login`, `user` (case□insensitive); or check for application-specific attributes, like `dn` or `cn` for LDAP; in HR systems, `employeeNumber` or `staffID` is frequently unique.
- Data type check:: If an attribute has a specific data type and/or value format (e.g., it is a string of length ≤ 254 and its format resembles an email or UUID), it becomes a candidate.

These heuristics assign a preliminary score (e.g., 0–1) to each candidate. Attributes scoring above a threshold (e.g., 0.7) are promoted to "correlation attribute" status.

LLM□Assisted Suggestion

If neither domain knowledge nor heuristics confidently identify a correlation attribute, we can invoke a lightweight prompt to an LLM to propose the likely ones. The prompt can include:

- Attribute descriptions or any available schema metadata (e.g., “emailAddress: user’s email”).

- Contextual note:: “Find the top attributes most likely to uniquely identify or correlate objects.”

The LLM’s response can be parsed into a ordered list.

3.3.2 Suggesting Correlation Rules

The other part of the solution is the determination of specific correlation rules: how should be the selected attributes used to find matching objects. Each rule references one or more correlation attributes, each with the description of the way of comparing its value while finding the matching objects (e.g., exact match, fuzzy match, or match on normalized data, like ignoring diacritics or whitespaces, case-insensitive match, and so on).

A typical approach consists of trying some of the following approaches. (They serve just for an inspiration here. More on selecting specific techniques is in Section 4.)

Exact Matching

This approach requires exact match of the correlation attribute values. A typical example is personal number.

NOTE Some attribute are inherently not suitable for exact matching. For example, many LDAP attributes have specified matching rule of `caseIgnoreMatch`, so they must be matched ignoring case, as described below.

Normalization-Based Matching

This is a comparison of normalized versions of the values. Examples:

- Ignoring case: "John.Doe@Example.COM" matches "john.doe@example.com".
- Ignoring whitespace at start and end of a value and collapse multi-spaces: " user@example.com " matches "user@example.com".
- Ignoring diacritics: "josé@example.com" matches "jose@example.com".

String Similarity Matching

In this approach, we compare string values approximately:

- Levenshtein distance: indicates how many operations (insertion/deletion/substitution) are needed to transform a value to another value.
- Jaccard token similarity: Split on common delimiters (@, ., _) and compute token overlap.

We consider a value to match if the similarity is equal or above a given threshold.

Phonetic Matching

For attributes like `fullName`, `displayName`, or even `name` (login name), where spelling variants or diacritical differences could occur, this approach computes and matches a phonetic code (e.g., Soundex or Metaphone) for both application and midPoint values. This is particularly useful for names (e.g., “Müller” vs. “Mueller”).

NOTE The human user should approve all recommendations for correlation rules, as there may be situations when automatically determined rule is too weak or too strong.

3.4 Baseline Experiments for Model Mapping/Matching and Correlation Recommendation System

In this section, we establish performance baselines for both schema-mapping and correlator-identification in tandem, focusing exclusively on the “user” entity type (which we find most complex) while planning to extend our approach to other types such as roles and associations. Rather than treating correlator discovery as a standalone experiment, we jointly evaluate schema matching and correlator identification on user schemas. In both Experiment 1 and Experiment 2, alongside schema matching, we also measure how well a simple embedding-threshold or a raw LLM prompt can recover our static, domain-standard correlator list (emailAddress, uid, employeeNumber, cn, etc.). We then proceed to Experiment 3, a proof-of-concept for Groovy code generation in schema mapping. Each experiment is accompanied by a clear description of its methodology, datasets, and evaluation metrics, allowing us to directly compare these basic baselines. We will compare open-source models against top SOTA commercial models, with the ultimate goal of demonstrating that an open-source solution can come close to, or even match, the performance of proprietary alternatives. We also plan to augment those AI-driven recommendations with existing mappings and metadata drawn from our integration catalog, to further improve accuracy and consistency.

3.4.1 Experiment 1: Embedding-based Empirical Threshold Optimization with Schema Matching and Correlation Prediction

Attribute-level schema matching is an inherently complex m:n retrieval problem. Each attribute in an external schema must be compared against every attribute in a reference schema, and correct correspondences often hinge on subtle semantic cues. Here, we focus on an **embedding-based** approach, mapping attribute names and descriptions into a shared vector space and using cosine similarity as our core signal. A subsequent experiment will explore complementary **LLM-based** strategies that leverage contextual reasoning.

Phases of the Embedding-Based Schema Matching

In this experiment, we evaluated the ability of embedding-based technologies to address increasingly complex tasks including:

- Identifying top k=1 matches for each attribute (1:1 matching),
- Identifying multiple potential matches for a single attribute (m:n matching),
- Narrowing the candidate space using static thresholding tailored to each model,
- Evaluating the success of discovered correlation matches.

This experiment represents the fourth iteration of our embedding-based schema matching pipeline:

- **v0 – 1:1 matching**: Initial phase with simple one-to-one attribute matching, based on a top-k thresholding approach.
- **v1 – m:n matching**: Introduced support for many-to-many attribute matching as we've identified scenarios where multiple suggestions per attribute are appropriate. This leverages using standard deviation (STD) and Z-score thresholding.
- **v2 – m:n with empirical thresholding**: Added empirically optimized thresholds to filter out low-confidence matches and improve accuracy. Trained on synthetically generated data.
- **v3 – correlator integration**: Final phase involving integration of correlation analysis to evaluate the identified correlation matches.

Each phase reflects the gradual improvement in both the accuracy and flexibility of the matching process, addressing limitations encountered in earlier versions.

Below we focus on the final version (v3), which represents the most current optimized procedure.

Embedding-Based Methodology

1. **Embedding Generation**
    - Convert each attribute's name and optional description into dense vectors using a variety of pre-trained models.
    - Four different approaches are used for embedding generation:
        - `property_embeddings`: Only the property values are embedded.
        - `description_embeddings`: Only the description values are embedded.
        - `combined_embeddings`: Both property and description values are embedded and combined by averaging them, with a configurable weighting factor (default 0.3 for property, 0.7 for description).
        - `linked_embeddings`: Property and description values are concatenated and then embedded together, allowing the model to learn from both simultaneously.
2. **Similarity Matrix Computation**

  - Compute cosine similarity for every external–reference attribute pair, forming an |external|×|reference| matrix.

3. **Empirical Threshold Optimization**
   - Sweep a grid of 500 thresholds in [0.0, 1.0].
   - For each threshold, binarize similarities (≥ threshold ⇒ match), compute TP, TN, FP, FN, then derive precision, recall, F1, accuracy, and MCC.
   - Select the threshold that maximizes F1 for each model–schema pairing.
4. **Schema Matching**
   - Apply the optimal threshold to extract candidate matches, rank them by similarity, and label each as correct or incorrect against ground truth.
5. **Correlator Analysis**
   - Use `evaluate_schema` to build two binary matrices per schema: one from predicted "Resource" matches and one from ground-truth "Correlator" flags.
   - Compute **corr_recall**: the overall recall across all correlator-flagged schemas, defined as

Parameter Selection

- **Embedding Approach**: `combined_embeddings` approach, which combines property and description embeddings with configurable weighting.
- **Property Weighting**: 0.3
- **Description Weighting**: 0.7
- **Threshold Optimization**: Empirical threshold optimization is performed for every source attribute against a synthetically generated dataset containing approx. **100 positive matches** and **3300 negative matches** per each source valid attribute.

Model Selection

We evaluated 16 models spanning diverse categories:

- **Multilingual**: sentence-transformers/LaBSE, intfloat/multilingual-e5-large-instruct, hkunlp/instructor-xl
- **Open Source LLM-Derived**: Alibaba-NLP/gte-Qwen2-7B-instruct
- **Sentence Transformers**: all-MiniLM-L6-v2, all-mpnet-base-v2, sentence-transformers/nli-mpnet-base-v2, paraphrase-MiniLM-L6-v2, all-distilroberta-v1, msmarco-MiniLM-L6-cos-v5, intfloat/e5-large-v2, BAAI/bge-large-en
- **T5-Based**: sentence-t5-xxl
- **OpenAI Embeddings**: text-embedding-ada-002, text-embedding-3-small, text-embedding-3-large

This selection ensures coverage of multilingual capacity, LLM-derived embeddings, popular sentence-transformers, and the latest OpenAI embedding engines.

Results

Below is the summary of performance metrics for each embedding model. Inference time is averaged over all schema pairs.

Model	MCC	Precision	Recall	F1	Hallucinations	Corr Recall	Inference Time Avg (s)
text-embedding-3-large	0.369	0.265	0.565	0.349	0	0.663	1.66
sentence-t5-xxl	0.366	0.264	0.563	0.345	0	0.685	12.427
hkunlp/instructor-xl	0.363	0.328	0.538	0.333	0	0.584	2.116
text-embedding-3-small	0.363	0.291	0.574	0.333	0	0.596	1.702
text-embedding-ada-002	0.349	0.245	0.557	0.327	0	0.573	1.496
all-distilroberta-v1	0.343	0.251	0.538	0.322	0	0.494	0.547
BAAI/bge-large-en	0.333	0.256	0.547	0.301	0	0.618	0.321
paraphrase-MiniLM-L6-v2	0.318	0.270	0.540	0.278	0	0.640	0.077
intfloat/e5-large-v2	0.301	0.233	0.506	0.272	0	0.562	0.562
all-MiniLM-L6-v2	0.298	0.182	0.564	0.264	0	0.629	0.604
all-mpnet-base-v2	0.293	0.189	0.562	0.259	0	0.551	0.727

Model	MCC	Precision	Recall	F1	Hallucinations	Corr Recall	Inference Time Avg (s)
msmarco-MiniLM-L6-cos-v5	0.276	0.189	0.472	0.256	0	0.539	0.078
sentence-transformers/nli-mpnet-base-v2	0.277	0.200	0.505	0.246	0	0.596	0.250
intfloat/multilingual-e5-large-instruct	0.270	0.188	0.510	0.242	0	0.551	0.754
sentence-transformers/LaBSE	0.257	0.200	0.424	0.240	0	0.404	0.495
Alibaba-NLP/gte-Qwen2-7B-instruct	0.083	0.052	0.430	0.076	0	0.382	4.560

Summary of Findings:

- **Top Performers**: The OpenAI embedding models (text-embedding-3-large, -3-small) achieved the highest F1 (0.349 and 0.333) and MCC (0.369 and 0.363).
- **Competitive Alternatives**: sentence-t5-xxl matched closely on F1 (0.345) but at a much higher inference cost (~12 s per schema pair).
- **High Precision vs. Latency**: hkunlp/instructor-xl led on precision (0.328) with moderate speed (~2.1 s), suggesting a good trade-off for precision-sensitive tasks.
- **Fast & Lightweight**: paraphrase-MiniLM-L6-v2 and msmarco-MiniLM-L6-cos-v5 offered very low latency (<0.1 s) with respectable recall but lower precision and F1.
- **Multilingual & LLM-derived**: Multilingual models (LaBSE, e5-large-instruct) showed solid recall but moderate overall F1; the large LLM-derived gte-Qwen2-7B lagged in both performance and speed.

3.4.2 Experiment 2: LLM-Based Empirical Prompt-Driven Schema Matching and Correlation Prediction

Building on our embedding-based retrieval, Experiment 2 uses large language models (LLMs) to directly reason over attribute pairs and decide matches via structured prompts. We also track correlator flags and parsing errors to measure robustness.

1. **Prompt Design & Inference**
    - Construct a JSON-based prompt containing external attribute name/description, reference attribute name/description, and example match labels.
    - Send the prompt to each LLM and parse its JSON response into binary match/non-match predictions.
2. **Batch Processing & Error Handling**
    - Process attribute pairs in batches per schema to optimize throughput.
    - Count **invalid_jsons** when the model's response cannot be parsed as valid JSON.
    - Record **hallucinations** for any predicted matches not present in ground truth.
3. **Evaluation & Correlator Analysis**
    - Use `evaluate_model` (with `include_correlator=True`) to aggregate per-schema results into model-level metrics.
    - Compute standard metrics (MCC, precision, recall, F1), plus:
    - **corr_recall**: overall recall on correlator-flagged matches.
    - **invalid_jsons** and **hallucinations** as described above.

Quantitative Results

Model	MCC	Precision	Recall	F1	Hallucinations	Corr Recall	Invalid JSONs	Inference Time Avg (s)
google/gemini-2.5-flash-preview-05-20	0.662	0.672	0.669	0.663	31	0.82	0	3.254
openai/o4-mini	0.660	0.678	0.666	0.659	0	0.798	0	22.928
anthropic/claude-sonnet-4	0.652	0.620	0.707	0.651	10	0.809	1	7.161

Model	MCC	Precision	Recall	F1	Hallucinations	Corr Recall	Invalid JSONs	Inference Time Avg (s)
qwen/qwen3-14b	0.643	0.719	0.592	0.639	16	0.685	0	31.742
meta-llama/llama-4-maverick	0.640	0.684	0.623	0.637	19	0.753	0	6.431
deepseek/deepseek-r1-0528	0.640	0.660	0.637	0.639	0	0.753	1	97.709
x-ai/grok-3-mini-beta	0.622	0.608	0.670	0.618	2	0.719	0	13.251
qwen/qwen3-32b	0.606	0.646	0.592	0.603	21	0.640	0	55.972
qwen/qwen-2.5-72b-instruct	0.600	0.652	0.580	0.598	82	0.730	0	12.058
meta-llama/llama-4-scout	0.589	0.730	0.495	0.575	19	0.629	1	6.458
google/gemini-2.5-pro-preview-05-06	0.586	0.406	0.897	0.546	0	0.921	0	70.486
openai/gpt-4.1	0.582	0.495	0.721	0.573	69	0.854	0	7.816
qwen/qwen-2.5-coder-32b-instruct	0.579	0.564	0.629	0.573	257	0.674	1	12.792

Model	MCC	Precision	Recall	F1	Hallucinations	Corr Recall	Invalid JSONs	Inference Time Avg (s)
google/gemma-3-27b-it	0.539	0.663	0.458	0.530	190	0.685	0	20.429
microsoft/phi-4	0.486	0.555	0.461	0.478	255	0.640	1	13.153
mistralai/devstral-small	0.470	0.433	0.563	0.457	225	0.640	3	9.911
mistralai/mistral-nemo	0.334	0.336	0.368	0.331	177	0.449	6	9.153
microsoft/phi-3.5-mini-128k-instruct	0.025	0.027	0.027	0.026	25	0.045	20	2.713

Summary of Findings:

- **Best Overall**: google/gemini-2.5-flash-preview-05-20 achieved the highest F1 (0.663) and MCC (0.662) with moderate latency (3.25s), but exhibited a notable number of hallucinations (31).
- **Strong Precision**: openai/o4-mini led on precision (0.678) with robust recall (0.666) and zero parsing errors.
- **Highest Recall**: google/gemini-2.5-pro-preview-05-06 achieved exceptional recall (0.897) but suffered in precision (0.406) and latency (70.5s).
- **Latency vs. Accuracy Trade-Off**: lighter models like anthropic/claude-sonnet-4 offered balanced performance (F1 0.651) with lower latency (~7s).
- **Parsing Robustness**: most LLMs produced zero invalid JSONs, though some large-coder and smaller open-source models had parsing failures (e.g., phi-3.5-mini).
- **Correlation Performance**: highest corr_recall was observed for google/gemini-2.5-pro-preview-05-06 (0.921) and other top LLMs (>0.80).

3.4.2.1 Experiment 2 (Continued): Multilingual Data Evaluation

To assess the robustness of our prompt-driven LLM pipeline across languages, we translated the original English schemas into eight target languages:

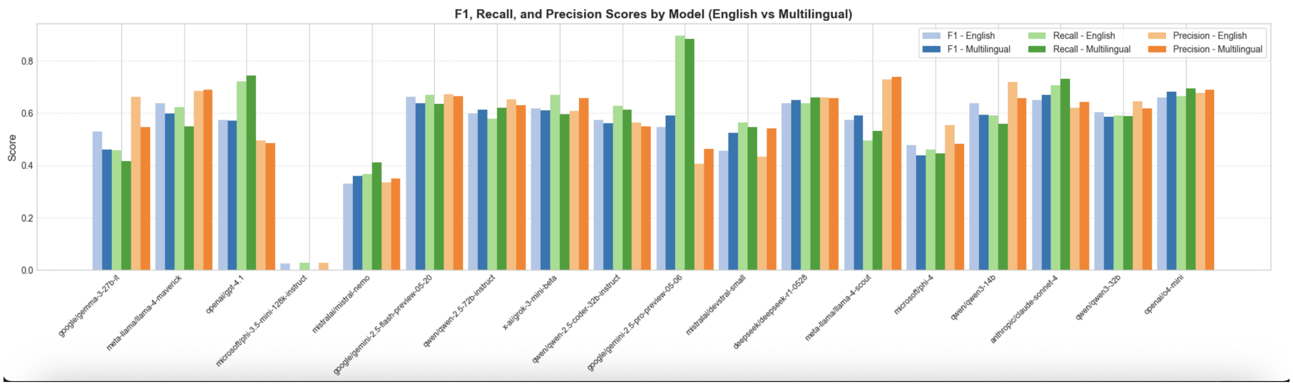
- Slovak: 7 schemas

- French: 4 schemas
- German: 3 schemas
- Portuguese: 2 schemas
- Spanish: 2 schemas
- Italian: 2 schemas
- Chinese: 2 schemas
- Japanese: 1 schema

All 23 multilingual schemas were processed with the same JSON-based prompt format as in the English experiment, replacing attribute names/descriptions with their localized equivalents.

model	mcc	precision	recall	f1	hallucinations	corr_recall	invalid_jsons	inference_time_avg
openai/o4-mini	0.683	0.689	0.695	0.681	2	0.864	0	23.716
anthropic/claude-sonnet-4	0.673	0.642	0.731	0.670	3	0.886	0	8.054
deepseek/deepseek-r1-0528	0.649	0.657	0.661	0.649	31	0.818	0	101.360
google/gemini-2.5-flash-preview-05-20	0.639	0.664	0.636	0.639	40	0.841	0	3.499

As illustrated in Image 1, the F1, recall, and precision scores remain largely consistent between English and multilingual inputs across all evaluated models.



Qualitative Analysis:

Based on inspection of true positives, false positives, and false negatives for a representative LLM (e.g., google/openai/gpt-4.1), we observed:

1. **Multi-Mapping Dependencies**
    - Ground truth includes three mappings for **organization**:

    ```
    organization → customerId
    organization → orgUnitPath
    organization → organizations
    ```

    - But the LLM mapped `costCenter → customerId` and `orgUnitPath → organizationalUnit`, surfacing valid relationships that the annotations omitted.
2. **Naming Alias Mismatches**
    - Ground truth: `emailAddress → icfs:name`
    - The LLM mapped `emailAddress → email`, showing correct semantic alignment despite differing labels.
3. **Unannotated Valid Matches**
    - The LLM predicted `telephoneNumber → phones`, a logical correspondence absent from ground truth, indicating annotation gaps.

Data-Quality Implications

These examples expose annotation and naming inconsistencies. Harmonizing aliases, expanding valid mapping sets, and normalizing attribute names would better reflect model capability and improve practical performance.

Summary

- The embedding-based approach (Experiment 1) underperformed:
    - Best model (text-embedding-3-large) achieved F_1≈0.35, MCC≈0.37, latency ~1.7 s, zero hallucinations.
- The LLM-based approach (Experiment 2) outperformed embeddings by a wide margin:
    - google/gemini-2.5-flash-preview-05-20: F_1≈0.663, MCC≈0.662, latency ~3.3 s, 31 hallucinations.
    - openai/o4-mini: highest precision (0.678), recall≈0.666, zero parsing errors.
- Multilingual robustness (Experiment 2.1):
    - All top LLMs processed 23 translated schemas across 8 languages with an average F_1 drop of < 0.02.

    - openai/o4-mini slightly **improved** on multilingual data (F_1 0.681 vs. 0.659 English).
    - google/gemini-flash maintained F_1 within 0.02 points (0.663 → 0.645).
    - No invalid JSON remained 1.0 for all models on multilingual inputs.
- **Key conclusion**:: State-of-the-art LLMs clearly surpass embedding methods in precision, recall, F_1 and overall robustness.

Next Steps

1. Switch to an **exclusively LLM-based pipeline**
2. Perform **data cleansing and normalization** based on qualitative findings
3. Select primary model
4. Refine **prompt engineering**
    a. Experiment with JSON-structured prompts and few-shot examples
    b. Implement fallback heuristics for any parsing failures
5. Build and deploy an **end-to-end prototype**
    a. Integrate LLM calls into both model-mapping and correlator selection modules
    b. Capture user feedback (accept/override) to inform and improve subsequent versions of our matching and correlator-identification algorithms

3.4.3 Baseline Groovy Code Generation for Schema Mapping

As part of the initial exploration phase, we conducted a series of baseline experiments focused on schema mapping – the task of transforming structured input attributes into a target textual format, based on a small number of examples. The goal was to evaluate whether modern LLMs (including SOTA and open-source models) are capable of inferring transformation logic purely from few-shot examples, without task-specific fine-tuning.

The primary aim was to test LLMs' ability to:

- Interpret the semantics of synthetic but realistic input samples
- Generate transformation functions in Groovy format (`def transform(context) { … }`)
- Produce output matching the expected `target` string based on a test `source` dictionary
- Run the generated function in a secure Groovy sandbox and compare the output with the expected one

The models were not trained on the task and relied solely on prompt instructions and examples.

Dataset

The evaluation dataset consists of small synthetic samples derived from real-world configurations (e.g., inbound/outbound mappings). Care was taken to select realistic yet safe samples (i.e., no sensitive business data). Each row contains:

- `source`: A dictionary with input attributes
- `target`: The expected output string after transformation

The inputs contain only attributes relevant to prediction, simplifying the task compared to production use-cases, where attribute filtering or preprocessing may be necessary.

Evaluation Procedure

Each LLM was prompted with few-shot examples and asked to generate a `transform(context)` function in Groovy. The function was:

1. Executed in a Groovy sandbox on the provided input context
2. Its return value was compared to the expected `target` string (after normalization)
3. The results were categorized into `correct`, `incorrect`, and `error` (e.g., broken syntax)

We collected runtime information (duration and latency) for each model to evaluate efficiency.

Results

The following table summarizes the model performance:

Model	Success Rate	Correct	Incorrect	Errors	Avg Duration (sec)
openai/o4-mini	0.90	27	2	1	14
openai/gpt-4.1	0.87	26	4	0	2
anthropic/claude-sonnet-4	0.83	25	5	0	3
qwen/qwen3-14b	0.80	24	5	1	4
x-ai/grok-3-mini-beta	0.77	23	5	2	7
google/gemini-2.5-pro-05-06	0.77	23	6	1	30
google/gemini-2.5-flash-preview-05-20	0.77	23	5	2	4
deepseek/deepseek-r1-0528	0.77	23	6	1	52
qwen/qwen3-235b-a22b	0.73	22	5	3	67
qwen/qwen3-32b	0.73	22	5	3	55
qwen/qwen2.5-72b-instruct	0.73	22	7	1	3
meta-llama/llama-4-scout	0.70	21	8	1	2

Model	Success Rate	Correct	Incorrect	Errors	Avg Duration (sec)
google/gemma-3-27b-it	0.70	21	7	2	4
meta-llama/llama-4-maverick	0.67	20	8	2	1
qwen/qwen2.5-coder-32b-instruct	0.63	19	10	1	2
mistralai/dev-4-small	0.63	19	8	3	2
microsoft/phi-4	0.57	17	9	4	2
mistralai/mistral-nemo	0.33	10	11	9	2
microsoft/phi-3.5-mini-128k	0.17	5	7	18	8

Observations

- Both state-of-the-art (SOTA) and smaller open-source (OSS) models demonstrated solid performance on this task.
- Smaller OSS models like **qwen3-14b** and **gemma-3-27b-it** achieved competitive results compared to top proprietary models.
- Surprisingly, `qwen3-14b` outperformed `qwen3-32b`, a trend also seen in earlier experiments, suggesting better efficiency-latency trade-off.
- SOTA models such as GPT-4.1 and Claude-Sonnet-4 performed as expected with minimal or no sandbox errors.
- Variability in results was observed between runs due to non-deterministic generation, especially with open-source models.
- Execution latency varied widely, with large reasoning models averaging 1+ minutes, which can impact scalability. However, infrastructure differences (e.g., server vs. local deployment) make direct comparison imprecise.

Summary

- LLMs can successfully infer schema transformations from few-shot samples even without domain-specific fine-tuning.
- Results show that open-source models are viable options for deployment in cost-sensitive environments.
- Careful prompt design and consistent validation improve reliability and reduce errors.
- Evaluation on synthetic, semi-realistic data provides valuable insights, but further testing on full samples is needed.

Next Steps

We identified several directions for future improvement:

- **Data**: Curate more evaluation data with realistic complexity; define which transformations are in-scope
- **Model usage**: Evaluate latency and consistency across OSS models on self-hosted infrastructure
- **Prompting**: Explore reflection loops and retry strategies for failed transformations
- **Validation**: Develop automated test harness for scoring transformations across model versions

3.4.4 Global Summary & Next Steps

Global Summary

Across our baselines, embedding-based schema matching peaked at F□≈0.35 (MCC≈0.37) and recovered all “obvious” correlators via a fixed threshold, while LLM-based matching more than doubled performance—hitting F□≈0.66 (MCC≈0.66) with robust correlator recall. Our proof-of-concept Groovy code generation also demonstrated that modern LLMs, even without fine-tuning, can correctly infer transformation logic ~80–90% of the time.

Next Steps

- **Adopt an LLM-Centric Pipeline**:
    - As the experiments show, using LLM for both schema matching and correlators suggestions outperformed embedding-based approaches. Therefore, we will focus on LLM-based approaches in the next steps. We will select the best-performing LLM and use it for both schema matching and correlator suggestion. In this step, only data sent to LLM are schemas describing attributes in application and midPoint, no personal data are used here.
    - Despite the very good performance of modern LLMs at generating transformation scripts, concerns remain—hallucinations, security and compliance issues around sending personally identifiable information, and regulatory constraints. To mitigate these, we’ll first develop and integrate supporting algorithms (data normalization, heuristic rules, alias mapping, etc.) for suggesting mappings; only if they yield no satisfactory candidate will we optionally invoke the LLM. This staged approach reduces noise, preserves privacy, and confines LLM use to genuinely complex case
- **Algorithmic Pre-Processing**: Develop and integrate supporting algorithms like data normalization, heuristic rules, alias mapping, etc. to reduce noise and improve input quality before prompting.
- **Core Algorithm Tuning**: Experiment with different prompting strategies, prompt formats, and where feasible, model fine-tuning.

- **Prototype End-to-End Integration**: Build a minimal working system integrating LLM calls into mapping and correlation modules.
- **Expand Evaluation**: Test on larger, more diverse real-world schemas and mappings; include semantic edge cases and complex multi-field correlators.
- **Validation & Monitoring**: Integrate XML/Groovy validators into the pipeline and set up observability to track hallucinations, parsing errors, and user overrides.
- **Iterate with User Feedback**: Deploy early to a pilot group, collect accept/override actions, and feed them back into prompt tuning and model fine-tuning cycles.

3.5 Delineation

Delineation is the process of clearly defining the boundaries and criteria by which a system classifies or segments objects within its data source. In midPoint this is configured via a single `<delineation>` element inside each `<objectType>` definition. It determines:

- **Which Resource Objects** (e.g., LDAP entries) belong to a specific type (such as `accounts`, `groups`, or `entitlements`).
- **How and Where** these objects should be searched (e.g., filtering by `objectClass`, `baseContext`, organizational unit paths, or custom filters).

The goals of delineation are to ensure that:

1. **Input Data** is accurately segmented according to its intended purpose (for example, regular vs. admin vs. test accounts, all of which share the same objectClass but must be kept separate for policy reasons).
2. **Processing Flows** operate on the correct subset of objects (for instance, synchronizing only those that meet specific criteria).

3.5.1 Experiments on Delineation Rule Discovery

To evaluate how reliably our system—and even a non-expert—can infer midPoint delineation rules, we conducted two complementary experiments:

Experiment 1: Single developer inference without domain knowledge: Determine whether a developer with no prior midPoint or identity governance domain knowledge could reconstruct the same `<delineation>` criteria as an expert.

Experiment 2: Statistical profiling and LLM-assisted rule generation: Use data-driven insights to guide the creation of delineation rules and test whether an LLM, given these insights, can generate rules comparable to expert specifications. For each `objectClass`, we computed:

1. **Cardinality & Unique-Value Ratios** for all attributes
2. **Missing-Value Counts** to identify low-information fields

3. **Top Value Counts** for low-cardinality attributes
4. **Prefix/Suffix Distributions** of text columns
5. **Frequent 2-grams and 3-grams** in attribute values
6. **Numeric Quantiles & Histograms** for numeric fields
7. **Data Pattern Distributions** (digit-only, alpha-only, alphanumeric, hex-only)

The statistical data is computed in the midPoint itself and then sent to the smart integration service. Some of these statistical data can contain personal information. There will be an opt-out mechanism to avoid sending them out. We will explore possibilities how to suggest delineation without them.

We also generated global crosstabs between `objectClass` and the lowest-cardinality attributes to highlight strong alignments.

LLM Prompt & Pipeline A Markdown report containing the above statistics was passed to an LLMChain (model: qwen/qwen3-32b) with the instruction: *"Based on the provided statistics, generate detailed delineation rules for each* `objectClass`*. Each rule should include identification criteria, value-based classification rules, location-based filters, and any additional conditions."*

3.5.2 Results & Next Steps

In our experiments, we found that basic delineation rules could be discovered successfully in the majority of cases in both experiments. Some rules mapped directly to expert annotations, while others reached the same outcomes by applying different rules. However, discovering business-specific delineation rules proved too reliant on domain context to infer reliably from data alone. As a result, our evaluation pipeline in its current form cannot depend exclusively on raw delineation annotations; some annotations must be simplified standardized, or supplemented with expert knowledge to become automation-ready. We also observed that similar schemas often diverge in practice. For example, one AD resource might use a `groupType` attribute to separate security groups from distribution groups, whereas another uses entirely different conventions for this. Based on this we need evaluate delineation rules both quantitatively - measuring how often objects landed in the correct ObjectType, and qualitatively - checking whether the rules rested on the real business ideas, not on accidental data characteristics.

Also, we observed that the LLM sometimes focused on mutable fields, such as ‘manager’, which are unsuitable for stable delineation; true delineation attributes should be immutable over time. We will address this through prompt engineering. The next phase will involve consulting a domain expert to capture their “go-to” criteria and audit processes, enabling us to refine our analytical steps and prompt designs.

We acknowledged that hidden business rules cannot be assumed and must be encoded explicitly via best-practice heuristics or elicited from domain experts. Although the task remains challenging, in areas of data quality, evaluation design, and algorithmic complexity, we concluded that the overall approach is viable when expectations are grounded in domain realities.

3.6 UX Research Approach

This chapter focuses on research in the areas of UI and UX for the more complex components of the project—specifically, the connector generator and interactions with the AI assistant within the MidPoint Studio environment.

The goal of this research is to identify user needs, design challenges, and potential improvements in usability and user interaction. These insights will help inform the development of intuitive and efficient workflows, especially in parts of the system that involve dynamic content generation and intelligent assistance.

3.6.1 Code Generator

We carried focused UX research with an eye toward how users interact with AI-assisted configuration tools in order to help to create a user-friendly and effective connector generator. N atural language input, automated inference, and structured configuration combined to create special design problems that needed careful study.

Our method integrated heuristic assessments, internal design observations, and a comparison of newly developing UX patterns in generative artificial intelligence systems. Across a multi-step wizard-like flow, we investigated how to lower user friction, improve openness, and preserve a clear sense of control and progress.

The motivation behind the research, important challenges found, pertinent AI UX field findings, and how these realizations affected our design decisions for the Web UI and the Studio plugin are compiled in this chapter.

UX Motivation and Context

The introduction of AI into the connector generation process significantly transformed the way users interact with configuration tools in midPoint. Unlike traditional wizards with predefined, deterministic steps, the AI-enhanced approach interprets partial user input and dynamically generates elements such as scripts, mappings, or endpoint structures.

AI assistance within configuration tools creates a fundamentally different interaction model — one where users no longer manually define each detail, but instead guide the system using intent-level inputs (e.g., prompts or incomplete forms). This shift introduces several key challenges:

- Preserving user orientation and process clarity when the system dynamically injects AI-generated steps.
- Avoiding the “black box” effect by making AI actions transparent and explainable.

- Balancing structured inputs (forms, tables, selection cards) with AI-driven suggestions and auto-fill derived from documentation and metadata.
- Reducing user anxiety around unpredictable or partially automated outcomes.
- Integrating natural language inputs alongside structured configuration controls and task descriptions.
- Designing for trust by enabling users to easily override or correct AI suggestions.
- Seamlessly integrating error-handling steps into the wizard flow without disrupting user momentum.
- Supporting interruption recovery and draft continuity to help users pause and resume complex configurations without losing context.

Key UX/UI Challenges and Insights

The development of the connector code generator revealed several UX challenges, both observed during internal prototyping and supported by research in human-AI interaction. This section outlines specific problems encountered in the user interface layer of the multi-step wizard and describes key decisions made to improve clarity, control, and user trust.

1. **Flow and User Orientation**

   Challenge: Users often lost orientation during configuration when parts of the UI were suddenly filled by the AI. Without clear context, they were unsure whether they had completed a step, whether the AI had acted, or if further user action was needed. This contradicts the UX principle of visibility of system status, leading to lower confidence and increased hesitation.

   Observations: Users rely on linear progression and will confused if the system populates fields unexpectedly [30], users expect clear status indicators and predictable behavior, sudden jumps or unlocked steps may break user focus and lead to missed actions, and lack of visual differentiation between AI-generated and user-entered values may reduce trust.

   Design Decisions and Recommendations

   - Introduce a stepper component with visual progress and active step status
   - Mark AI-suggested fields with subtle labels and info icons
   - Add tooltips or light callouts explaining system-generated content [31]
   - Enforce a straight linear flow, preventing premature step access or skipping

2. **Dynamic Step Insertion and Flow Continuity**

   Challenge: Users may lose confidence if the flow appears inconsistent or unpredictable. New steps being added "on the fly" can lead to confusion about where they are, why a step exists, or whether they missed something earlier.

Observations: Users expect a linear and predictable flow in wizards, percieve dynamically inserted steps create anxiety or friction if not explained clearly, misunderstand the process logic and get to cognitive overload when steps are skipped or injected, and feeling about completness may be reduced when dynamic UI breaks ("did I really finish this part?").

Design Decisions and Recommendations

- Group steps by category (e.g., Basic settings, Connection, Object Types) to maintain mental model [32]
- Implement light transitions or highlights to signal that a new step was added
- Ensure that drawer prompt assistant is available in complex steps to reduce perceived friction [33]

3. **Documentation Relevance and Uncertainty**

   Challenge: Users may be unclear how documentation affects the generator or whether issues are caused by documentation or system limitations.

   Observations: Users may interpret AI silence as failure and overloaded documentation (too many sources) may reduce precision [34].

   Design Decisions and Recommendations

   - Provide contextual warning when too many sources are added without structure [35]
   - Display basic info about each source (e.g., format, parsing success, number of endpoints found, quality, etc.)

4. **Object Type Interpretation and Overload Risk**

   Challenge: Users may not know what each object type represents, especially when labels are vague. If many options are shown without explanation, decision paralysis or incorrect selection may occur.

   Observations: Too many object types with unclear or missing context can cause cognitive overload [36], users might not realize they can return and configure other types later, and get stuck at configuration if unsure from description which type is critical [37].

   Design Decisions and Recommendations

   - Use card layout with short, descriptive subtitles per object type [38]
   - Keep the list short (4-5 items) and use the ones with the strongest confidence only
   - If detection is weak, showing AI confidence level about found results would likely to improve trust and decision speed [39]

5. **Trust and Progressive Disclosure Across the Flow**

Challenge: Overconfident automation or missing validation lowers trust. Users need clarity on what was suggested, confirmed, and what remains editable or revertible.

Observations: Users feel uneasy about black-box generation, especially in security or schema steps [40], and the lack of editing options reduces their perceived control [41].

Design Decisions and Recommendations

- Use progressive disclosure to hide advanced options until needed
- Store version history or allow backtracking in critical steps (e.g., schema editing)

6. **Saving, Drafts and Interruption Recovery**

   Challenge: When supporting saving at any point, there is no persistent mental model for what's saved, what's incomplete, or what will happen when user returns.

   Observations: Users often forget what they've already done and tend to restart the flow instead of continuing a saved draft [42]

   Design Decisions and Recommendations

   - Introduce draft status indicators (e.g. "Schema not completed", "2 object types pending")
   - On re-entry, consider adding/showing summary about what is done and what is remaining
   - As the process may take long time to finish, consider auto-save function in unplanned occasions

7. **Two-Column Layout Overload in Complex Steps**

   Challenge: Too many editable fields, AI suggestions, and controls in a two-column layout can overwhelm users and harm clarity.

   Observations: Dense layout hides key elements (e.g., validation, buttons) and poor responsiveness on medium screens increases scrolling

   Design Decisions and Recommendations

   - Use progressive disclosure (collapsibles, tabs)
   - Split complex tasks into smaller chunks [43]
   - Apply asymmetric layout (e.g., 2/3 config, 1/3 context) [44]

8. **Lack of Customization for Technical vs. Non-Technical Roles**

   Challenge: Users have opposing needs: some prefer a quick, guided flow, others need full control and insight into AI behavior.

Observations: Technical users will be frustrated by the limitations of the AI (lack of display of raw details) and less experienced users will be frightened by the terminology or the possibility of "messing everything up" [45]

Design Decisions and Recommendations

- Add options for advanced user for discovering more details like (How and why this decision for specific step was made by AI)
- Add functions like "undo" or "revert previous action" where it is possible
- Add and adjust task information to be more descriptive where possible [46]

9. **Structured Input vs. Rich Context**

Challenge: Simple forms may lead users to underestimate their importance, resulting in vague or skipped inputs and weaker documentation matching.

Observations: Users often ignore or vaguely fill fields like Description, assuming they're optional [47] — especially since this step gives no immediate feedback. Adding suggestions or prompts too early may reduce approachability, and as shown in a Google x DeepMind study, unguided free text can lead to overthinking.

Design Decisions and Recommendations

- Keep structured input fields (e.g., Name, Description) with strong step description text explaining their purpose
- Avoid full prompt concept at initial stage and allow natural-language interactions to later phases for correcting AI outputs
- Postpone any AI-driven assistance to the next step (documentation selection), where the input is used to suggest links

10. **Drawer vs. Main Flow Conflict**

Challenge: The AI drawer may feel disconnected or compete with main content, confusing users about when and how to use it.

Observations: Users overlook it unless the drawer is clearly triggered, visual hierarchy issues arise when it resembles a separate app [48] and chat may reference elements that aren't visible on screen

Design Decisions and Recommendations

- Anchor drawer to specific steps, not globally.
- Usage of semantic headers in the drawer (e.g., "Schema Assistant: User object") should be applied to improve its understandability [49]
- Avoid chat-style full-width responses.

11. **Inconsistent Component Styles Across Steps**

    Challenge: Components like buttons, headers or cards may behave differently across steps (e.g., different spacing, icon usage, button alignment), which disrupts visual rhythm.

    Observations: Inconsistent styling lowers learnability (users must re-learn patterns) [50], and reduces trust in the UI's quality [51].

    Design Decisions and Recommendations

    ◦ Define and reuse UX micro-patterns (e.g., selection cards, confirmation blocks)
    ◦ Validate consistency in header typography, divider spacing, and component stacking

12. **Wizard Memory and Navigation**

    Challenge: Users are unsure what changes when navigating back — they fear losing AI output or overwriting manual input [52].

    Observation: This fear leads to reluctance to edit or total resignation [53]. Users may unintentionally destroy progress without warning.

    Design Decisions and Recommendations

    ◦ Show contextual notes on backward navigation (e.g., re-fetching clears endpoint) [54].
    ◦ Visually indicate which data is AI-controlled or at risk of change (e.g., colored borders around AI-controlled sections).

Design Decisions and Patterns Selected

Based on the challenges identified during the design and testing of the connector code generator, several UX principles and interface patterns were consciously applied to ensure clarity, user control, and confidence in the AI-assisted configuration process. The goal was to support a dynamic yet understandable user flow while maintaining flexibility for both technical and non-technical users.

Layout

Usage of a two-column layout with an integrated drawer for additional support actions — a natural and non-intrusive structure that supports prompt-assisted editing, preserves context, enables iteration and feedback, and remains scalable [55]

- Splitting panel/layout into 3 main regions [56]
  ◦ **Navigation and contextual overview** – A narrow left-hand section providing orientation and indicating the user's current position in the process
  ◦ **Main interactive area** – The central and dominant section dedicated to completing core tasks and interactions

    - **Supplementary drawer** – A right-hand, collapsible panel that offers additional tools, explanations or any other functionality to support user or current step without disrupting the main flow

- Usage of prompt like chat as tool inside drawer to allow additional form of correcting/using AI
- Potential usage of AI as tool inside drawer for further task details (e.g. providing help for user where to find something/for elaboration)
- Centered main flow with generous padding/margins to highlight main task to help reduce visual stres and enhance readability and focus [57]
- Clear structural design for gathering content using four key presentation types:
    - **Tabular** – For structured data which needs table to work with [58]
    - **Selection cards (decision skeletons)** – Used in steps where the user must choose between strategic paths or configuration directions. Cards represent distinct decision options, helping users quickly compare and commit to a direction (e.g., choosing a connector type, integration method, or template variant)
    - **Form fields** – For direct input values with AI-powered pre-fill, enabling quick completion of standard configuration parameters such as names, endpoints, or identifiers
    - **Tabbed views for complex content** – For displaying and editing more advanced or multi-faceted outputs, such as scripts alongside their resulting attribute structures or request payloads
- Contextual callouts will be used within the dynamic area to indicate AI actions, such as auto-filled values, success/failure of system operations, or important system feedback — helping users stay informed without disrupting the flow

Navigation and Progress

- Use of grouped steps for major configuration phases (e.g., "Connection", "Schema", "Operations", "Object type", etc.)
    - Once all sub-steps within a group are completed, the group collapses into a single summarized step in the navigation to maintain a clear high-level structure
    - Each collapsed step includes a status indicator (e.g., "Completed", "In Progress") to help users track their progress
    - Reopening the grouped step reveals all inputs in a consolidated view, enabling efficient review and edits while supporting full traceability and reusability
    - Reduction of navigation depth to a maximum of two levels to minimize cognitive overload and improve clarity
- To preserve flow continuity, navigation progresses only forward; if an issue arises, a fix step is dynamically inserted rather than navigating backward

- Backtracking is avoided (but allowed) in favor of corrective steps that appear in-line when needed, keeping the user in a guided forward path

3.6.2 MidPoint Studio Plugin

Research within the midPoint studio plugin focused on the possibilities of integrating AI tools into development environments. In our case, we focused on the IntelliJ IDEA development environment from JetBrains. We focused on existing open source plugins with the ability to be extended for our purposes.

The possibility of implementing process wizards and dialog windows in the IntelliJ IDEA development environment was also investigated.

This assistance concept is required in the development environment to support the integration of connector generation and configuration suggestions provided by the AI assistant.

MidPoint Studio AI Assistant

The research focused on existing AI tools and assistance in development environments that are freely available. The research process is described in the following points.

- Implementation of AI assistant in IntelliJ IDEA: AI Assistance for midPoint studio plugin
    - At this point, we focused ourselves on the conceptual design and prototypical integration of an AI assistant, into the midPoint studio plugin for IntelliJ IDEA. The goal was to increase the productivity of administrators and system integrators by providing recommendations for configuration mappings, correlation rules, and overall midPoint development tasks through intelligent, context-aware code suggestions.
- Best Practices for Integration and UX Control
    - Effective integration of AI assistant of midPoint studio plugin requires close alignment with established developer workflows in IntelliJ IDEA. Following GitHub Copilot's example, UX design must minimize disruption, offering contextual assistance without obstructing the editor interface. This includes leveraging inline code completions, ghost text suggestions, and popup-based tooltips. AI generated responses should be non-intrusive, with clearly distinguishable suggested code that can be accepted, ignored, or modified by the user.
- Full integration with Editor of code and AI code generation
    - Full editor integration is achievable by implementing support for IntelliJ's Language Server Protocol (LSP), which allows standard communication between the IDE and AI based code suggestion services. AI assistant can harness this protocol to deliver real-time completions, syntax-aware suggestions, and contextual feedback based on the e.g., XML or YAML configurations currently open in the editor. Leveraging LSP, it can access editor states, file structure, and cursor position to enhance the relevance of its suggestions.
- Handling unclear responses and missing context

- In cases where the AI cannot confidently generate a response, the system should:
  - Prompt the user for missing context or highlight ambiguous input.
  - Offer guided questions or show previous examples relevant to the current file type or schema.
  - Provide navigation aids or quick links to documentation or previous working examples within the project.
  - This layered fallback design improves the robustness and reliability of the assistant in professional environments.

Relevant plugins for IntelliJ IDEA:

Plugin name	Positive features	Negative features
JetBrains AI Assistant[59]	Official plugin by JetBrains offering chat-based AI, code completion, refactoring suggestions, and documentation generation	Built-in support for OpenAI, Anthropic, and others, though it uses hosted services under a JetBrains license, it's not self-hosted
Craftspire GPT Assist[60]	Open source, possible extends (customizable), chat inside IntelliJ, Simple UI	limitations within the of AI Models integration, there have been no changes for a long time
AI Coder[61]	Apache-2.0 open source, patch-based transformation and request logging, nice implementation for our potential extents, current plugin support	Less documentation, fewer community users (as of now)

Summary

For our needs, there is one open source plugin solution available that was developed to integrate AI assistant and LLM into IntelliJ IDEA called AI Coder. This plugin provides the ability to connect to an API providing an AI model. It is also possible to extend and customize the plugin, but due to the necessary such far customization for our needs, for example, panels that provide selection of supported object classes and operations for the generated connector, adding necessary menu actions associated with connector generation, etc., it is not enough. For this reason, it is necessary to implement the integration into our midPoint Studio plugin.

Enhancing midPoint Studio plugin with AI assistant, once fully integrated, would become a central assistive feature for integrating midPoint tool within the midPoint Studio plugin bridging AI-powered assistance with the professional IntelliJ IDEA functionality. The design proposal should focus on the experience of using artificial intelligence assistants in development environments that are currently freely available, and should also take into account the method of integration into the midPoint web application. These innovations contribute to a more intelligent and efficient midPoint configuration experience.

MidPoint Studio Wizard

IntelliJ IDEA by JetBrains offers a general Project Wizard that facilitates the creation of new projects through a series of guided steps. This wizard can be customized and extended via plugins to support various project types and configurations. For example, primary dialog windows and tooltips are used to navigate project creation, configuration or other processes. This solution for general wizard in IntelliJ IDEA can also be used in specific scenarios of lightweight wizard for midPoint configuration needs.

3.7 AI Development & Testing Tools

As part of the project, we have to provide tools that support MLOps by facilitating, and potentially automating, the day-to-day work of our ML engineers, DevOps engineers, and developers.

MLOps is a methodology, or set of practices, combining DevOps, Machine Learning, and Data Engineering (Kreuzberger D. et al., 2023). One way to think about MLOps is through its five "pillars" (MLOps at a Reasonable Scale [The Ultimate Guide], 2024):

- Data ingestion (and optionally a feature store)
- Pipeline and orchestration
- Model registry and experiment tracking
- Model deployment and serving
- Model monitoring

Each pillar can be implemented in various ways, from simple scripts to sophisticated enterprise solutions. Many tools can be mapped to these pillars (AI Infrastructure Landscape, 2023).

The main obstacle we faced was the broad scope of MLOps and the vast landscape of MLOps tools. To overcome this, we focused on identifying our requirements to narrow the scope. Discussions with our ML teams resulted in several key requirements (mentioned in the previous section), dramatically reducing the scope.

Another important constraint is our focus on open source tools. This offers the advantage of fewer tools to evaluate but the disadvantage of potentially fewer features.

3.7.1 Requirements

During discussions with our AI/ML team, we identified the following requirements.

High Level Requirements:

- **R1**: Provides observability and monitoring of AI/ML pipelines and models.
    - Includes a GUI that allows visualization of collected traces at individual AI/ML pipeline node granularity.
- **R2**: Provides a means to run tests/evaluations of existing AI/ML pipelines and models, as well as pipelines and models under development.
    - Includes a GUI that allows running tests/evaluations manually.
- **R3**: Provides a means to maintain existing AI/ML pipelines and models.
    - Deployment of new pipelines and models.
    - Rollbacks to previous versions.
    - Deployment automation.

Specific Requirements from AI/ML Teams:

- **R4**: Supports the creation, maintenance, and execution of experiments.
- **R5**: Integrates seamlessly with JupyterLab.
- **R6**: Provides ways to run experiments utilizing GPU resources.
- **R7**: Supports experiment tracking and evaluations.
- **R8**: Supports end-to-end testing and evaluation of the AI/ML pipeline.
- **R9**: Live production tracing and evaluations with simple statistics such as:
    - Number of requests
    - Latency
    - Number of failures
    - Number of LLM retries
    - Number of LLM hallucinations

These requirements indicate that we need to introduce some level of MLOps.

3.7.2 Key AI Studio Areas

By evaluating our requirements, we identified two areas with different (but overlapping) needs:

- **Production Area**: Requires mainly observability and continuous evaluation (calculation of various performance metrics) of running AI pipelines.
- **Development Area**: Requires tools for developing, testing, and experimenting on AI/ML pipelines and algorithms.

These areas are connected by the **deployment** process, which takes built artifacts from development and deploys them to the production environment.

We also identified another area, not very relevant in the current project stages but potentially useful in the future:

- **Data Area**: Covers the collection, transformation, and labeling of production data.

Automated data collection and transformation are not currently relevant because of the limited amount of available data. Our engineering team processed the data manually faster than we could provide relevant tooling. We also lack an automated data ingestion source for transformation (apart from our support channels, which contain little useful data).

Production Area

For the production side of the AI Studio, good observability and monitoring are key (Chandrachood A., 2023; AI Observability: Ensuring Trust and Transparency in Machine Learning Pipelines, 2025). We focused on observability tools aimed at LLM-based AI pipelines, as this is the most probable technology our ML/AI team will use.

We began by researching available open source LLM observability tools. We evaluated seven tools:

- Weights and Biases
- LangTrace
- LangWatch
- LangFuse
- Lunary
- Phoenix
- Opik

We disqualified Weights and Biases (commercial license for on-premise use), LangWatch (commercial license), and Lunary (enterprise license required for pre-built container images).

LangTrace, LangFuse, Phoenix, and Opik provide almost the same functionality:

- Tracing at the AI pipeline "node" level
- Evaluations of collected traces
- Prompt management
- Dataset management

We will choose one of these four tools after practical evaluation based on:

- Ease of use

- Deployment complexity
- Maintainability
- Supported integrations

Development Area

For our purposes, let’s define a few terms related to experimentation in the AI Studio context:

- **Experiment**: The execution of an evaluation pipeline with specified inputs and expected outputs, while collecting various execution traces.
- **Experiment Evaluation**: The process of calculating different metrics based on the expected experiment output and collected traces. Specific metrics are defined by the executed evaluation pipeline.
- **Pipeline**: A set of connected steps forming an execution flow. Pipeline definition depends on the specific toolset. We differentiate three types of pipelines:
    - **Evaluation (or testing) Pipeline**: Defines the various steps of the evaluation process, potentially including input preparation, calls to the system under test (e.g., a particular AI pipeline or LLM model), response evaluation, metric calculation, etc. The evaluation pipeline can be considered an "experiment" definition.
    - **AI Pipeline**: In our context, a pipeline utilizing AI and providing business value.
    - **Data Pipeline**: A pipeline that processes input data and provides processed data as output. Data loading and saving may also be part of the pipeline. Data pipelines are not currently in scope but may be introduced later.
- **Experiment Tracking**: The process of collecting results and potentially other metadata of run experiments. Tools providing experiment tracking often support visualizing individual experiment results and sometimes allow experiment comparison.

Note: Term "evaluation" is more common than "testing" in AI/ML contexts, but we use them interchangeably.

We can summarize our development requirements as follows:

- Main emphasis on experimenting and testing, including experiment orchestration and tracking.
- Support for JupyterLab for writing (defining) and running experiments.
- Ability to run evaluation/testing pipelines from a GUI, ideally with parameterized input.

Many MLOps tools relate to experimentation, testing, and pipeline orchestration (AI Infrastructure Landscape, 2023). We found several open source tools that meet our needs (some only partially):

- MLflow

- ClearML
- Kubeflow
- Apache Airflow
- ZenML
- Polyaxon

These are not "all-in-one" solutions and often need to be combined. Below are brief descriptions:

- **MLflow**: Provides tracing, evaluations, experiment tracking, a model registry, etc., but does not orchestrate evaluation pipeline execution or allow GUI-based experiment execution.
- **ClearML**: A platform providing workflow orchestration, experiment tracking, observability, dataset management, a model registry, and more.
- **Kubeflow**: A Kubernetes-native set of tools for orchestrating ML workflows, hyperparameter tuning, model serving, Jupyter Notebook integration, etc., primarily focused on traditional ML and requiring Kubernetes knowledge.
- **Apache Airflow**: A platform primarily for workflow orchestration. Insufficient on its own for our requirements.
- **ZenML**: A framework for workflow orchestration, experiment tracking, etc. The open-source version has limitations.
- **Polyaxon**: A platform for workflow orchestration, experiment tracking, a model registry, etc. Development seems to have slowed. It is fairly complex.

The observability tools mentioned earlier also provide some experimentation functionalities, particularly experiment tracking combined with dataset management. However, they cannot run experiments; they only collect traces and potentially calculate predefined metrics.

After further research, including tool deployment, we chose ClearML as the best fit for our needs. The main reasons are:

- Tight Git integration.
- Fulfills all our requirements.
- Powerful GUI with the ability to run experiments directly.
- Simple SDK that will be easy for our AI/ML engineers to learn.

[1] https://developers.google.com/custom-search/v1/overview

[2] https://learn.microsoft.com/en-us/bing/search-apis/

[3] https://duckduckgo.com/api

[4] https://docs.github.com/en/rest/search

[5] https://api.stackexchange.com/

[6] https://scrapy.org/

[7] https://www.crummy.com/software/BeautifulSoup/

[8] https://www.selenium.dev/, https://playwright.dev/, https://pptr.dev/
[9] https://pdfminersix.readthedocs.io/, https://tika.apache.org/, https://pypdf2.readthedocs.io/
[10] https://www.import.io/
[11] https://www.diffbot.com/
[12] https://commoncrawl.org/
[13] https://www.octoparse.com/, https://www.webharvy.com/
[14] https://www.kofax.com/products/rpa
[15] https://github.com/features/copilot
[16] https://aws.amazon.com/q/developer/
[17] https://www.tabnine.com/
[18] https://replit.com/
[19] https://github.com/features/copilot
[20] https://aws.amazon.com/q/developer/
[21] https://www.tabnine.com/
[22] https://replit.com/
[23] https://github.com/biggorilla-gh/flexmatcher?tab=readme-ov-file
[24] https://dbs.uni-leipzig.de/research/projects/coma
[25] https://docs.adeptia.com/articles/!ac-professional/use-a-mapping-function
[26] https://github.com/YuxingLu613/KARMA
[27] https://flatfile.com/product/mapping/
[28] https://github.com/fireindark707/Python-Schema-Matching
[29] https://github.com/flyingwaters/prompt-matcher-for-schema-matching
[30] https://www.nngroup.com/articles/wizards/
[31] https://www.nngroup.com/articles/ten-usability-heuristics/
[32] https://www.nngroup.com/articles/mini-ia-structuring-information/
[33] https://www.nngroup.com/articles/wizards/
[34] https://www.smashingmagazine.com/2016/09/reducing-cognitive-overload-for-a-better-user-experience/
[35] https://www.nngroup.com/articles/minimize-cognitive-load/
[36] https://uxpsychology.substack.com/p/lost-in-navigation-overcoming-the
[37] https://www.nngroup.com/videos/ux-visualization-techniques/
[38] https://www.nngroup.com/articles/cards-component/
[39] https://arxiv.org/html/2402.07632v3
[40] https://arxiv.org/abs/1811.02164
[41] https://www.nngroup.com/articles/user-control-and-freedom/
[42] https://www.mdpi.com/2414-4088/6/1/2
[43] https://www.nngroup.com/articles/content-dispersion-methodology/
[44] https://www.uxpin.com/studio/blog/symmetry-vs-asymmetry-in-design/
[45] https://www.nngroup.com/articles/flexibility-efficiency-heuristic/
[46] https://www.nngroup.com/articles/ux-copy-sizes
[47] https://www.nngroup.com/articles/web-form-design/
[48] https://www.nngroup.com/articles/visual-hierarchy-ux-definition
[49] https://www.interaction-design.org/literature/topics/visual-hierarchy
[50] https://www.nngroup.com/articles/consistency-and-standards/
[51] https://uxmag.medium.com/consistency-in-ui-ux-design-the-key-to-user-satisfaction-60ab598ec42c
[52] https://baymard.com/blog/back-button-expectations
[53] https://www.smashingmagazine.com/2022/08/back-button-ux-design/

[54] https://www.nngroup.com/articles/reset-and-cancel-buttons/

[55] https://m3.material.io/foundations/layout/understanding-layout

[56] https://developer.apple.com/design/human-interface-guidelines/split-views

[57] https://uxplanet.org/the-power-of-whitespace-a1a95e45f82b

[58] https://www.nngroup.com/articles/tabs-used-right/

[59] https://www.jetbrains.com/ai-assistant/

[60] https://github.com/MateuszZozulinski/CraftspireGPTAssist

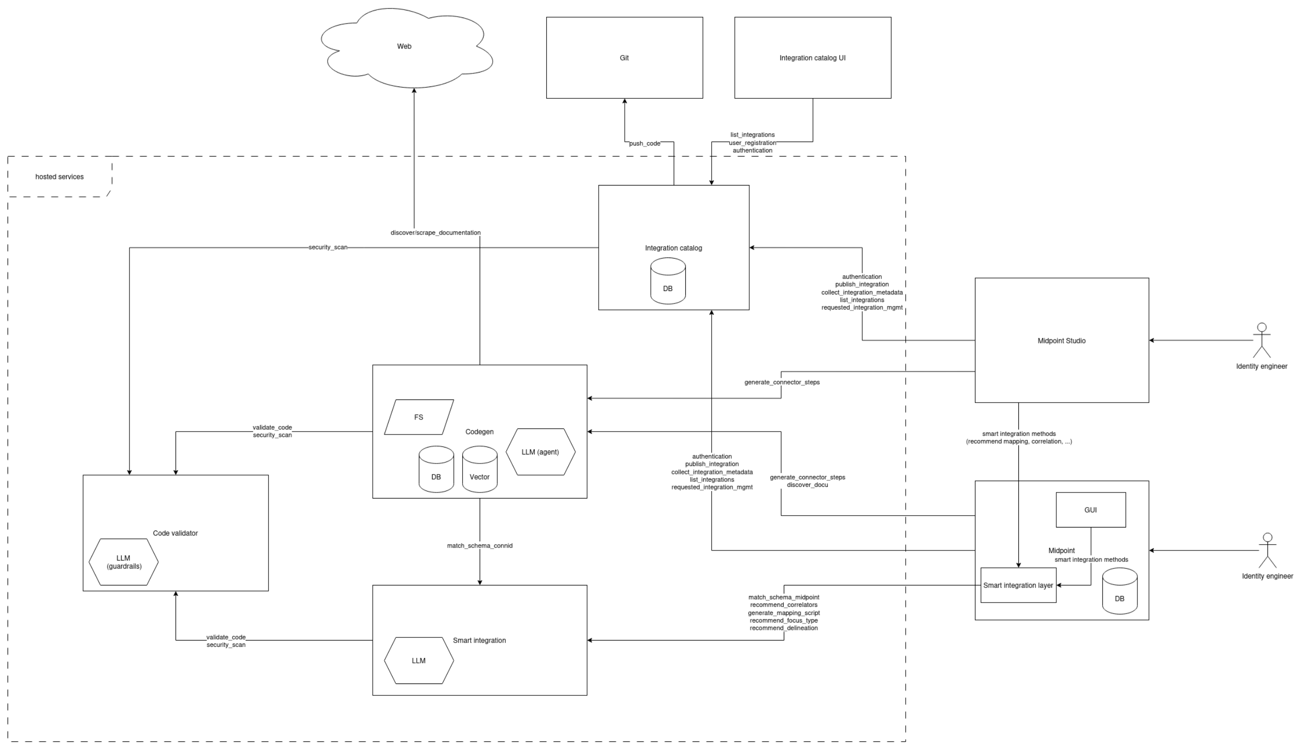
[61] https://github.com/SimiaCryptus/intellij-aicoder

4. Architecture and Design

This chapter presents the proposed architecture of the solution, in the form of component diagram, detailed descriptions of individual components, and their interfaces. Afterwards, it describes the current state of the design of individual components and their functions. This includes identified necessary changes in midPoint, ConnId Framework, and midPoint Studio.

4.1 Component-Level Diagram

The diagram below outlines the high-level architecture and its main components, omitting details on the infrastructure and supporting tools.



4.2 Component Descriptions

MidPoint

- Description: MidPoint is an identity management and governance platform that will use the Smart Integration Layer and Codegen service to simplify the process of integrating a new application.
- Type: Web application
- Internal architecture: MidPoint is composed of several subsystems, each subsystem contains several components.
  - User interface subsystem implements web-based administration and configuration interface.

  - Repository subsystem is storing authoritative identity data, pointers to identity objects in other systems (such as accounts), definition of roles, expressions, synchronization policies, configuration and almost all the persistent data structures in the system. The data are stored in a relational database.
  - Provisioning subsystem can talk to other systems, can read data and modify the data. It is doing so by using the connectors.
  - IDM model Subsystem is gluing everything together. It is a component through which all activities pass, therefore it is an ideal place to enforce policies, fill-in missing data, map attributes and properties, govern processes and do all the identity management logic. Smart integration layer is part of the IDM model Subsystem.
  - Infrastructure system provides common data structures, code and specifications that are used by all other components. This is the place where midPoint data model (schema) is defined.
- Responsibilities related with project:
  - Offers panels for managing applications, including the full lifecycle of created applications.
  - Allows users to add new applications using a list retrieved from the Integration catalog.
  - Supports the creation of connectors, during which the GUI communicates with the Codegen service. Includes the ability to manage incomplete connectors.
  - Enables users to upload connectors and configurations to the Integration catalog.
  - Applies configuration suggestions provided by the Smart integration layer. These suggestions include delineation, mappings, correlation rules, and association configurations, which can be incorporated into the current configuration.
  - Can submit requests for creating application connector and also display a list of requested application connectors retrieved from the Integration catalog.
  - Provides validation of the connector to determine whether it can be used in midPoint based on a signed list of application connectors obtained from the Integration catalog.
  - Offers API endpoints that use the Smart Integration Layer and expose its functionality to midPoint studio.
- Dependencies
  - Dependence on the Integration catalog and Smart integration layer for data provision.
- Data requirements:
  - Requests Integration catalog for existing application connectors in order to be displayed in GUI.
  - Use the signed list of supported connectors retrieved from the Integration catalog.
  - Requires configuration suggestions from the Smart Integration Layer.

MidPoint Studio

- Description: MidPoint Studio is an integrated development environment (IDE) to develop solutions based on midPoint. The primary purpose of MidPoint Studio is to support process of creating, developing and maintaining midPoint configurations.
- Type: plug-in for IntelliJ
- Responsibilities related with project:
    - Supports working with application objects directly via XML files.
    - Allows users to modify connectors retrieved from the Integration catalog directly using scripts.
    - Supports the creation of connectors with the help of the Codegen service, including the ability to manage incomplete connectors.
    - Enables users to upload connectors and configurations to the Integration catalog.
    - Applies configuration suggestions provided by the Smart Integration Layer. These suggestions include delineation, mappings, correlation rules, and association configurations, which can be incorporated into the XML file of the current configuration.
- Dependencies
    - Both, midPoint and midPoint Studio depends on the Integration catalog and Smart integration layer for data provision.

Codegen

- Description: Codegen is a service designed to assist with the generation of fully functional connector code for integrating applications with midPoint. It streamlines the process of creating connectors by generating declarative configuration and scripts using the base connector framework.
- Type: Service / Microservice
- Responsibilities:
    - Provides an API for accessing algorithms related to code generation, including documentation handling and the actual code generation.
    - Generates declarative configuration and scripts according to the needs of base connector framework.
    - Integrates with the Smart integration service to apply "mappings" between ConnId schema and the native schema of the application.
    - Ensures the quality, security of generated scripts by integrating Code validator service
    - Discovers and extracts documentation from web sources to allow non-technical users to integrate applications with midPoint.
- Data requirements:

  - Requires information about application to integrate, e.g. name of the application, (URL to) technical documentation, (URL to) the API specification

Smart Integration Layer

- Description: Entry point to the smart delineation, mapping and correlation features in midPoint and midpoint Studio
- Type: Java package
- Responsibilities:
  - Provides a clean interface to the smart features used by midPoint and midPoint Studio
  - Implements the business logic needed for integration with the UI
  - Prepares data for the ML/AI algorithms and for the communication with Smart integration service
  - Implements algorithms and heuristics on top of core algorithms of the Smart integration service
  - Utilizes existing data from the Integration catalog to improve recommendations
- Data requirements:
  - Queries and processes data directly from midPoint database (accounts, groups, users, etc.) in order to be passed to the underlying algorithms
  - Requests Integration catalog for existing data in order to be analyzed for improving recommendations

Smart Integration

- Description: Service providing "core" integration AI algorithms for the delineation, mapping and correlation functionality in midPoint
- Type: Stateless microservice
- Responsibilities:
  - Provides an API for accessing "core" algorithms and their variations depending on the use case
  - Core algorithms are schema matching, correlator recommendation, mapping script generation, and delineation recommendation
  - Ensures the quality and security of generated scripts by integrating Code validator service
  - Provides a configuration mechanism that will allow to plug in custom LLM models.
- Data requirements:
  - Requires complete data schemas as input for the schema matching and correlator recommendation algorithms

    - Requires a few correlated data samples (accounts, groups, users, etc.) as input for the script generation algorithm
    - Requires a few data samples as input for focus type suggestion (optional)
    - Requires structured data analysis results on the resource's dataset (like value counts, prefix analysis, etc.) as input for the delineation algorithm

Integration Catalog

- Description: Integration catalog is a centralized service designed to manage, present, and facilitate interaction with application connectors. Its core purpose is to provide a clear overview of all available application connectors and streamline their usage.
- Type: Stateless microservice (REST API), Stateful web application (GUI)
- Responsibilities:
    - Provides both API and GUI to:
        - View available application connectors with metadata.
        - Filter and search connectors by criteria.
        - Upload new integrations, either connectors or configuration, with relevant metadata.
        - Download individual connectors for use in midPoint.
        - Access a digitally signed list of connectors as a verified, tamper-proof snapshot.
        - Request missing application connectors and view the list of requested ones.
        - Provided integration configurations to the Smart Integration Service to improve AI assistant suggestions.
    - Ensures security and accountability through user registration and authentication, allowing only authorized users to upload connectors.
- Data requirements:
- Requires connectors and configuration that is uploaded by midPoint, midPoint Studio or directly by GUI of the Integration catalog.
- Dependencies:
    - Integration catalog will utilize the Code Validator to ensure the quality, correctness, and security of uploaded content.

Code Validator

- Description: Service providing code validation and security scanning for the generated/created code
- Type: Stateless microservice
- Responsibilities:

    - Validates AI-generated code related to connectors and mapping scripts (checks for compilation errors, etc.)
    - Scans code generated by AI or being published to the Integration catalog for vulnerabilities and misuse
- Data requirements:
    - Requires code and configuration that is being produced by Codegen and Smart integration services or published to the Integration catalog

4.3 API Specifications

4.3.1 Codegen Interface

Method: Discover a Technical Documentation for the Application

Parameters:

- Application name

Returns:

- list of found documentation URLs for the application

Method: Get a Type of Connector for the Application

Parameters:

- Application name

Returns one of:

- type of connector for the application
- **error:** connector type can not be found in the documentation
- **error:** unknown connector type
- **error:** missing documentation for the application

Method: Get a Base URL for the Application

Parameters:

- Application name

Returns one of:

- base URL for the application

- **error:** base URL can not be found in the documentation
- **error:** missing documentation for the application

Method: Get an Authentication Method(s) for the Application

Parameters:

- Application name

Returns one of:

- (list of) authentication method(s) for the application
- **error:** authentication method can not be found in the documentation
- **error:** missing documentation for the application

Method: Get a List of Object Classes for the Application

Parameters:

- Application name

Returns one of:

- list of Object Classes for the application
- **error:** missing documentation for the application

Method: Get an Object Class schema (list of attributes and types) for the application

Parameters:

- Application name
- Object Class name

Returns one of:

- schema (list of attributes and types) for the application
- **error:** missing documentation for the application

Method: Generate a Native Schema Script for the Object Class of the Application

Parameters:

- Application name
- Object Class name

Returns one of:

- native schema script for the Object Class of the application
- **error:** missing documentation for the application

Method: Generate a search all script for the Object Class of the application

Parameters:

- Application name
- Object Class name

Returns one of:

- search script for the Object Class of the application
- **error:** missing documentation for the application

Method: Upload Technical Documentation for the Application

Parameters:

- Application name
- URL(s) and/or file(s) with technical documentation for the application

Method: Upload list of (confirmed) Object Classes for the application

Parameters:

- Application name
- list of Object Classes for the application

Method: Upload (confirmed) schema (list of confirmed attributes and types) for the application

Parameters:

- Application name
- list of Object Classes for the application

Method: Upload (confirmed) native schema script for the ObjectClass of the application

Parameters:

- Application name
- native schema script for the Object Class of the application

Method: Upload (confirmed) search all script for the ObjectClass of the application

Parameters:

- Application name
- search all script for the Object Class of the application

4.3.2 Smart Integration Layer Interface

Smart integration layer provides midPoint and midPoint Studio with intelligent mapping and correlation capabilities. It exposes Java interface that will be used by midPoint GUI as well as REST adapter for the same functionality used by midPoint Studio.

Method: Suggest Delineation

Suggests delineations for the application (resource) object class. Each suggestion consists of the type identifier (kind/intent) and the delineation (base context, attribute filter, condition).

Parameters:

- resource oid (required)
- object class (required)

Returns:

- delineation suggestions: kind/intent, delineation (base context, attribute filter, condition)

Method: Suggest Focus Type

Suggests a discrete focus type for the application (resource) object type.

Parameters:

- resource oid (required)
- kind/intent (required)

Returns:

- focus type

Method: Suggest Correlation

Suggests correlation mappings and correlation rules following existing XSD schema. Provides statistics on the expected correlation rule quality. Interaction metadata contains information on user interaction (e.g. rejected suggestions that won't be recommended again).

Parameters:

- resource oid (required)

- kind/intent (required)
- focus type (required)
- interaction metadata

Returns:

- correlation rule and mapping suggestions (XML config)
- correlation statistics

Method: Suggest Attribute Mapping

Suggests mappings for the application attributes: to existing midPoint attributes (standard, extension), plus suggests creation of new extension attributes. Provides statistics on the expected mapping quality when suggesting mapping to existing attributes. Mappings are suggested for the whole object class schema or for selected attributes. Interaction metadata contains information on user interaction (e.g. rejected suggestions that won't be recommended again).

Parameters:

- resource oid (required)
- kind/intent (required)
- focus type (required)
- filter
  - midpoint attributes
  - application (resource) attributes
  - "inbound/outbound" selector
- interaction metadata

Returns:

- mapping suggestions related to the standard midPoint schema and existing extension attributes (XML config)
- new extension attribute mapping suggestions (XML config)
- mapping statistics

NOTE The application object class schema can be quite large, having hundreds of attributes. We assume that the recommender will not recommend mappings for all of them. The reason for this assumption is that most probably we won't have hundreds of extension attributes in midPoint, so the recommender will not have enough counterparts for all these application attributes. (Regarding suggesting

> new extension attributes, we assume at most a few of them being suggested, e.g., only for candidate correlation attributes.)

Method: Suggest Associations

Suggests all associations for the application (resource) or for a specific object type following existing XSD schema. Provides statistics on the expected mapping quality. Interaction metadata contains information on user interaction (e.g. rejected suggestions that won't be recommended again).

Parameters:

- resource oid (required)
- filter
  - set of subject types (kind/intent)
  - set of object types (kind/intent)
- interaction metadata

Returns:

- association suggestions (XML config)
- mapping statistics

4.3.3 Smart Integration Service Interface

Smart integration service provides "core" integration AI algorithms for the delineation, mapping and correlation functionality. It exposes RESTful API and it is primarily used by "Smart integration layer" that further combines these algorithms with business logic and custom heuristics.

Method: Suggest Delineation

Suggests delineations for the application (resource) object class. Each suggestion consists of the type identifier (kind/intent) and the delineation (base context, attribute filter, condition).

Parameters:

- object class (required)
- application schema for given object class (required)
- structured analysis and statistics on the application's data (required)

Returns:

- delineation suggestions: kind/intent, delineation (base context, attribute filter, condition)

Method: Suggest Focus Type

Suggests a discrete focus type for the application object type.

Parameters:

- kind/intent (required)
- application schema for object class (required)
- N data samples

Returns:

- focus type

Method: Match MidPoint Schema

Suggests which application's attribute to map to which midPoint's attributes. Application schema contains attribute names and types, and optionally attribute descriptions.

Parameters:

- application schema for given object class (required)
- midPoint schema (required)

Returns:

- list of application attribute suggestions for each midPoint attribute

Method: Match ConnId Schema

Suggests which application's attribute to map to which ConnId's attributes. Application schema contains attribute names and types and optionally attribute descriptions.

Parameters:

- application schema for given object class (required)
- ConnId schema (required)

Returns:

- list of application attribute suggestions for each ConnId attribute

Method: Suggest Correlation Attributes

Suggests attributes suitable for correlation (focused on extension attributes).

Parameters:

- list of attributes (required)

Returns:

- list of correlation attributes

Method: Suggest Mapping Script

Generates and suggests a mapping transformation script or a complex attribute from a few data samples.

Parameters:

- N samples of the input data and the expected output (required)

Returns:

- Mapping script or a complex attribute

4.3.4 Integration Catalog Endpoints

Method: Search Connectors

List all possible application connectors in the Integration catalog available to use in midPoint and midPoint Studio.

Authentication:

- without authentication

Parameters:

- filter
- paging

Returns:

- list of application connectors with attributes (probably JSON)

Method: Get Connector Versions

Showing possible versions with version’s attributes of one application connector in midPoint and midPoint Studio.

Authentication:

- without authentication

Parameters:

- identifier of application connector

Returns:

- list of application connector versions with attributes (probably JSON)

Method: Download Connector

Download the connector to integrate a new application with the midPoint, or continue developing the connector in the midPoint or midPoint Studio, if it is possible.

Authentication:

- without authentication

Parameters:

- identifier of application connector

Returns:

- connector file

Method: Request a Connector

Request a connector to be developed and added into the Integration catalog.

Authentication:

- without authentication

Parameters:

- application name
- application version (relevant for on-prem apps)
- Think about other parameters, such as description

Returns:

- success or an error with details

Method: Request Extension of the Connector

Request an extension of the connector capabilities or support of some object class. Connector is in Integration catalog, but some needed capability or object class missing.

Authentication:

- without authentication

Parameters:

- identifier of application connector
- missing capability
- description of missing object class

Returns:

- success or an error with details

Method: Search Requested Connectors and Extensions of the Connectors

If someone from the community would like to contribute, we will show the requested connectors and extensions of the connectors in midPoint and Studio.

Authentication:

- without authentication

Parameters:

- filter
- paging

Returns:

- list of requested connectors and extensions of the connectors with attributes (probably JSON)

Method: Upload of New Connector

Upload the new connector that was created in either midPoint or Studio.

Authentication:

- only authenticated users

Parameters:

- in-progress connector structure

Returns:

- new connector

Method: Upload of Resource Configuration

Upload the resource configuration—primarily focusing on the Schema Handling section (e.g., mappings, correlation rules)—associated with the application used in midPoint or Studio.

Authentication:

- only authenticated users

Parameters:

- resource configuration
- identifier of application connector

Returns:

- success or an error with details

Method: Search Resource Configurations

Using by Smart integration service to reflect the resource configurations in suggestions. The Smart Integration service needs to retrieve the stored configurations for the corresponding Resource Object Type. Since this type is defined based on the object class, we need to obtain the object class as a parameter. This parameter will be used to filter the available configurations for the given application, ensuring that only the relevant ones are selected and processed.

Authentication:

- behind authentication, allowed only for Smart Integration service

Parameters:

- identifier of application connector
- object class

Returns:

- list of resource configurations for the given application connector and object class

Method: Get Signed Sheet of Application Connectors

Used by midPoint to check whether the use of the connector is permitted.

Authentication:

- without authentication

Parameters:

- timestamp (optional)

Returns:

- signed sheet of application connectors added after timestamp

4.3.5 Code Validator Interface

Method: Validate Code

Validates code snippet for errors (e.g. compilation errors) in a given context. The type parameter specifies the context in which to run validation for example: mapping script, codegen specific methods.

Parameters:

- code snippet (required)
- type (required)

Returns:

- success or an error with details

Method: Scan Code

Scans code snippet for potential security issues in a given context. The type parameter specifies the context in which to run for example: mapping script, codegen.

Parameters:

- code snippet (required)
- type (required)

Returns:

- success or an error with details

4.4 Connector Code Generator

The connector code generator is a proposed component intended to streamline development of connectors for midPoint, specifically targeting applications that lack existing integration. The design focuses on simplifying the connector creation process by leveraging AI-assisted tooling and reusable architectural patterns, particularly for users with limited programming expertise or limited knowledge of the target system's API.

Connector development is typically a complex, multi-step process (outlined in Section 3.1), involving documentation analysis, data model mapping, endpoint handling, and relationship logic. The generator component is designed to abstract and modularize this process, allowing users to progress through guided steps while automated components handle lower-level tasks.

The component is structured into the following subsystems:

1. **Search and Discovery (Optional)**: This module will initiate queries based on application name or keywords to identify likely sources of technical documentation. It will return a list of candidate URLs for user review. Once the user has made a selection, the chosen URLs will be passed to the Scraper module as input. If no suitable documentation is found automatically, the user may also provide URLs manually.
2. **Scraper**: Starting from user-provided or discovered URLs, this module will navigate through documentation portals to extract relevant content. It will handle semi-structured HTML, Markdown, OpenAPI specs, and other common documentation formats. The goal is to identify API reference material containing endpoints, object types, and usage examples.
3. **Digester**: The task of the digester will be processing of the scraped or user-supplied documentation to extract structured API metadata. This includes:
   - Identification of object types (e.g., users, groups, memberships)
   - Schema definitions (attribute names, types, cardinality)
   - Endpoint definitions (HTTP methods, paths, authentication requirements)
   - Relationship semantics (e.g., ownership between objects)

   The digester serves as a bridge between unstructured input and formal models used by the generator.
4. **Code Generator**: It will consume the digested API metadata and generate connector code targeting the ConnId connector framework and its integration with midPoint. Generated code will leverage a pre-defined base connector framework for consistency (described in Section 4.4.1) The generated output is intended to be executable and testable out-of-the-box, with optional hooks for interactive user confirmation or correction (e.g., during object schema alignment).

Connector code generation will be performed in small, well-defined steps to avoid producing large, unmanageable code blocks at once. The goal is to generate specific operations for specific object types incrementally. For example, the process may begin by generating a schema operation for the user object type, followed by a search operation for the same type. This incremental approach simplifies validation and helps maintain a strong focus on security. Each step will be independently testable, enabling partial verification and correction before progressing to the next stage. Users will be actively involved throughout the process — at each step, they will have the opportunity to review, confirm, modify, or reject generated outputs. This user-in-the-loop approach enhances the overall robustness, transparency, and security of the generated code.

4.4.1 Base Connector Framework

The base connector framework is a support component for connector code generator, designed to provide a flexible and extensible base, implementing common reusable functionality for supported protocols and technologies, reducing the amount of generated code and boilerplate code in connectors.

The base connector framework will be designed to meet the requirements identified in analysis.

The framework will consist of the following components:

- **Base Connector Framework Support**:
  - Set of base classes necessary to integrate with ConnId framework.
- **Schema Mapping & Schema Builder APIs**:
  - Provides a way to define and manage schema mappings protocols and ConnId schema.
  - Allows for easy customization of attribute names and types regardless of protocol used.
  - Allows for specifying object classes, attributes, relationships in declarative way.
- **Authentication & Authorization**:
  - Common configuration model for authentication and authorization mechanisms, such as OAuth 2.0, API keys, and JWTs.
  - Providing initialized REST client with authentication and authorization headers, tokens, and other required parameters to REST and SCIM Support classes.
- **ConnId Support & Operation Builders**:
  - Support for ConnId operations using strategy design pattern, where based on ConnId request, the appropriate strategies are selected to handle the request.
    - Strategies are configured & provided by protocol specific parts and custom scripts.
- **REST Support**:
  - Declarative support for calling, retrieving, creating, updating and deleting data using REST APIs.
  - Endpoint-based strategies for handling ConnId search, update, create, and delete operations.
  - Support for custom strategies for handling non-standard cases such as complex updates, splitting operations into multiple sub-operations, indirect searches.
- **SCIM Support**:
  - Supports automatic schema discovery using SCIM 2.0 protocol.
  - Contributes to schema mappings and provides strategies based on discovered SCIM 2.0 schema, so there is no need to manually define schema mappings for SCIM 2.0 compliant applications.

  - Built-in strategies for handling ConnId search, update, create, and delete operations based on SCIM 2.0 protocol.
    - Multiple strategies for update operations, delete operations.
    - Support for custom strategies to be defined for handling non-standard cases such as complex updates, splitting operations into multiple sub-operations, indirect searches.
- **Scripting Support**:
  - Custom Groovy DSL for configuring & defining custom schema mappings, logic, strategies, and operations.
  - Scripting support is intended to implement custom connectors using Groovy with intent to minimize boilerplate code.
- **SQL Support**:
  - Supports automatic schema discovery using SQL database metadata.
  - Contributes to schema mappings and provide strategies based on discovered SQL schema, so there is no need to manually define schema mappings for database tables.
  - Support for customization of naming, attribute mappings and table relationships.
  - Built-in strategies for handling ConnId search, update, create, and delete operations.
  - Support for custom strategies for handling non-standard cases such as complex updates, splitting operations into multiple sub-operations, indirect searches.

Connector Packaging

The generated connector configuration and code will be packaged as a ConnId connector, following the ConnId packaging guidelines. The connector is packaged as a ZIP file, which can be easily deployed to the ConnId framework.

The package will contain the following:

- **Connector Manifest** - Required by ConnId to correctly identify package as a connector, provides metadata about base connector entry points.
- **Connector Framework JAR** - the base connector framework JAR file, which contains the base classes, support for ConnId framework, support of loading declarative configuration of connector frameworks.
- **Configuration Loader** - a manifest / class responsible for listing all declarative configurations available in the connector package.
- **Declarative Configuration** - the generated connector configuration, which contains the connector implementation, schema mappings, and custom scripts.

The packaged generated connectors will be uploaded to Integration catalog in order to allow users to easily discover and use them, minimizing the need to regenerate the connector code multiple times for the same application.

4.5 Mapping Recommendation

The goal of the Model-Mapping Recommendation System is to help users by suggesting how to map attributes from connected application to their counterparts in midPoint. The system should be able to generate various types of recommendations, starting with the simplest one and extending to the more complex ones. An example of the simple mapping might involve direct linking from an attribute in the connected application to a midPoint attribute, without a need of any other transformation of the value. More complex cases may require transformation, such as scripts to convert or combine values before they're mapped. As a result, the recommender should be able to recommend single, most likely mapping while also offering alternative suggestions and allowing users to select the most suitable one. The system could leverage lightweight heuristics, statistical and transformer-based models, and large language models (LLMs) to generate its recommendations to maximize precision and minimize manual correction effort.

In addition to these core matching and mapping algorithms, the final solution will also include a simple “Suggest focus type” method, which recommends the appropriate midPoint focus type for a given application object type. Because there is only a small and fixed set of midPoint focus types (user, role, org, service), this is quite a simple classification task: to guess e.g. whether given application object type is rather a user account or a group belonging to a role. We assume (without detailed technical analysis) that a LLM will be able to do this.

The Model-Mapping Recommendation System will also suggest creating new extension attributes in midPoint. For example, potential correlation attributes can be suggested here.

The four-step cascade described in section 3.2.2, i.e., direct identity mapping, static lookup tables, heuristic transformations, and LLM-assisted transformation code generation—provides a robust theoretical foundation for schema mapping. In our prototype, we will first implement the direct, lookup, and LLM-assisted phases, then conduct experiments and quantitative analysis to pinpoint the value patterns and attributes where the LLM underperforms. Based on these findings, we'll introduce targeted heuristic transformations (e.g., normalization, fuzzy matching, and string/date functions) ahead of the LLM step to reliably cover predictable cases. For scenarios involving sensitive or personal outbound data, where sending values to an external LLM may not be permitted, we'll fall back exclusively on basic deterministic algorithms and heuristics. This hybrid, data-driven workflow leverages transparent, rule-based mappings for routine variations while reserving the LLM to generate custom transformation code for truly complex, domain-specific logic, delivering a scalable, accurate, and privacy-aware mapping solution.

4.6 Correlation Recommendation

In this project, the Correlation Recommendation System aims to provide the midPoint configuration necessary to carry out the correlation. In cooperation with the Model-Mapping Recommendation System it does the following:

1. Suggests *correlation attributes*, i.e., attributes in the application and in midPoint that are relevant for the correlation.
2. Suggests *correlation rules*.

Processes of schema mapping and correlation often go hand in hand: the determination of non-trivial attribute mappings may depend on the presence of (at least partially) correlated data. Hence, the overall process of connecting an application to midPoint regarding schema matching, mapping, and correlation will look like this:

1. The first correlation attribute is determined and the correlation is executed. It is assumed that this attribute in application directly corresponds to an attribute in midPoint, i.e., there is an "as-is" mapping between them.
2. Mappings for other attributes are created, based on this preliminary correlation.
3. If needed, other correlation attributes are determined, now utilizing newly discovered mappings. They are used in the correlation. And this process can iterate by repeating from point 2 above.

4.6.1 Suggesting Correlation Attributes

Our goal here is to suggest which midPoint and/or application attributes can be used to correlate objects (e.g., user accounts or groups). We will select the most suitable approaches among those mentioned in Section 3:

- Predefined correlator list
- Heuristic-based selection (name patterns, data types check)
- LLM-assisted suggestion

4.6.2 Suggesting Correlation Rules

Expected approach uses most suitable approaches among those mentioned in Section 3, like this:

1. We'll attempt the exact matching first.
2. If that fails (e.g., because of case mismatch, diacriticals, or minor formatting differences), we'll fall back to a set of relaxed matching strategies:
   - Normalization-based matching
   - String similarity matching (e.g., Levenshtein distance, Jaccard similarity)
   - Phonetic matching (e.g., Soundex, Metaphone)

Note that midPoint currently does not support some of the above methods (e.g., phonetic matching or Jaccard similarity). If it's needed, we can consider adding this functionality.

4.7 Delineation

Experiments in this area, described in Section 3, suggest that we need to do the following:

1. Consulting domain experts to capture their implicit knowledge how to create and verify delineations.
2. Based on this knowledge, refine our analytical steps and prompt designs.
3. Verify our solution using improved evaluation pipelines, taking into account the issues we identified (multiple seemingly equivalent options when creating delineations, avoiding the use of accidental data characteristics, avoiding the use of mutable fields, and so on).

4.8 User Interface (MidPoint Studio Plugin & Web UI)

User interface design must place a strong emphasis on both User Experience (UX) and User Interface (UI) principles. Our goal is to create a product that is not only visually consistent and responsive, but also intuitive, accessible, and user-centric. By combining clean and purposeful design with a deep understanding of user needs, we ensure that the interface is easy to navigate, efficient to use, and provides a seamless overall experience.

In the project, there are three distinct areas where the user interface varies based on the environment and the type of user interacting with it. Each interface is tailored to meet the specific needs, context, and workflows of its intended user group, ensuring a more efficient and user-friendly experience across different usage scenarios.

To begin with the explanation, we need to define few key terms. The first is **Application Connector**, which represents a potential integration with a selected application. The second is **Application**, which is a new object in midPoint, representing an integrated/used application in the environment with configured business attributes such as application owner, level of approval, etc. Next is **Resource**, which is the technical representation for application integration and uses the connector to communicate with the application's service. The last term is **Application Catalog**, which is a list of all applications in the company; from a technical perspective, it is a list of applications along with the business attributes related to application.

MidPoint

In midPoint, we need to focus on users who are not highly technically skilled and who do not feel comfortable working directly with object representations such as XML, JSON, or YAML. Instead, they prefer a more visual and intuitive configuration experience.

MidPoint will include a sequence of guided steps designed to lead users through entire processes in a clear and structured way. Emphasis is placed on keeping users fully informed about where they are within the flow and what their specific task is at each stage. The main flows will include:

- creating a connector for integrating an application, with support for defining all necessary capabilities;
- modifying an existing connector, whether due to changes on the side of the application (e.g. API changes) or to add additional capabilities;
- and creating or updating a resource configuration for an application, which will also include a flow for using AI-assisted recommendations to suggest relevant parts of the configuration.

MidPoint Studio

In midPoint Studio, we target a more experienced and technically skilled user who is comfortable working directly with object representations in XML, JSON, or YAML. This environment is designed for advanced users who prefer full control over configurations and are familiar with the underlying data structures and syntax. Moreover, midPoint Studio will also integrate with the connector code generator functionality, allowing users to prepare new connectors for application integrations directly within the IDE.

In midPoint Studio, it is important to follow the UX/UI conventions of the IntelliJ IDEA application. While users will primarily work with object representations in XML, JSON, or YAML, there will be also support for implementing new connectors using the connector code generator component. For the connector code generator, there will be a need for a lightweight wizard to help with necessary steps to successfully create a new connector. For panels that interact with AI, we aim to apply the same layout standards used by existing copilots, ensuring a familiar and seamless experience for users who are already accustomed to IntelliJ IDEA.

Integration Catalog

The Integration catalog will contain a list of connectors representing possible application integrations and provide a set of available operations that can be performed on them. It will serve as a central point for managing applications integrations, allowing users to easily browse, upload, or download existing connectors.

These functions will also be supported directly within midPoint, which will communicate with the Integration catalog. However, it is necessary to support them directly in the UI of the Integration catalog as well, to accommodate midPoint deployments that do not have direct access to the Integration catalog.

The Integration catalog will also allow users to request applications integrations that are not yet available and will display a list of application integrations that have been requested. This feature helps track demand and encourages contributions of new connectors for requested applications integrations from community members.

4.8.1 Description of Guided Wizards and Integration Catalog UI (User Flow Diagrams)

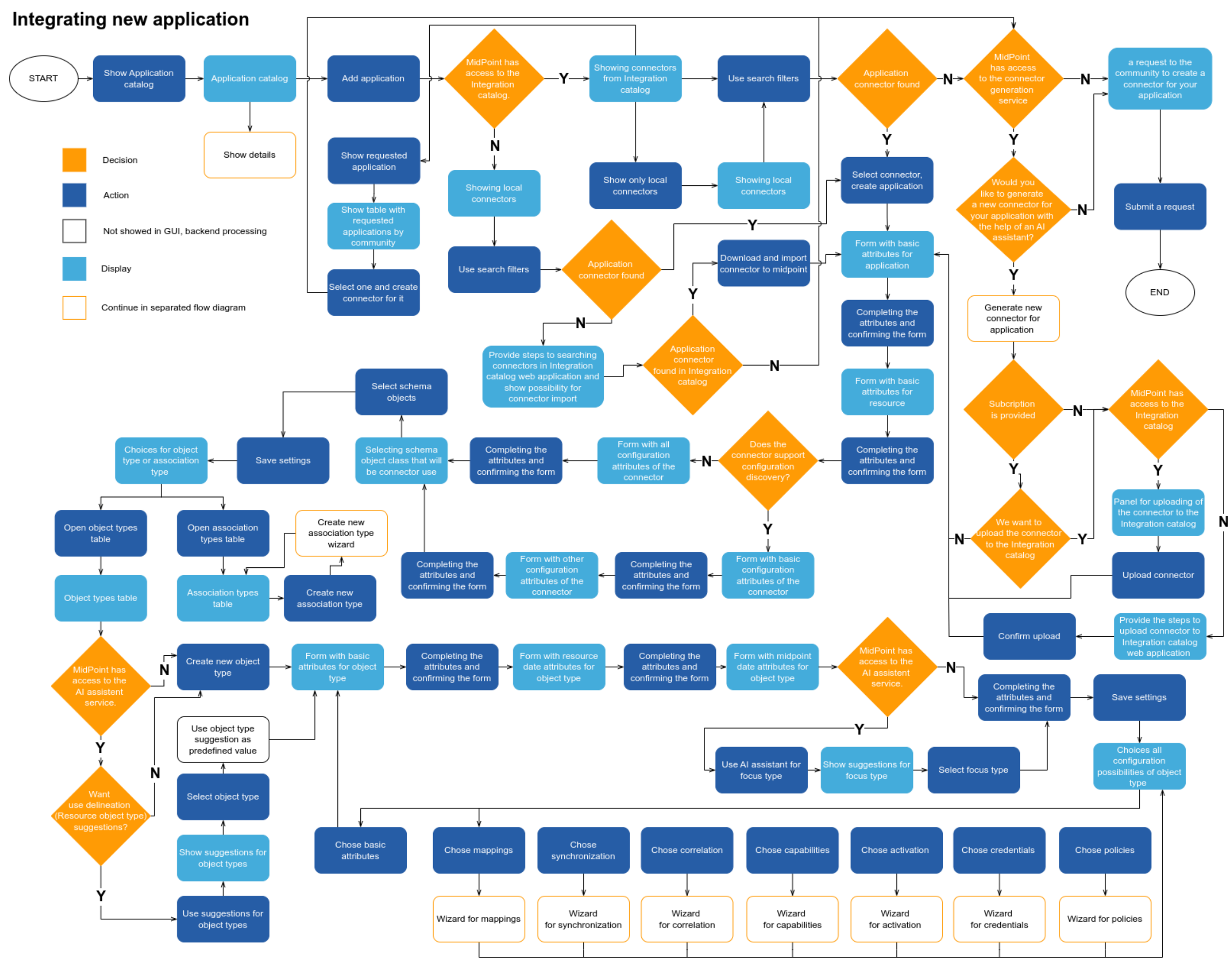
The initial user flow diagrams presented in this section are intended to illustrate the expected flow of user interaction and to capture the functionality provided by the system.

These diagrams serve as a conceptual model of the anticipated behavior and user journey within the interface. However, they are subject to change as a result of the detailed design process of individual UI panels and adjustments that may arise during later stages of implementation.

They should be considered as preliminary drafts that help align expectations and provide a foundation for further refinement during the UI development lifecycle.

Integrating a New Application

The following diagram illustrates the sequence of steps a user must follow when integrate an application through the midPoint graphical interface. This high-level flow also incorporates references to several sub-process diagrams, which are described later in the document.


Integrating new application
START
Show Application catalog
Application catalog
Add application
MidPoint has access to the Integration catalog.
Showing connectors from Integration catalog
Use search filters
Application connector found
MidPoint has access to the connector generation service
a request to the community to create a connector for your application
Show details
Show requested application
Show table with requested applications by community
Select one and create connector for it
Showing local connectors
Show only local connectors
Showing local connectors
Select connector, create application
Would you like to generate a new connector for your application with the help of an AI assistant?
Submit a request
END
Decision
Action
Not showed in GUI, backend processing
Display
Continue in separated flow diagram
Use search filters
Application connector found
Download and import connector to midpoint
Form with basic attributes for application
Completing the attributes and confirming the form
Generate new connector for application
Provide steps to searching connectors in Integration catalog web application and show possibility for connector import
Application connector found in Integration catalog
Form with basic attributes for resource
Subcription is provided
MidPoint has access to the Integration catalog
Select schema objects
Choices for object type or association type
Save settings
Selecting schema object class that will be connector use
Completing the attributes and confirming the form
Form with all configuration attributes of the connector
Does the connector support configuration discovery?
Completing the attributes and confirming the form
Panel for uploading of the connector to the Integration catalog
Open object types table
Open association types table
Create new association type wizard
We want to upload the connector to the Integration catalog
Object types table
Association types table
Create new association type
Completing the attributes and confirming the form
Form with other configuration attributes of the connector
Completing the attributes and confirming the form
Form with basic configuration attributes of the connector
Upload connector
Confirm upload
Provide the steps to upload connector to Integration catalog web application
MidPoint has access to the AI assistent service.
Create new object type
Form with basic attributes for object type
Completing the attributes and confirming the form
Form with resource date attributes for object type
Completing the attributes and confirming the form
Form with midpoint date attributes for object type
MidPoint has access to the AI assistent service.
Completing the attributes and confirming the form
Save settings
Use object type suggestion as predefined value
Use AI assistant for focus type
Show suggestions for focus type
Select focus type
Choices all configuration possibilities of object type
Want use delineation (Resource object type) suggestions?
Select object type
Show suggestions for object types
Use suggestions for object types
Chose basic attributes
Chose mappings
Chose synchronization
Chose correlation
Chose capabilities
Chose activation
Chose credentials
Chose policies
Wizard for mappings
Wizard for synchronization
Wizard for correlation
Wizard for capabilities
Wizard for activation
Wizard for credentials
Wizard for policies
Y
N

From the panel displaying an Application catalog, the user can access the functionality for integrating a new application.

The first step involves selecting an application to integrate. A list of available integrations is provided by the Integration catalog. If the desired application is not present in this list, the user can either leverage an AI assistant to generate a new connector for integrating with the new application, or request the creation of a new connector if they prefer not to generate one themselves.

Once an application is selected (or a new connector is generated), the user proceeds to configure basic attributes of resource that corresponds to the application. After this initial setup, users can use suggestions of object types that might exist, which are recommended to them by the AI assistant for delineation. Once the object type is selected, users can proceed with configuring the remaining basic attributes, including all necessary configuration aspects such as mappings, correlation rules, synchronization settings, and more.

If the required application is not found and the user does not wish to create a connector at that time, he may submit a request for connector creation. The integration process for that application can then be continued at a later.

The "Show Details" diagram is not explicitly included here, but it refers to the application’s detail view, where key aspects of the application are presented in separate panels. These panels are accessible via the details menu and allow users to view and manage specific parts of the application and corresponding resource configuration.

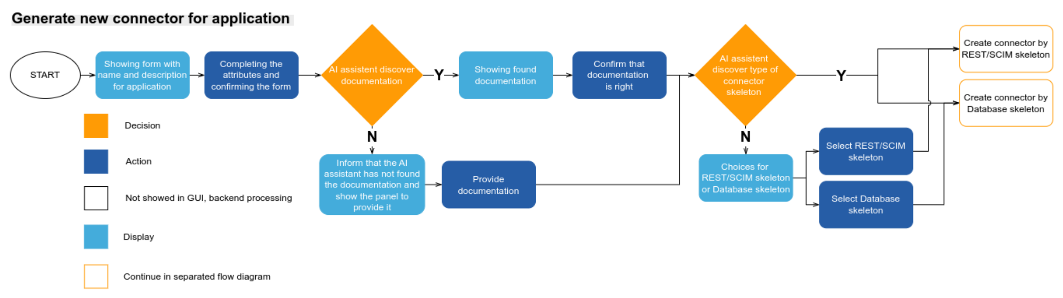
The diagrams titled "Wizard for Synchronization," "Wizard for Capabilities," and "Wizard for Policies" are already implemented as part of midPoint’s standard functionality. Since these wizards will be used without modification during the application creation process, they are not depicted in this document to avoid redundancy.

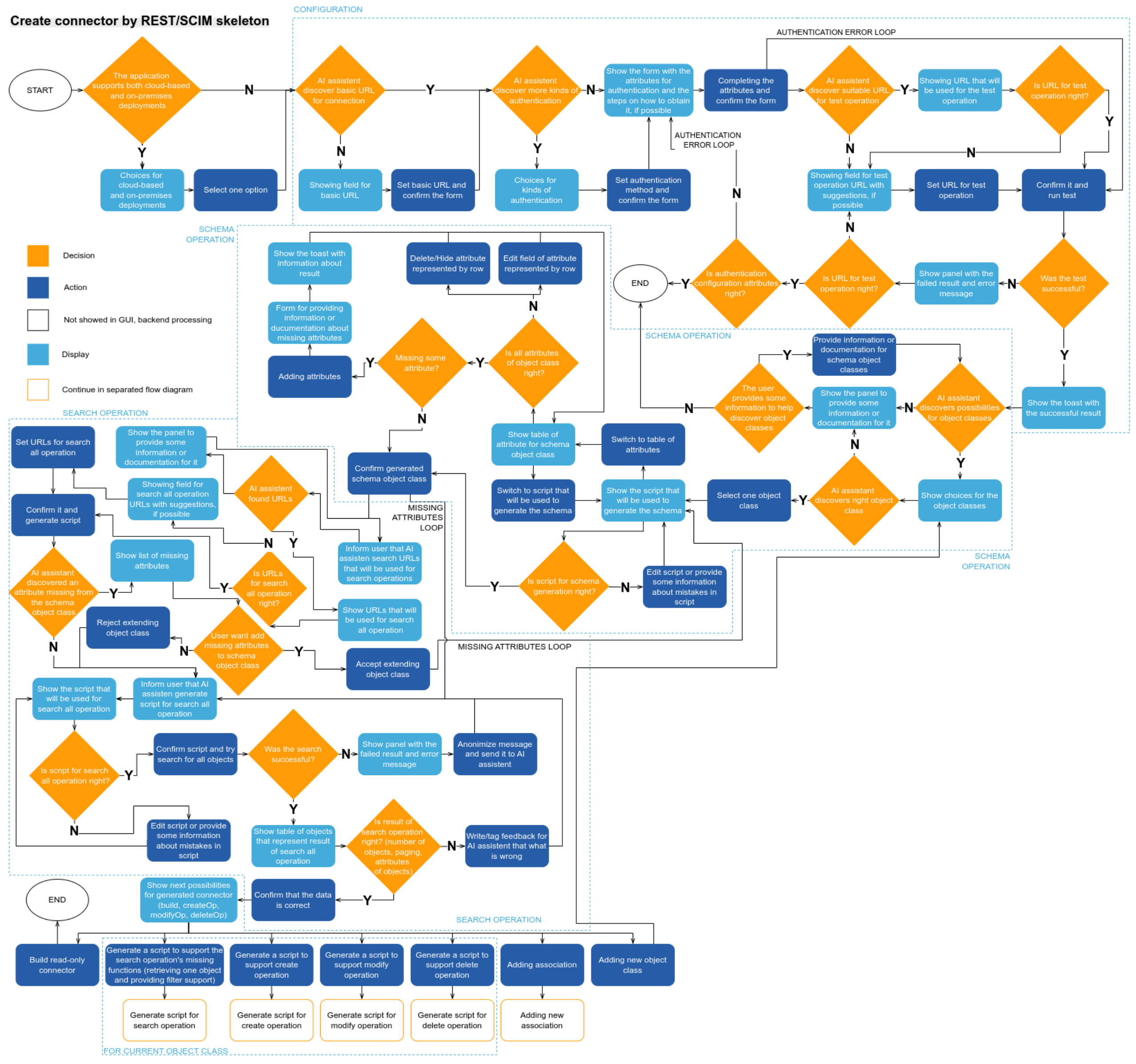
This structure ensures that only the modified or newly introduced parts of the process are illustrated in detail.

Generate New Connector for Application

This diagram illustrates the process of generating a connector for a new application that is not yet present in the Integration catalog.

It outlines the steps a user follows to define and create a new connector using available tools, such as the AI assistant or manual input.


Generate new connector for application
START
Showing form with name and description for application
Completing the attributes and confirming the form
AI assistent discover documentation
Y
Showing found documentation
Confirm that documentation is right
AI assistent discover type of connector skeleton
Y
Create connector by REST/SCIM skeleton
Create connector by Database skeleton
N
Inform that the AI assistant has not found the documentation and show the panel to provide it
Provide documentation
N
Choices for REST/SCIM skeleton or Database skeleton
Select REST/SCIM skeleton
Select Database skeleton
Decision
Action
Not showed in GUI, backend processing
Display
Continue in separated flow diagram


Create connector by REST/SCIM skeleton
CONFIGURATION
START
The application supports both cloud-based and on-premises deployments
N
Y
Choices for cloud-based and on-premises deployments
Select one option
AI assistent discover basic URL for connection
Y
N
Showing field for basic URL
Set basic URL and confirm the form
AI assistent discover more kinds of authentication
N
Y
Choices for kinds of authentication
Show the form with the attributes for authentication and the steps on how to obtain it, if possible
Set authentication method and confirm the form
AUTHENTICATION ERROR LOOP
AUTHENTICATION ERROR LOOP
Completing the attributes and confirm the form
AI assistent discover suitable URL for test operation
Y
Showing URL that will be used for the test operation
Is URL for test operation right?
Y
N
N
Showing field for test operation URL with suggestions, if possible
Set URL for test operation
Confirm it and run test
N
N
END
Is authentication configuration attributes right?
Y
Is URL for test operation right?
Show panel with the failed result and error message
N
Was the test successful?
Y
Show the toast with the successful result
SCHEMA OPERATION
SCHEMA OPERATION
Decision
Action
Not showed in GUI, backend processing
Display
Continue in separated flow diagram
Show the toast with information about result
Form for providing information or documentation about missing attributes
Adding attributes
Delete/Hide attribute represented by row
Edit field of attribute represented by row
N
Missing some attribute?
Y
Is all attributes of object class right?
Y
N
Show table of attribute for schema object class
Switch to table of attributes
Switch to script that will be used to generate the schema
Show the script that will be used to generate the schema
Select one object class
Y
AI assistant discovers right object class
Show choices for the object classes
Y
N
The user provides some information to help discover object classes
Y
Provide information or documentation for schema object classes
Show the panel to provide some information or documentation for it
N
AI assistant discovers possibilities for object classes
N
SCHEMA OPERATION
Confirm generated schema object class
MISSING ATTRIBUTES LOOP
Inform user that AI assisten search URLs that will be used for search operations
Show URLs that will be used for search all operation
Accept extending object class
MISSING ATTRIBUTES LOOP
Is script for schema generation right?
Y
N
Edit script or provide some information about mistakes in script
SEARCH OPERATION
Set URLs for search all operation
Show the panel to provide some information or documentation for it
Showing field for search all operation URLs with suggestions, if possible
AI assistent found URLs
Confirm it and generate script
N
Y
Show list of missing attributes
Is URLs for search all operation right?
Y
AI assistant discovered an attribute missing from the schema object class
Y
Reject extending object class
User want add missing attributes to schema object class
N
Y
N
Show the script that will be used for search all operation
Inform user that AI assisten generate script for search all operation
Is script for search all operation right?
Y
N
Confirm script and try search for all objects
Was the search successful?
N
Show panel with the failed result and error message
Anonimize message and send it to AI assistent
Y
Edit script or provide some information about mistakes in script
Show table of objects that represent result of search all operation
Is result of search operation right? (number of objects, paging, attributes of objects)
N
Write/tag feedback for AI assistent that what is wrong
END
Show next possibilities for generated connector (build, createOp, modifyOp, deleteOp)
Confirm that the data is correct
Y
SEARCH OPERATION
Build read-only connector
Generate a script to support the search operation's missing functions (retrieving one object and providing filter support)
Generate a script to support create operation
Generate a script to support modify operation
Generate a script to support delete operation
Adding association
Adding new object class
Generate script for search operation
Generate script for create operation
Generate script for modify operation
Generate script for delete operation
Adding new association
FOR CURRENT OBJECT CLASS

The diagram is divided into two parts. The first part illustrates the process of gathering documentation for a given application, highlighting the point at which it becomes necessary to more precisely define the type of application. The second part depicts the procedure for generating a connector for an application that communicates via REST/SCIM protocols.

The process may vary slightly for different types of applications. However, the application using REST/SCIM has the most complex workflow, which is why it is specifically demonstrated in the diagram.

During the procedure, it is necessary to configure several components. First, the configuration settings themselves need to be established.

As the next step, the user selects the object class he wants to support in the connector. The available object classes are suggested by the AI assistant. After selecting a specific object class, the first required action is to generate a schema with all the necessary attributes the user wants to support for that object class. The schema is generated by the AI assistant, and the user either approves it or modifies the necessary attributes if the generated schema is not entirely accurate.

The process then continues with the configuration of the search operation, specifically the search all function, which is essential for the connector to work correctly with the selected object class. The user can verify the correctness of the search all function by running it and checking the results.

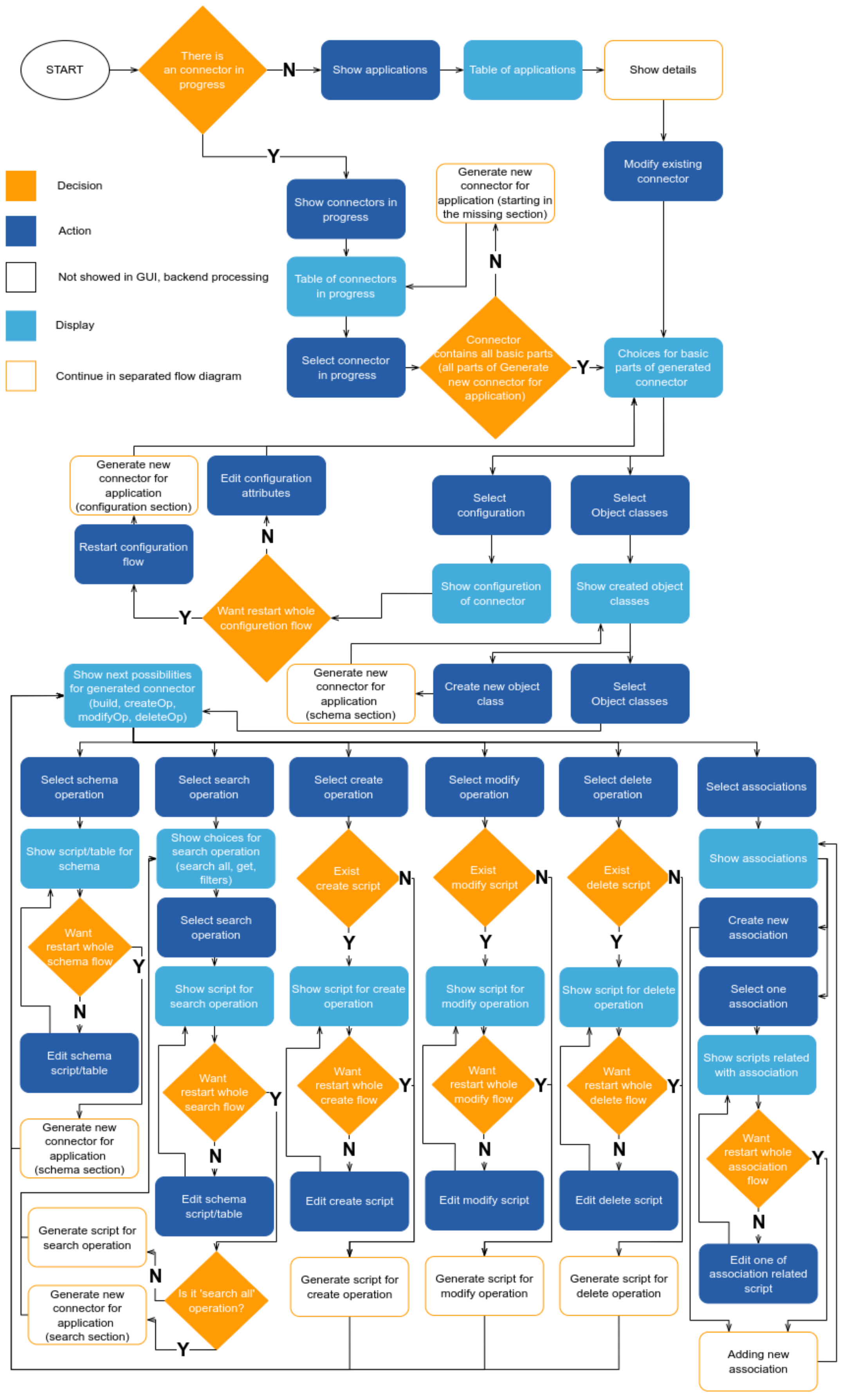
Subsequent steps allow the user to optionally define additional operations, such as create, modify, and delete. These steps are not mandatory but provide extended functionality for the connector to fully support CRUD (Create, Read, Update, Delete) operations. Additional options include extending the search operation with support for more filters, and creating an association involving the chosen object class.

Overall, the diagram offers a clear, step-by-step representation of how to obtain necessary documentation and generate a fully functional connector for a REST/SCIM-based application, with flexibility for other application types as well.

Modify Existing Connector

The following diagram illustrates the option to return to a connector that is still in progress and has not yet been completed, or to add extended functionality to an already existing connector. This includes adding unsupported operations, search filters, associations, or object classes.

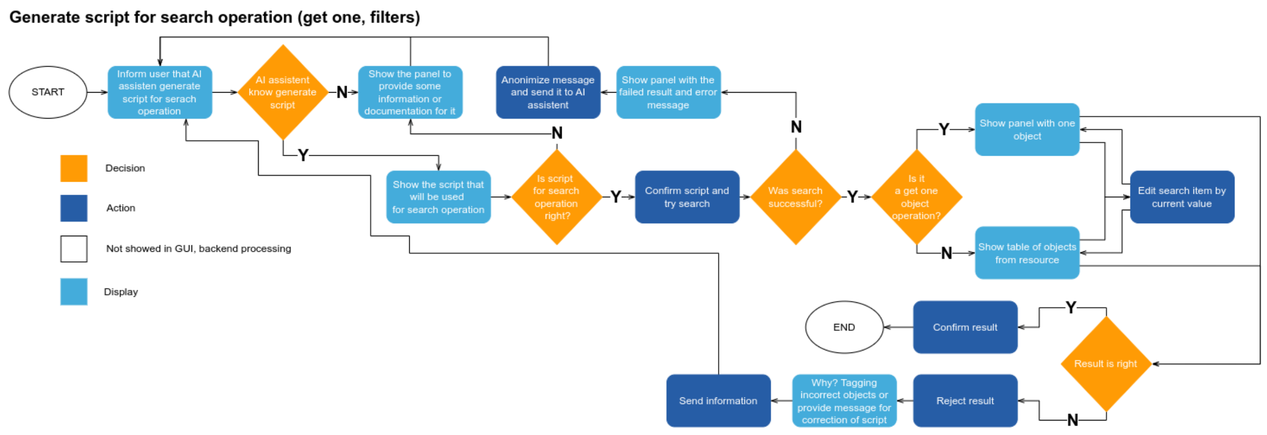
Modify existing (generated) connector (REST/SCIM)


START
There is an connector in progress
N
Y
Show applications
Table of applications
Show details
Modify existing connector
Decision
Action
Not showed in GUI, backend processing
Display
Continue in separated flow diagram
Show connectors in progress
Generate new connector for application (starting in the missing section)
Table of connectors in progress
Select connector in progress
Connector contains all basic parts (all parts of Generate new connector for application)
Choices for basic parts of generated connector
Generate new connector for application (configuration section)
Edit configuration attributes
Select configuration
Select Object classes
Restart configuration flow
Want restart whole configuretion flow
Show configuretion of connector
Show created object classes
Show next possibilities for generated connector (build, createOp, modifyOp, deleteOp)
Generate new connector for application (schema section)
Create new object class
Select Object classes
Select schema operation
Select search operation
Select create operation
Select modify operation
Select delete operation
Select associations
Show script/table for schema
Show choices for search operation (search all, get, filters)
Exist create script
Exist modify script
Exist delete script
Show associations
Want restart whole schema flow
Select search operation
Create new association
Show script for search operation
Show script for create operation
Show script for modify operation
Show script for delete operation
Select one association
Edit schema script/table
Want restart whole search flow
Want restart whole create flow
Want restart whole modify flow
Want restart whole delete flow
Show scripts related with association
Generate new connector for application (schema section)
Want restart whole association flow
Edit schema script/table
Edit create script
Edit modify script
Edit delete script
Generate script for search operation
Edit one of association related script
Is it 'search all' operation?
Generate script for create operation
Generate script for modify operation
Generate script for delete operation
Generate new connector for application (search section)
Adding new association

This process allows users to revisit and continue development on partially finished connectors, ensuring flexibility and iterative improvement. Additionally, it supports the enhancement of existing connectors by integrating new features or capabilities that were not initially supported, thereby increasing the connector's functionality and adaptability to changing requirements.

Filter of Search Operation

This diagram illustrates the sequence of steps involved in supporting filters for the search operation. The diagram consists of two main parts the generation of the script and followed by the verification of the results.

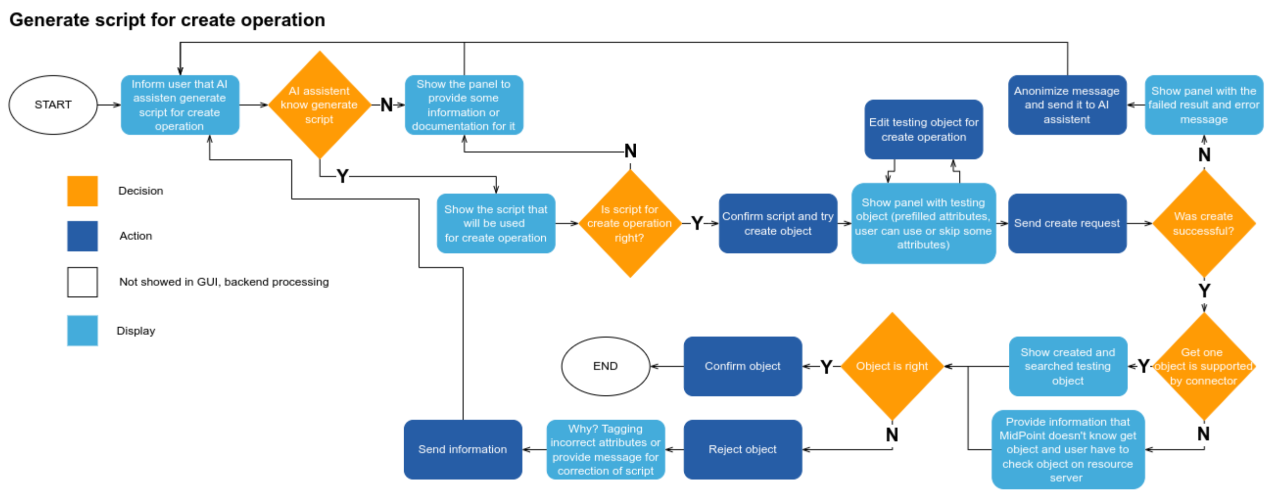

Generate script for search operation (get one, filters)
START
Inform user that AI assisten generate script for serach operation
AI assistent know generate script
N
Y
Show the panel to provide some information or documentation for it
Anonimize message and send it to AI assistent
Show panel with the failed result and error message
N
N
Show the script that will be used for search operation
Is script for search operation right?
Y
Confirm script and try search
Was search successful?
Y
Is it a get one object operation?
Y
N
Show panel with one object
Show table of objects from resource
Edit search item by current value
Decision
Action
Not showed in GUI, backend processing
Display
END
Confirm result
Y
Result is right
N
Send information
Why? Tagging incorrect objects or provide message for correction of script
Reject result

Verification is performed either through a table displaying the returned objects or by confirming the correctness of a single object if the operation was "get one" object.

This process ensures that search filters are properly implemented and validated, enabling precise and efficient querying within the application. It helps users confirm that the filter criteria yield the expected results, whether multiple objects or a specific single object.

Create Operation

This diagram represents the configuration process for the create operation.


Generate script for create operation
START
Inform user that AI assisten generate script for create operation
AI assistent know generate script
N
Show the panel to provide some information or documentation for it
Y
Show the script that will be used for create operation
Is script for create operation right?
N
Y
Confirm script and try create object
Edit testing object for create operation
Show panel with testing object (prefilled attributes, user can use or skip some attributes)
Send create request
Was create successful?
N
Anonimize message and send it to AI assistent
Show panel with the failed result and error message
Y
Get one object is supported by connector
Y
Show created and searched testing object
N
Provide information that MidPoint doesn't know get object and user have to check object on resource server
Object is right
Y
Confirm object
END
N
Reject object
Why? Tagging incorrect attributes or provide message for correction of script
Send information
Decision
Action
Not showed in GUI, backend processing
Display

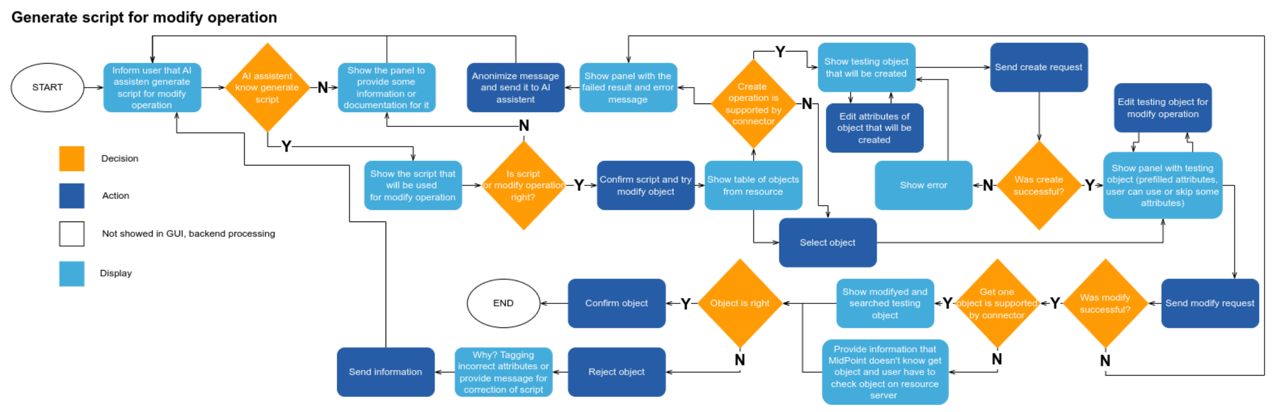
The first step is to validate the correctness of the generated script. Once the correctness of the script is confirmed, the next stage involves testing the operation.

In this phase, a test object can be created to verify that the operation behaves as expected. If the connector already supports the "get one" search operation, the test object can be verified directly in midPoint. Otherwise, the verification must be performed within the target application itself.

This structured approach ensures that the create operation is not only implemented but also thoroughly tested, minimizing the risk of errors during deployment. It also provides flexibility in the verification method based on the connector's current capabilities.

Modify Operation

This diagram represents the configuration process for the modify operation.


Generate script for modify operation
START
Inform user that AI assisten generate script for modify operation
AI assistent know generate script
N
Show the panel to provide some information or documentation for it
Anonimize message and send it to AI assistent
Show panel with the failed result and error message
Y
Show the script that will be used for modify operation
Is script or modify operation right?
N
Y
Confirm script and try modify object
Show table of objects from resource
Create operation is supported by connector
Y
N
Show testing object that will be created
Edit attributes of object that will be created
Send create request
Was create successful?
N
Show error
Y
Edit testing object for modify operation
Show panel with testing object (prefilled attributes, user can use or skip some attributes)
Select object
Send modify request
Was modify successful?
Y
N
Get one object is supported by connector
Y
N
Show modifyed and searched testing object
Provide information that MidPoint doesn't know get object and user have to check object on resource server
Object is right
Y
Confirm object
END
N
Reject object
Why? Tagging incorrect attributes or provide message for correction of script
Send information
Decision
Action
Not showed in GUI, backend processing
Display

Similar to the create operation, the workflow follows the same steps, first, the generated script must be validated, and then the modify operation itself is tested.

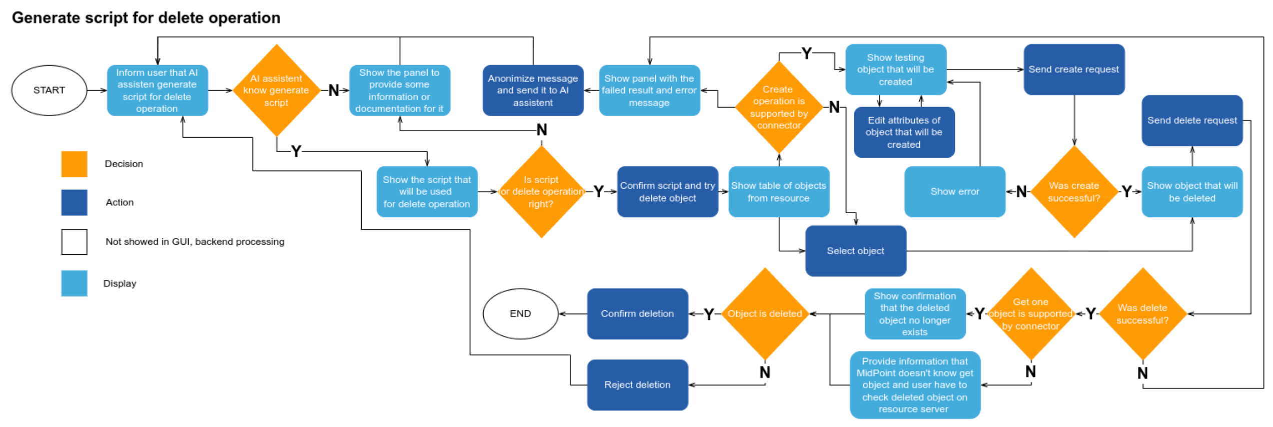
During the testing phase, it is necessary to select an existing test object — or create new, if the connector already supports it — on whom the modify operation will be executed.

As with the 'create' operation, the results can be verified directly in midPoint if the connector supports the "get one" search operation. Otherwise, verification must be performed within the target application.

This approach ensures that the modify operation is correctly implemented and properly validated, providing consistency and reliability in how object data is updated across connected systems.

Delete Operation

The last of the basic operations for the object class is the delete operation.


Generate script for delete operation
START
Inform user that AI assisten generate script for delete operation
AI assistent know generate script
N
Y
Show the panel to provide some information or documentation for it
Anonimize message and send it to AI assistent
Show panel with the failed result and error message
Create operation is supported by connector
Y
N
Show testing object that will be created
Edit attributes of object that will be created
Send create request
Send delete request
Decision
Action
Not showed in GUI, backend processing
Display
Show the script that will be used for delete operation
Is script or delete operation right?
Y
N
Confirm script and try delete object
Show table of objects from resource
Show error
N
Was create successful?
Y
Show object that will be deleted
Select object
END
Confirm deletion
Y
Object is deleted
N
Show confirmation that the deleted object no longer exists
Y
Get one object is supported by connector
Y
Was delete successful?
N
N
Reject deletion
Provide information that MidPoint doesn't know get object and user have to check deleted object on resource server

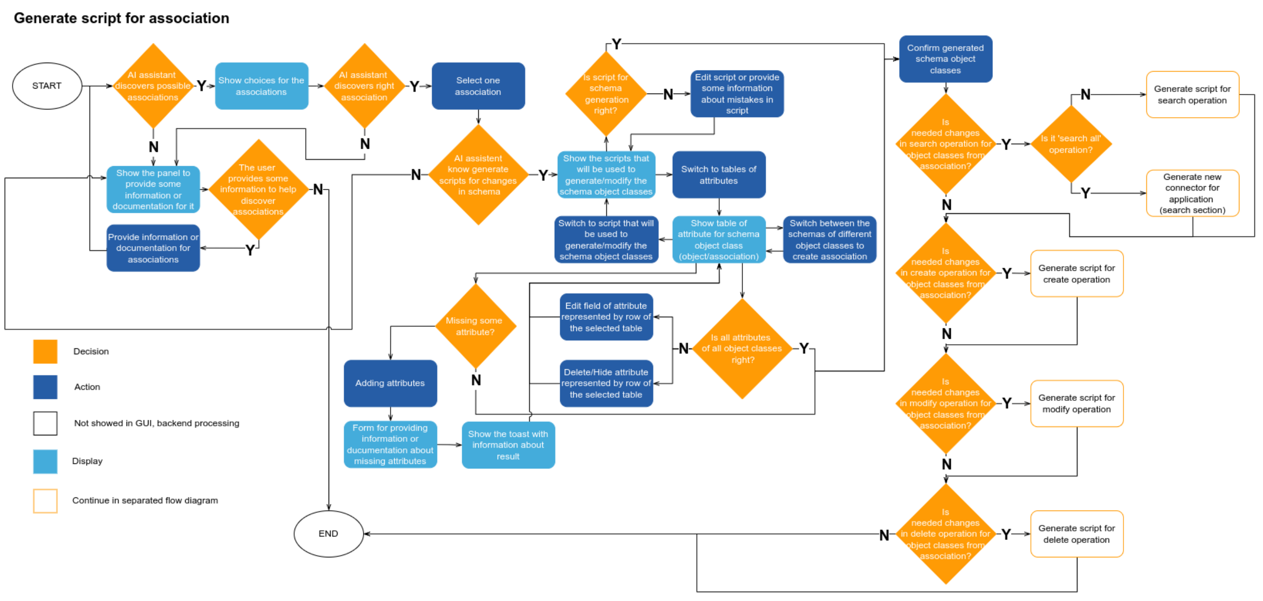
Just like with the create and modify operations, the first step is to confirm the correctness of the generated script, followed by verification of the operation's result.

For verification, a test object must be selected or created, which will then be deleted using the configured operation. To ensure the object has been successfully removed, the deletion can be verified in midPoint, if the connector supports the "get one" search operation, or alternatively, by checking directly in the target application to confirm the object is no longer present.

This step ensures that the delete operation works as intended and that the connector correctly handles object removal, maintaining data consistency between midPoint and the connected application.

Association

The following diagram illustrates the configuration of an association between two object classes.


Generate script for association
START
AI assistant discovers possible associations
Y
Show choices for the associations
AI assistant discovers right association
Y
Select one association
N
N
Show the panel to provide some information or documentation for it
The user provides some information to help discover associations
N
Y
Provide information or documentation for associations
AI assistent know generate scripts for changes in schema
N
Y
Is script for schema generation right?
Y
N
Edit script or provide some information about mistakes in script
Show the scripts that will be used to generate/modify the schema object classes
Switch to tables of attributes
Switch to script that will be used to generate/modify the schema object classes
Show table of attribute for schema object class (object/association)
Switch between the schemas of different object classes to create association
Missing some attribute?
N
Adding attributes
Form for providing information or ducumentation about missing attributes
Show the toast with information about result
Edit field of attribute represented by row of the selected table
Delete/Hide attribute represented by row of the selected table
Is all attributes of all object classes right?
N
Y
Confirm generated schema object classes
Is needed changes in search operation for object classes from association?
Y
Is it 'search all' operation?
N
Generate script for search operation
Y
Generate new connector for application (search section)
N
Is needed changes in create operation for object classes from association?
Y
Generate script for create operation
N
Is needed changes in modify operation for object classes from association?
Y
Generate script for modify operation
N
Is needed changes in delete operation for object classes from association?
Y
Generate script for delete operation
N
END
Decision
Action
Not showed in GUI, backend processing
Display
Continue in separated flow diagram

The user begins by selecting from a list of available associations that can be configured within the system.

Once an association is selected, the necessary scripts required to ensure the association functions correctly are displayed. These scripts may include schema extensions for the involved object classes, or modifications to specific operation scripts (such as search, create, modify or delete) that need to be adjusted to support the association.

This process provides a structured way to establish relationships between different object classes — such as linking users with roles or groups — while ensuring that all necessary script modification is applied for the association to operate reliably across all supported operations.

Suggestions Panels

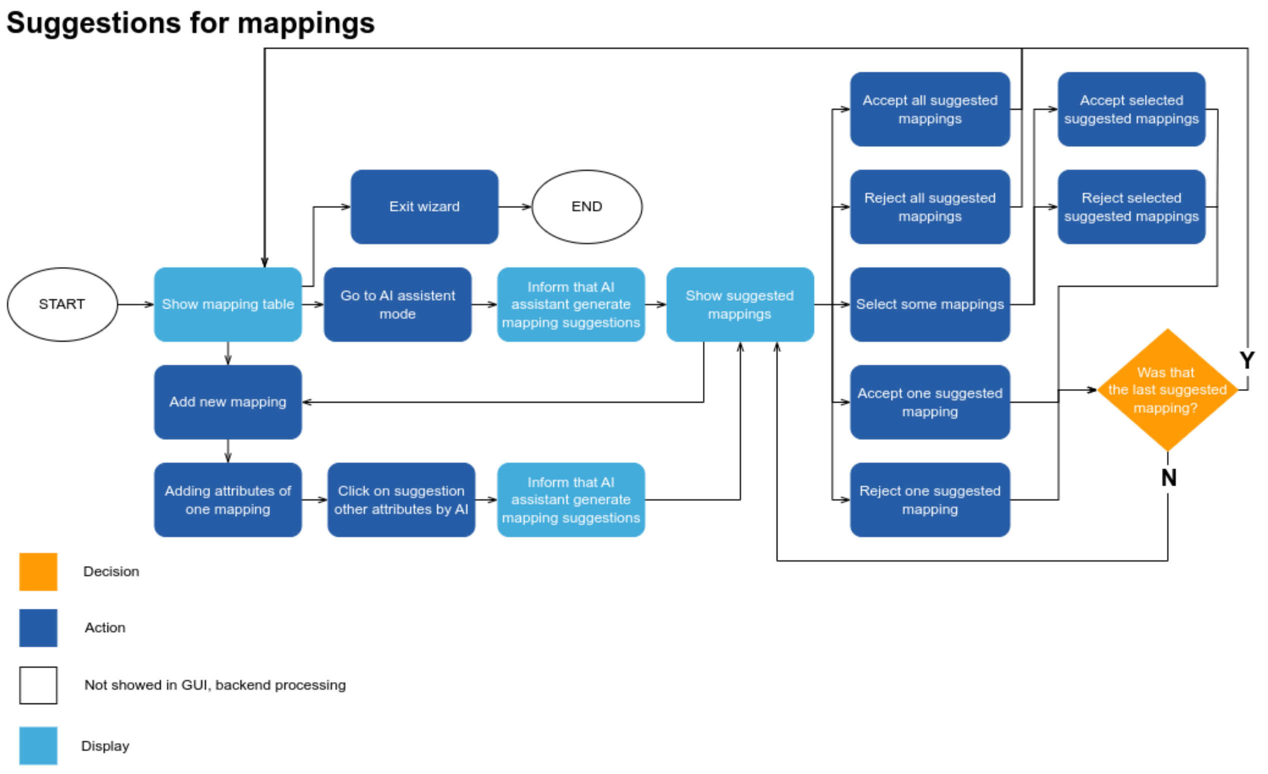
These diagrams represent partial wizards that are linked to specific configurations of the ResourceType. Each wizard guides the user through a focused configuration task related to the resource.

All of these wizards have been enhanced with the ability to provide suggestions, reducing the need for manual configuration of the main components.

These guided wizards serve as helpful tools, especially for less experienced users, by simplifying complex tasks and offering intelligent assistance throughout the configuration process.

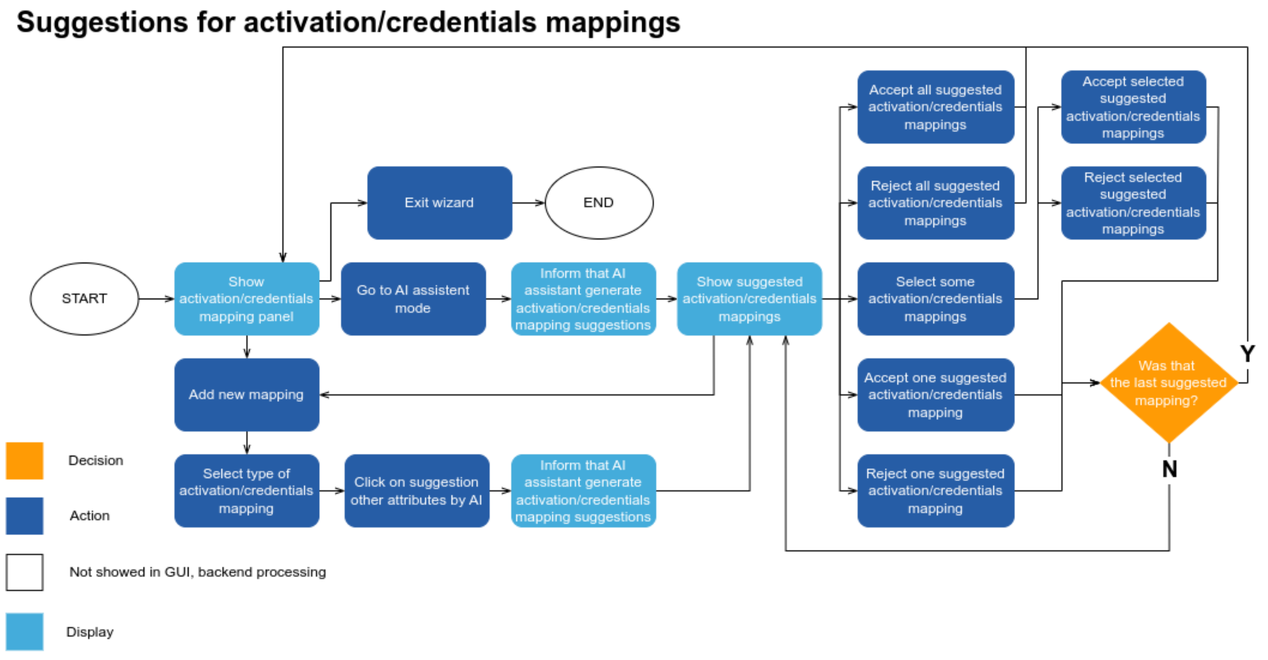
Suggestions for Mappings

The following diagram illustrates the process of displaying and then either accepting or rejecting suggested mappings. These suggestions can be specifically targeted toward a particular attribute, either in midPoint or on the resource side.


Suggestions for mappings
START
Show mapping table
Go to AI assistent mode
Inform that AI assistant generate mapping suggestions
Show suggested mappings
Exit wizard
END
Add new mapping
Adding attributes of one mapping
Click on suggestion other attributes by AI
Inform that AI assistant generate mapping suggestions
Accept all suggested mappings
Reject all suggested mappings
Select some mappings
Accept one suggested mapping
Reject one suggested mapping
Accept selected suggested mappings
Reject selected suggested mappings
Was that the last suggested mapping?
Y
N
Decision
Action
Not showed in GUI, backend processing
Display

Suggestions for Activation/Credentials Mappings

This diagram operates on the same principle as the previous one but focuses specifically on activation and credentials mappings.

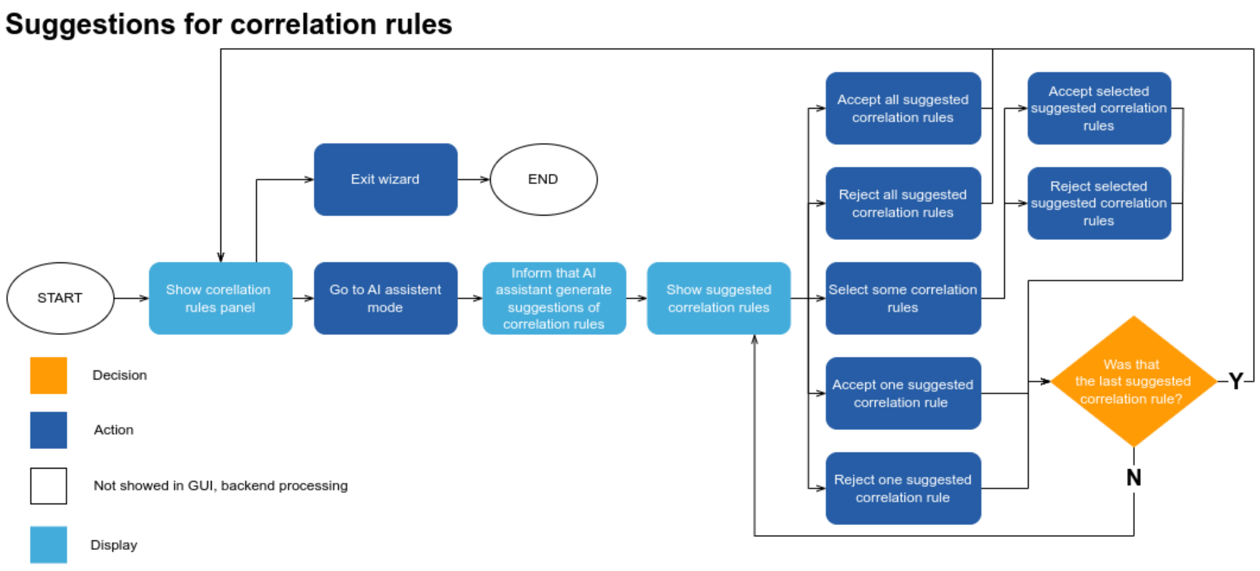

Suggestions for activation/credentials mappings
START
Show activation/credentials mapping panel
Go to AI assistent mode
Inform that AI assistant generate activation/credentials mapping suggestions
Show suggested activation/credentials mappings
Exit wizard
END
Add new mapping
Select type of activation/credentials mapping
Click on suggestion other attributes by AI
Inform that AI assistant generate activation/credentials mapping suggestions
Accept all suggested activation/credentials mappings
Reject all suggested activation/credentials mappings
Select some activation/credentials mappings
Accept one suggested activation/credentials mapping
Reject one suggested activation/credentials mapping
Accept selected suggested activation/credentials mappings
Reject selected suggested activation/credentials mappings
Was that the last suggested mapping?
Y
N
Decision
Action
Not showed in GUI, backend processing
Display

It illustrates the process of reviewing, accepting, or rejecting suggested mappings related to user activation status (such as enable/disable) and credential-related attributes (such as password).

As opposed to attribute mappings, these suggestions may target only to midPoint attributes.

Suggestions for Correlation Rules

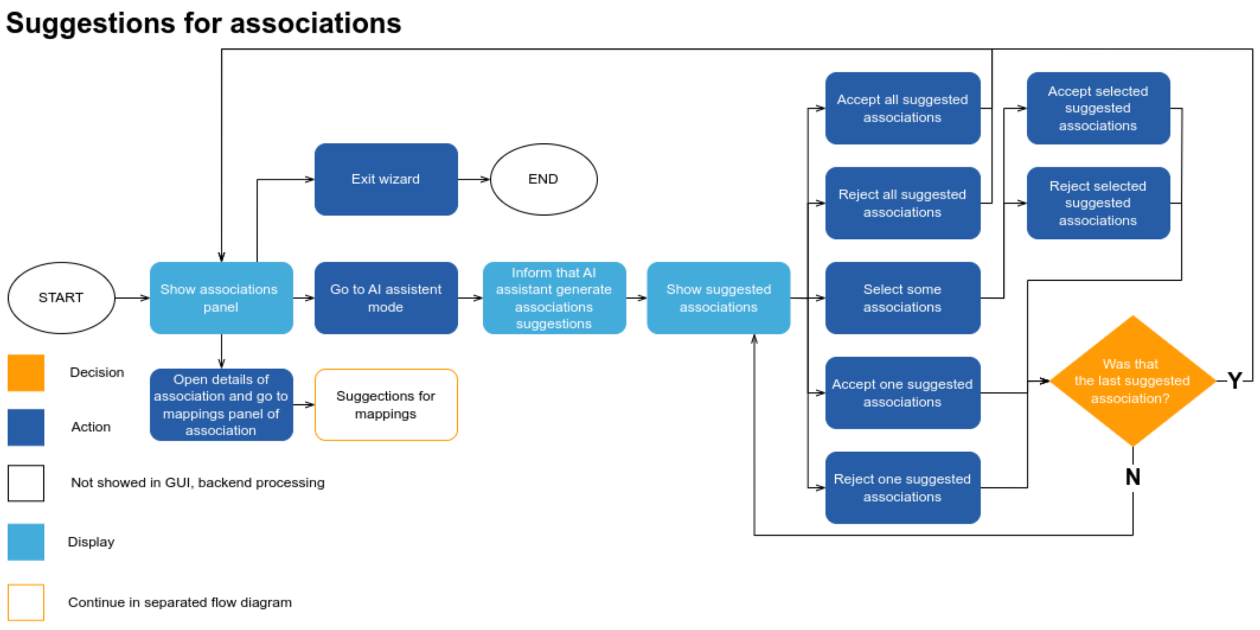
This diagram illustrates the suggestions for correlation rules, along with options to either accept or reject them.


Suggestions for correlation rules
START
Show corellation rules panel
Exit wizard
END
Go to AI assistent mode
Inform that AI assistant generate suggestions of correlation rules
Show suggested correlation rules
Accept all suggested correlation rules
Reject all suggested correlation rules
Select some correlation rules
Accept one suggested correlation rule
Reject one suggested correlation rule
Accept selected suggested correlation rules
Reject selected suggested correlation rules
Was that the last suggested correlation rule?
Y
N
Decision
Action
Not showed in GUI, backend processing
Display

The user is given the ability to review each proposed rule, assess its relevance, and decide whether to incorporate it into the configuration.

Suggestions for Associations

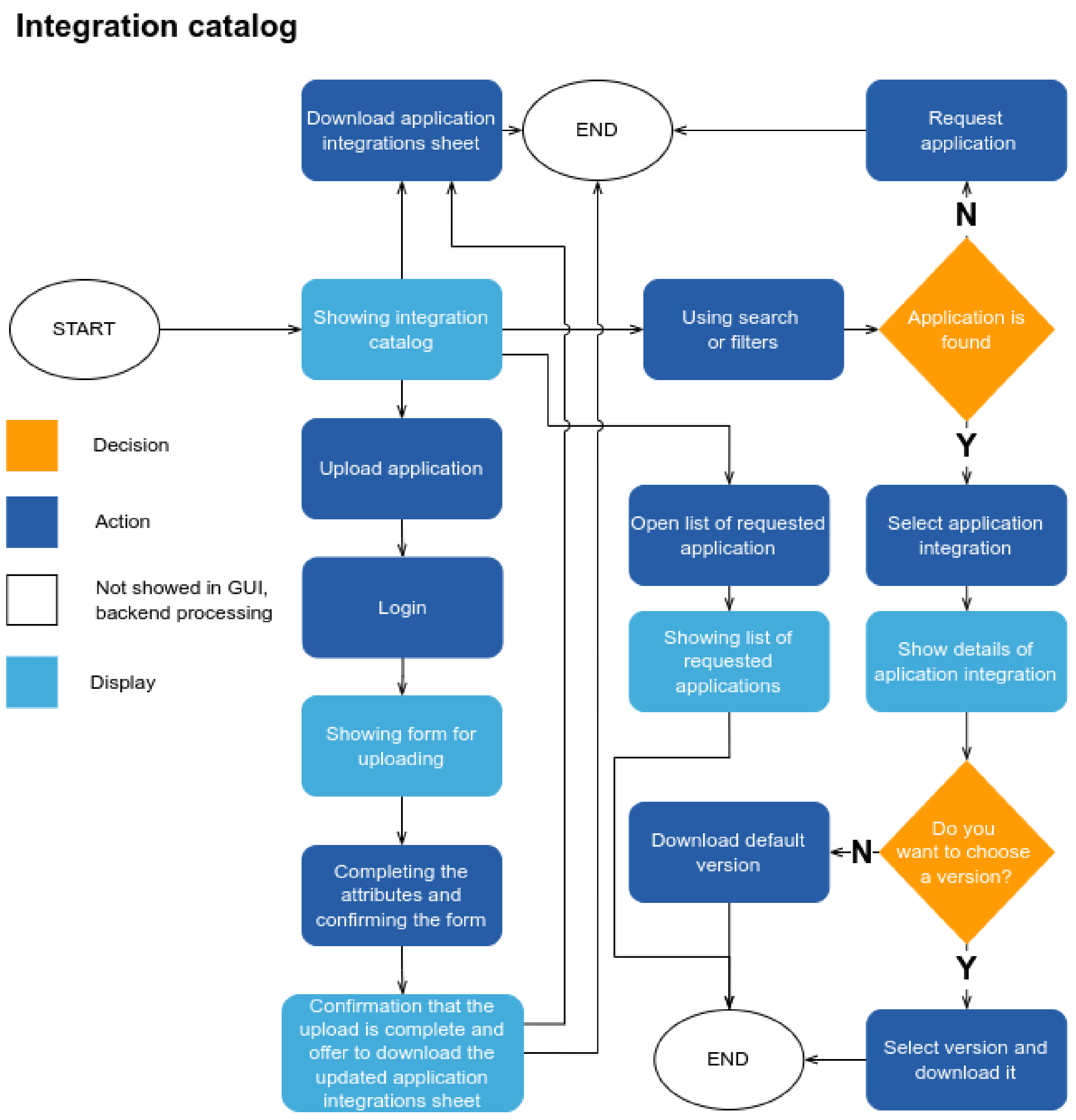
The last diagram in this section shows the sequence for recommending associations.


Suggestions for associations
START
Show associations panel
Exit wizard
END
Go to AI assistent mode
Inform that AI assistant generate associations suggestions
Show suggested associations
Open details of association and go to mappings panel of association
Suggections for mappings
Accept all suggested associations
Reject all suggested associations
Select some associations
Accept one suggested associations
Reject one suggested associations
Accept selected suggested associations
Reject selected suggested associations
Was that the last suggested association?
Y
N
Decision
Action
Not showed in GUI, backend processing
Display
Continue in separated flow diagram

Just like in previous diagrams, these suggestions can be either accepted or rejected by the user. Additionally, during the association setup, the user can also take advantage of suggested mappings for the association object.

Integration Catalog

The last diagram illustrates the interaction flow within the Integration catalog service.


Integration catalog
START
Showing integration catalog
Download application integrations sheet
END
Request application
N
Using search or filters
Application is found
Y
Decision
Action
Not showed in GUI, backend processing
Display
Upload application
Login
Showing form for uploading
Completing the attributes and confirming the form
Confirmation that the upload is complete and offer to download the updated application integrations sheet
Open list of requested application
Showing list of requested applications
Select application integration
Show details of aplication integration
Do you want to choose a version?
N
Download default version
Y
Select version and download it
END

The Integration catalog provides a centralized view of all available application integrations. It allows users to browse and filter existing integrations, download or upload new application integration (connector) and configuration of application, and retrieve a full signed list of available integrations.

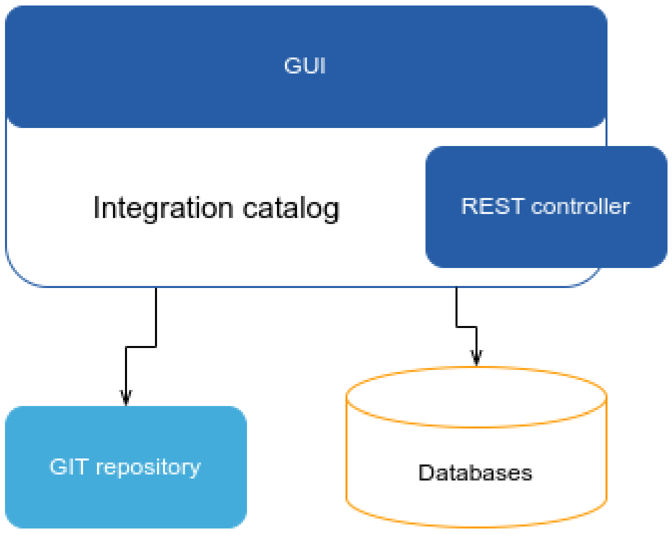
If the user cannot find the application they need, they can request its creation. Additionally, if someone from the community wishes to contribute by creating a connector, they can view a list of the most requested applications.

4.9 Integration Catalog

In this section, we will take a closer look at the Integration Catalog, exploring its architecture and core functionalities in more detail. We will examine how the catalog is structured and what capabilities it offers to users—from connector management to configuration handling.

4.9.1 Architecture

The Integration catalog consists of multiple architectural layers, each serving a specific role within the system.



- One of the core components is the graphical user interface (GUI), which provides interaction with end users. This interface is publicly accessible and exposes all core functionalities of the integration catalog, including browsing, searching, uploading, and downloading application connectors, as well as user registration and authentication.
- Another important layer is the REST controller, which enables communication with other services, primarily with midPoint and Studio. This layer is responsible for handling external API requests and ensuring secure and structured data exchange between the catalog and connected systems.
- For data storage, the Integration catalog relies on two databases. The first is a relational database used to store all essential information, such as available application connectors with metadata and user accounts. The second is a specialized database for storing configurations of integrated applications (mappings, correlation rules, association, etc.) collected from individual deployments. This configuration data is used to improve AI assistant recommendations and to gather statistical insights for evaluating the assistant's

effectiveness. This data layer ensures consistency, reliability, and efficient access to information required by both the graphical user interface and backend services.

- Another layer of the Integration catalog is a Git repository, which serves as the storage for the source code of uploaded connectors and compiled JAR files of uploaded connectors. Whenever a user uploads a new connector, the system automatically pushes its source code into the Git repository. This mechanism ensures version control, change tracking, and provides a transparent history of all modifications made to the connectors over time.

Together, these layers form a modular and scalable architecture that supports the catalog's operational needs while ensuring data integrity, and user accessibility.

4.9.2 Function

The system provides both an API and a graphical user interface (GUI), giving users flexible access to key features.

One of the core capabilities is the ability to view all currently available application connectors. Each connector includes detailed metadata such as its name, description, capabilities, popularity, and the type of system it integrates with, allowing users to easily understand its purpose and technical scope.

To simplify navigation and help users find specific connectors within a potentially large catalog, users can filter and search based on various criteria. These include attributes like the connector's name, type, capabilities, and popularity, making the discovery process efficient and user-friendly.

Another key feature is the ability to upload new integrations into the catalog. Users can either publish entirely new connectors or upload configuration files that define how two systems should communicate. When uploading a connector, the process typically involves providing the connector file itself, along with a description and essential metadata such as the version, the type of system it supports, and its technical specification. Alternatively, users may upload integration configurations, which define how applications communicate through the connector. These configurations may include mappings, correlation rules, or associations, and must be accompanied by metadata that describes their origin. This functionality is available only to authenticated users.

Users can also download individual application connectors. This enables them to import the downloaded connectors into the midPoint system and begin using them as part of their integration processes.

To provide additional security and trust, the catalog includes access to a digitally signed list of all available connectors. This list acts as a tamper-proof, verified snapshot of the system's current state and is used in midPoint to verify the authenticity of the connectors in use.

If a desired connector is not available, users can submit a request for it. These requests are tracked and presented in a dedicated view, offering visibility into user needs and guiding future development priorities.

One of the key features of the Integration Catalog is the ability to provide integration configurations—specifically, resource configurations that have been uploaded by authenticated users. These configurations are made available to the Smart Integration Service, which takes them into account when generating new suggestions within midPoint or midPoint Studio.

Each configuration is associated with a specific application and object class, ensuring contextual relevance. By leveraging real-world, user-contributed configurations, the system can significantly improve the accuracy and quality of its suggestions. This enables Smart Integration to offer suggestions that are better aligned with common integration scenarios and practical use cases.

In addition to enhancing suggestion quality, the collected configurations also serve a secondary purpose: they are used to evaluate the accuracy and effectiveness of the suggestions provided by the Smart Integration Service. This feedback loop allows the service to continuously refine its models and adapt to actual user needs and behaviors.

This functionality is strictly secured and accessible only to the Smart Integration Service. Access to upload and consume these configurations is controlled through user authentication to ensure that only authorized systems can contribute or retrieve this sensitive integration data.

To maintain a secure and accountable environment, the catalog implements user registration and authentication features. Only authorized users are allowed to upload connectors or configuration, which helps protect the system from misuse and ensures traceability of uploaded components.

4.10 Validation Service

One project goal is simplifying onboarding new applications to midPoint. The system will dynamically generate various artifacts using AI/ML techniques.

Because these techniques are non-deterministic, the generated artifacts are also non-deterministic and require validation. Validation will differ depending on the artifact type (e.g., Groovy code vs. XML snippets).

The validation service will also provide feedback to the AI/ML services to improve generated artifacts. For example, it might report missing XML parts, which the AI/ML can then use to fix the issues. Artifact validation will be available through a REST API.

Given the types of artifacts our recommendation services and code generators will produce, we must provide validators for two types of generated content:

- XML

- Groovy

The validation service is implemented as a microservice, based on the Spring Boot framework, providing REST endpoints. The request contains the generated code snippet in the request body and the code type (XML/Groovy) in the request headers. The validation result contains a list of all issues found.

XML Validator

The XML Validator is designed to validate XML configurations or snippets specifically targeted for midPoint. It leverages the Prism framework's parsing context in compatibility mode to ensure correctness and schema consistency with midPoint's internal data model.

Currently, XML configuration validation (mappings, correlations, etc.) is performed according to the midPoint schema. This provides good syntax validation of XML configurations.

Design

Our main requirements for the XML validator are to ensure that the XML is:

- Syntactically correct
- Parseable into Prism objects (midPoint's internal object representation).

Validation Against XSD Schema

Several libraries can validate XML against XSD schemas. We tried:

- Xerces (bundled in the JDK)
- Woodstox (FasterXML project)
- XMLUnit (primarily a testing library)

However, regardless of configuration and library, we identified limitations:

- Validation stops at the first error (with some exceptions). For example, if three elements are mandatory but the XML only contains the first, the error message only reports the missing second element.
- Validation fails if XML element order differs from the XSD. This is technically correct for `<sequence>` elements in XSDs, but the order is practically insignificant in midPoint and causes more problems than benefits.

After discussion, we relaxed the schema validation requirement, partially substituting it with parsing into Prism objects (generated from the schema).

Validation Using Prism Parsing

Our next approach leveraged the Prism framework's parsing context in compatibility mode to ensure correctness and schema consistency with midPoint's internal data model.

Prism, by Evolveum, is a data representation layer framework, parsing data from formats like XML, JSON, or YAML and making it available to Java applications abstractly.

The existing midPoint schema is used to parse XML objects, capturing detected errors and warnings. Compatibility mode allows processing to continue as long as the data is roughly schema-compatible, silently ignoring illegal values and unknown elements. Parsing starts by creating an object type definition (an input parameter for the validator). If a different `objectType` from the XML configuration is needed, the optional `itemPath` parameter specifies the object type for Prism parsing definition.

Currently, XML configuration validation (mapping, correlations, etc.) is performed according to the midPoint schema, providing good syntax validation but not semantic validation. We may require semantic validation in later project stages, requiring further research.

Groovy Validator

The Groovy Validator ensures that dynamically generated Groovy code is syntactically and semantically correct before execution. Groovy code will be generated primarily for connectors but may also be generated for mappings and other needs. The validator should verify syntactic correctness and static compilability.

Design

Our initial Groovy code validation requirements were:

- Syntactic correctness
- Static compilability
- Ability to call code outside the standard library

Our initial PoC validation using `GroovyShell` revealed that validator implementation depends heavily on the generated code's structure, specifically calls to "external code". Calls to the standard library are not problematic, but if the script caller binds custom methods/variables to the script at runtime, these must be passed to the validation service for static compilation.

We identified several approaches for handling calls to external code:

- **Variable bindings**: Binding an object to a variable accessible from the code at runtime. Disadvantages: less readable, exposes Groovy builder usage.
- **Method closure bindings**: Binding a variable as a `MethodClosure`. We couldn't achieve successful compilation and execution. While compilation worked, the script failed at

runtime.

- **Base script class**: Limits how "external code" is passed to the Groovy code. Requires defining all accessed methods/variables in the base script class.

All approaches require the validation service to be aware of and have access to the "external code."

After discussions, we chose the base script class approach.

Our research resulted in a prototype using `GroovyShell` to parse and evaluate user-provided Groovy code snippets.

Ensuring code safety

Besides syntactic correctness, it is necessary to ensure that the code is safe, i.e., that it does not harm its environment in any way. We plan to implement LLM guardrails to do this.

4.11 AI Studio

The aim of the AI Studio is to provide a proper set of tools and integrations to our ML engineers, DevOps engineers, and developers that facilitate, and potentially automate, their day-to-day work. In the context of the entire project, this means providing tools that support several parts of MLOps.

4.11.1 Chosen Toolset

After assessing various available open source tools that provide MLOps functionalities, we selected a few tools that fulfill the requirements. However, for LLM observability tools, we still have several similar options to evaluate in practice. Below is a table with the chosen toolset mapped to the requirements they fulfill.

Tool	Fulfills alone	Fulfills in combination with another tool from the table	Fulfills in combination with CI/CD tooling
ClearML	R5, R6	R7	R2, R3, R4, R7, R8
LangFuse/LangTrace/Phoenix/Opik	R1, R5, R9	R7, R8	-
JupyterLab	-	R2, R4	-
Ollama	-	-	R3

ClearML: This is the central tool, playing a role in fulfilling most of our requirements. It serves a crucial role in experimentation and testing as our main orchestrator. It also provides abilities to store and version datasets, track experiments, visualize experiment results, and more.

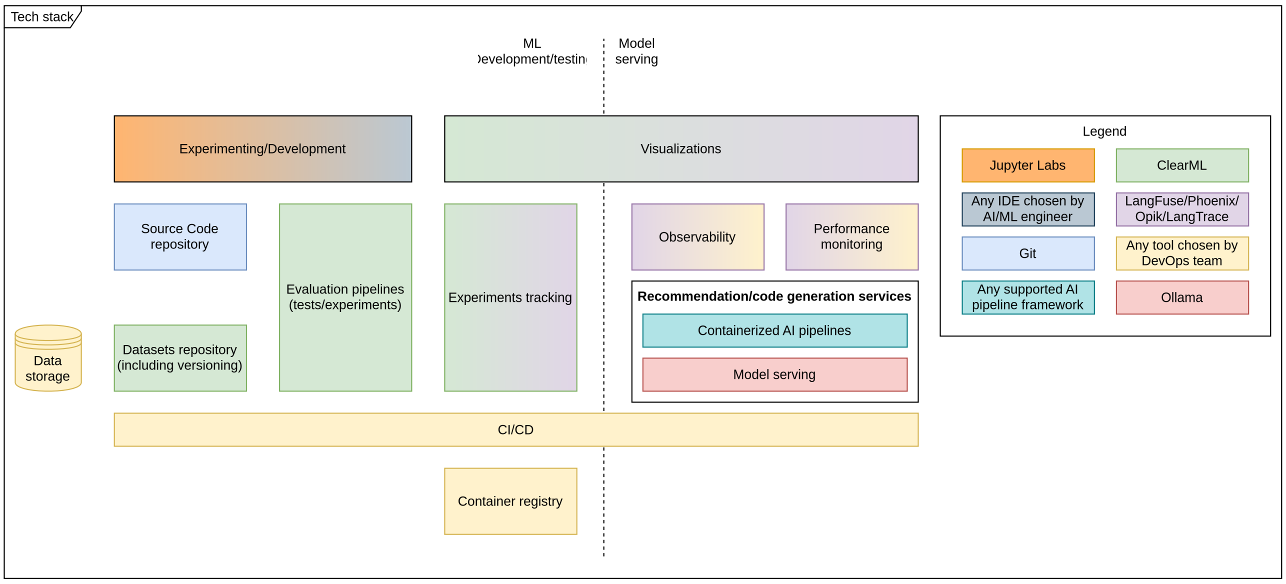
LangFuse/LangTrace/Phoenix/Opik: These belong to the family of LLM observability tools. Their main job is to track traces from AI pipelines, run evaluations on collected traces, and provide AI performance monitoring. They also have some feature overlap with ClearML, notably managing datasets and tracking experiments. As mentioned earlier, we will need to evaluate and compare them further in practice to choose one.

Ollama: Our model serving tool. We chose Ollama, at least in our early stages, for its simplicity and ease of use. We are aware of Ollama's performance limitations compared to other alternatives. However, we believe it can boost us significantly in our early stages and help us prove our concept quickly. We may switch to a more performant alternative in the future.

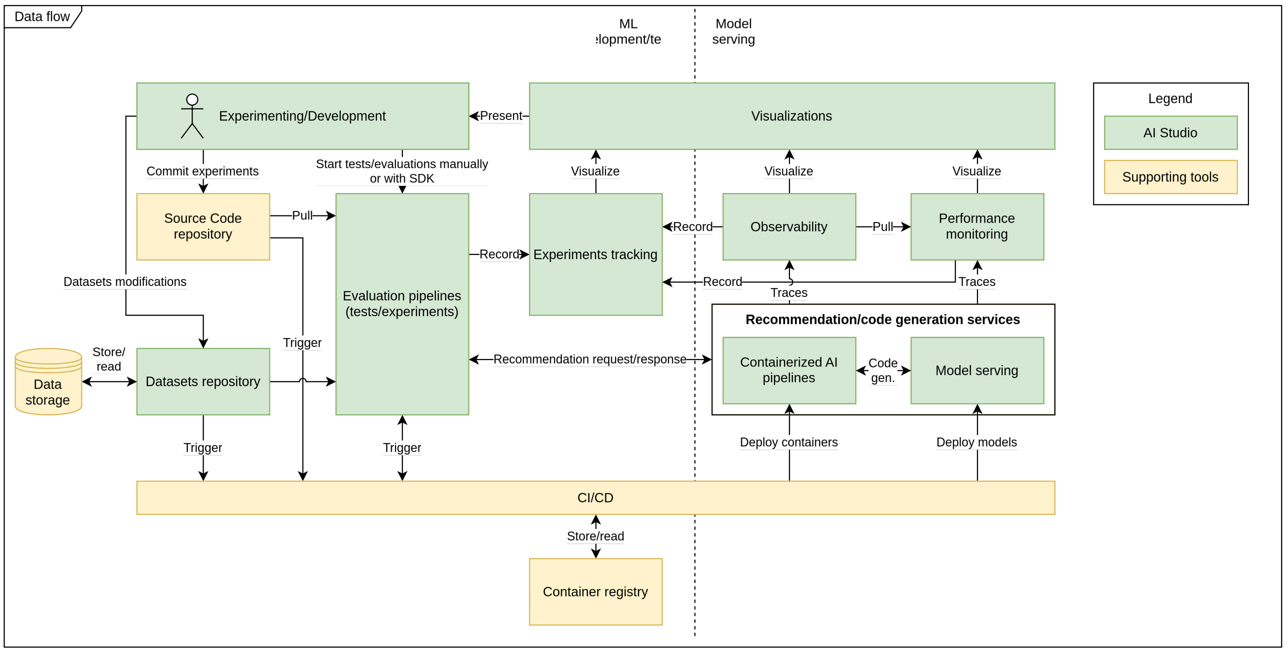
JupyterLab: The IDE used by our ML engineers for day-to-day experimentation and development. This tool is a de facto standard in the data science and ML community, making it a clear choice. However, we do not limit our engineers and developers to JupyterLab; they can use any IDE they prefer.

The above tools may be collectively considered our "AI Studio". To reach their full potential, they need to be integrated into the broader ecosystem of other DevOps tools. They will have close integration with CI/CD, Git, and potentially many others.

Below is an illustrative diagram showing how the selected tools map to the MLOps aspects we want to provide.


Tech stack
ML Development/testing
Model serving
Experimenting/Development
Visualizations
Source Code repository
Evaluation pipelines (tests/experiments)
Experiments tracking
Observability
Performance monitoring
Recommendation/code generation services
Containerized AI pipelines
Model serving
Data storage
Datasets repository (including versioning)
CI/CD
Container registry
Legend
Jupyter Labs
ClearML
Any IDE chosen by AI/ML engineer
LangFuse/Phoenix/Opik/LangTrace
Git
Any tool chosen by DevOps team
Any supported AI pipeline framework
Ollama

An example of the data flow between various components of the stack is illustrated in the diagram below.



4.11.2 Next Steps

We have identified promising contenders for our AI Studio, but we need to verify their practical integration.

To test the AI Studio in practice, we will:

1. Deploy AI Studio tools (or core parts) to our testing Kubernetes cluster.
2. Integrate our simple demo "application" we created with AI Studio tooling. The demo application mimics the architecture and design of the connector code generator.
3. Rewrite some existing experiments about mappings and correlations to integrate with AI Studio tooling.

This will allow us to test various AI Studio aspects, including:

- Writing, running, and managing experiments.
- Experiment tracking.
- Creating evaluation pipelines.
- Pipeline orchestration.
- Observability.
- Deployment flows.
- JupyterLab integration.

4.11.3 Risk Factors

The current main risk factors are:

- **Experiment evaluations**: Uncertainty about the optimal integration flow between our tools, particularly getting necessary execution data to the evaluation pipeline. We have several approaches/theories to investigate and test.
- **Distributed traces**: The project's architecture will consist of several microservices, including "regular" microservices and AI pipelines utilizing AI tooling (LLMs or others). While AI Studio tooling will collect traces from AI pipelines, we want end-to-end visibility of user interactions, correlating traces from all microservices and presenting them in a unified view. How to implement this remains an open question, potentially requiring another tool.
- **Deployment flows**: We have a theoretical deployment approach, but we need to test it practically. It will likely combine a testing tool (e.g., ClearML) with CI/CD tooling, but the implementation details are undecided.

4.12 Security Considerations

Security is a top concern of this project. In this section we concentrate on a crucial aspect: **How can we be sure that the generated and/or shared code is safe to execute?** By "the code" we mean code in connectors and in mappings. (It is possible that the scope could be extended a little bit if we start generating e.g. delineation conditions.)

We plan to undertake the following:

1. **Checking the generated code.** The code generated by LLMs will be checked for correctness and safety. This is the responsibility of the Validation Service, described below.
2. **Checking the shared code.** Also, the code shared via the Integration Catalog will be checked for correctness and safety. This is the responsibility of the Validation Service as well.
3. **Reducing the power of generated or shared code.** As mentioned in section 4.5.5 above, Groovy is extremely powerful language. We can lower the risk by allowing to use more restricted, less powerful, and hence safer language.
4. **Limiting the amount of generated code.** By moving some of the functionality into the standard midPoint or connector code, the amount of application-specific code being generated or shared decreases. For example, the base connector framework (section 3.1.7) provides many low-level (technical) features necessary for connect to remote applications, reducing or sometimes even eliminating the need to write custom connector code. Or, planned support for mapping complex attributes (section 4.5.1) eliminates or at least reduces the need for creating code that manipulates embedded data structures, using declarative configuration style instead.

   As there is less code to be generated or shared, it is easier to inspect this code (either automatically or manually) to ensure its safety. Moreover, reducing the responsibilities of custom code may mean that a more restricted, safer language can be used instead of Groovy.

4.13 Changes to MidPoint

This section describes changes to midPoint that have to be done in order to fulfill goals of this project.

4.13.1 Complex Attributes

The original design of midPoint and ConnId connector framework has been limited to *simple attributes* so far. This means that objects transferred between a connector and midPoint consist of attributes that are of simple types only: string (potentially with language variants), number, date and time, byte array, and object reference. Internal processing in midPoint is also limited to simple data types. (A notable exception is processing of *associations* that are based on object references. But they contain only simple data types as well.)

However, the initial analysis revealed that many of the applications that we plan to be able to connect work with *complex (structured) attributes*. Even the SCIM2 standard contains them. For example, `name` complex attribute consists of `givenName`, `middleName`, `familyName`, `honorificPrefix`, `honorificSuffix`, and `formatted` sub-attributes. Also, `emails`, `phoneNumbers`, `addresses` are multi-valued complex attributes, typically containing `type`, `value`, and `primary` sub-attributes. Besides SCIM2, we know of real deployments where complex attributes are present.

The current version of midPoint allows treating complex attributes only by funneling them through string attributes by applying some kind of encoding: JSON, XML, CSV, or the like. For example, when synchronizing complex data from the resource, the connector has to pack the received data into an appropriate format (e.g., JSON) and provide it to midPoint in a string attribute. MidPoint then must unpack (parse) the JSON string and process the unpacked data accordingly.

This works but is unnecessarily complex.

It is preferable to enhance midPoint so that it will handle complex attributes right out of the box in order to prevent overburdening the AI generator by forcing it to provide sophisticated workaround code.

These changes will require significant modifications of the panels in the GUI as well.

4.13.2 Schema Enhancements

With SCIM2 becoming more prevalent, we expect a lot of cases of matching standard SCIM2 schema to midPoint. For example, a multivalued complex `emails` SCIM2 attribute to a single-valued, simple, `emailAddress` midPoint property. The `phones` vs `telephoneNumber` is the same case. SCIM2 `address` attribute has no counterpart in midPoint at all.

Today’s real deployments need more. Many of them put secondary email addresses, secondary telephone numbers, and postal addresses into custom extension properties. Each of them is implementing the same or similar functionality in different ways, wasting time and energy on something that could be provided out of the box.

From the perspective of this project, the AI schema matching generator will have to find matches between SCIM2 schema and these custom schema extensions. It will be able to do that. But it’s an unnecessary complexity.

Therefore, we will enhance midPoint schema with pieces from standard SCIM2 schema, so that mappings will be much more straightforward, providing efficiency and consistency of generated solutions.

4.13.3 Natural (Business) Keys in Item Paths

MidPoint uses an item path as a concept to identify a concrete item in structured objects. The item path is used in various roles, such as specification of source and target attributes in mappings, specification of attributes in queries, deltas and other places. The current item path uses numeric identifiers for multivalue properties, which makes it hard to read, configure and understand. With the introduction of complex attributes in ConnId and midPoint schema, there is increasing need to introduce support for natural (business) keys in item paths (eg. email type, phone type, etc.). As the preliminary analysis suggests, we will probably need even more: arbitrary filters and potentially also queries in item paths.

This change will allow to use more human-readable item paths, which will be easier to understand and use with complex attributes.

The change affects multiple places in midPoint, including the item path parser, item path serializer, item path validation, and item path support in various components processing queries, deltas, and mappings.

4.13.4 Support for Advanced Aggregate and Analytical Queries

MidPoint currently supports basic queries, such as filtering objects by attributes, searching for objects by name, attribute values, and other criteria. The use-cases in Smart Integration (mappings, delineations) require more advanced query support for statistical analysis of distribution of attribute values, duplicates.

The proposed change is to enhance midPoint Aggregate Query API and PostgreSQL Native Repository to support advanced query capabilities, such as grouping, counting, averaging by native attributes and extension attributes.

4.13.5 Safer Scripting Language

Groovy (or similar) scripts play a pivotal role in midPoint deployments. All non-trivial mappings use them, including mappings generated by this project. Custom reports and GUI customizations are often based on them. They are present in ScriptedSQL connectors and soon they will be in connectors generated by this project.

These scripts are extremely powerful. They can execute all operations available to the operating system (OS) user under which midPoint is executing. This includes arbitrary modifications of midPoint repository content (including roles and authorizations), modifying files in local filesystem, executing OS commands, opening network connections, and so on. Although midPoint has Expression Profiles that are designed to limit the possibilities of what scripts can do, they are not bullet-proof. They are not easy to configure, and there are ways to circumvent them.

Normally, the majority of scripts are created by system administrator, so there is no problem in their power. But this project introduces at least two other sources of these scripts: they will be generated by LLMs and also provided by third parties in the Integration catalog. Although this project will provide legal and technical mechanisms to prevent malicious code execution, we plan to provide even better option: using a safer scripting language.

The chosen language will be restricted enough to be relatively safe to execute even if the source of the scripts is not absolutely trustworthy. Of course, planned mechanisms to prevent malicious code from being run will be still applied - to avoid even less harmful effects, like denial of service or information leakage.

Regardless of whether such a language will be found and used, we will continue to support Groovy, with all the planned safety mechanisms.

4.14 Changes to ConnId Framework

The connector framework needs some enhancements to provide the functionality necessary for rapid application onboarding.

- We need to pass more schema information from a resource to midPoint. For example, human-readable descriptions of object classes and their attributes have been missing up to now. They are crucial for AI algorithms for schema matching and mapping, besides others.
- MidPoint now needs more efficient access to the schema. In particular, midPoint must be able to iteratively query the schema, obtaining a list of object classes first, in order to provide AI algorithms a possibility to select which object classes are relevant for further processing and which are not. (Fetching all the data at once may not be possible because of system resource constraints.)
- We need to support complex configuration items in connector configuration. Up to now, only simple properties (e.g. of string, integer, boolean and similar types) have been supported. The analysis carried out up to now indicates that we need also complex structures to be passed to connectors as their configuration.
- In order to support complex attributes in midPoint, some changes need to be done in ConnId as well. For example, UID and Name attributes should become optional for embedded object

classes.

- The analysis of existing systems revealed that UID and Name attributes should support data types other than string. Currently, they are limited to string values; if a resource requires other values (e.g., integers), they must be converted from/to strings. This is more workaround than a serious solution. When connectors are to be generated by AI, we want to eliminate the need for such workarounds.

4.15 Changes to MidPoint Studio

The current version of midPoint Studio relies on the midPoint XSD schema for validation, code completion, and other features. We identified that the support using XSD schema is not sufficient for the integration with the AI-assisted workflows.

The MidPoint Studio integration needs to be enhanced to support native MidPoint schema format (Prism) which provides additional metadata necessary to correctly identify suggestions, validations, and metadata necessary for AI-assisted workflows.

The switch from XSD schema to Prism schema will also allow for native support of additional languages such as JSON and YAML for configuration objects.

5. DevOps & Operational Standards

5.1 Hardware and Infrastructure

The overall architecture of the solution is designed as a set of services that may scale independently in the future. At the same time, we want to run these services on our hardware to ensure we have compliance and security fully under our control. We decided to deploy the solution on Kubernetes[1] for a number of reasons:

- Kubernetes is a well-established container orchestrator.
- We want to containerize the services to streamline our CI/CD pipelines, using containers as a method of delivery.
- We already have some experience with Kubernetes and want to extend it further.

Currently, there are no identified cases where Kubernetes and containers will not be sufficient. But to cover all the bases, we will also have virtual machine capability available.

Finally, as a storage solution, we will use Ceph[2]. It is a battle-proven open-source solution for software-defined storage which is more economical than hardware-based storage.

5.1.1 Hardware Design

Our current internal infrastructure is mostly composed of dated hardware placed in spaces repurposed as a server room. That's why we plan to procure new hardware and place it in a professional data center to meet all compliance requirements. Our main hardware infrastructure will be placed on the primary site and will consist of:

- Three servers, one of which is equipped with GPUs.
- Necessary networking such as switches, etc.
- Firewall appliance providing the first level of protection against common threats such as DoS/DDoS, with IDS/IPS support.

The three servers will serve multiple duties:

- All servers will be part of the Ceph cluster (storage).
- All servers will be part of virtual machine cluster - for this Proxmox[3] solution is considered.
- On top of the VMs, Kubernetes cluster will be built.

We will build 3-node cluster, and we will ensure that each node runs on a different physical node. Building the Kubernetes cluster on top of virtual machines gives us a lot of flexibility. For instance, we can have a dedicated test cluster completely separated from the production one, or we can experiment with Kubernetes clusters of more than just three nodes.

For this setup, three physical nodes are the minimum if we require reliability and proper operation of both the Kubernetes and Ceph cluster. Because of the multiple roles each server node fulfills, the nodes should be reasonably sized, ideally dual-processor with plenty of RAM and disk space.

The secondary site, geographically dislocated (at least a few kilometers apart), consists of a single backup server that also provides Zabbix monitoring functionality.

5.1.2 GPU Requirements

While some low-key development can rely on local GPU/APUs, for any serious development, testing, and future production high performance GPUs are must. Because we want to host our hardware in a data center, data-center-grade GPUs are required - this means passive cooling solution allowing high-density computing. This may limit our options both in terms of availability, because of the current AI hype, and in terms of cost as well, because data center GPUs are more expensive.

At the start of the project, it wasn't clear what will be the ratio of the training and inference workloads. At this moment, it is obvious that we will need much more inference than training. Occasional bursts of training can also be served by external services if required, but inference workloads will be our bread and butter.

Current design uses LLMs for Smart Integration (expected 2 models with 32 billion parameters, or 32B), Code Generator (at least 2 instances of 32B parameter LLM), and Code Validator (1x 32B LLM). All these models must run at the same time and be ready to serve requests, that is, they must be warmed up. This all points to the total VRAM requirements of 200 GB or more.

For flexibility, it would be also ideal to have multiple cards instead of one giant one. This would allow us to isolate different kinds of workloads, and run the to-be-production services simultaneously with development tasks. This would suggest at least two GPUs, but three would be better as it would allow us to use two for the solution with one for additional tasks. Naturally, at the beginning of the development cycle, all GPUs would be used for development, testing, and data preparation.

To summarize GPU requirements, we will need: Three GPUs with at least 80 GB of VRAM each, and with performance in the range of 100 TFLOPS for FP32 and 2000 TOPS for INT8, or more.

5.1.3 Hardware Origin and Risk Assessment

To ensure cybersecurity and compliance requirements of our solution we will choose hardware and supplier that adheres to good practices in ICT supply chain cybersecurity as outlined by ENISA, e.g. in Good Practices for Supply Chain Cybersecurity, June 2023[4]. While formal certification to standards such as ISO/IEC 20243 or IEC 62443 is not required, alignment with their principles - including supply chain transparency, origin control, and secure delivery - is expected.

5.2 Kubernetes and Network Infrastructure

Before we get to the CI/CD pipeline for the final solution, we will need to bootstrap the Kubernetes environment and supporting services.

5.2.1 Kubernetes Cluster and Supporting Services

The Kubernetes cluster is planned to be built in the following steps:

1. **Virtual machine creation**
   3 for control plane nodes and 3 for worker nodes. Each physical node will run one control plane and one worker node. The sizing of the worker nodes will be generous, details depending on the delivered hardware.
2. **Operating system**
   Ubuntu 24.04 LTS will be used as the operating system, as it is our preferred OS and widely supported by Kubernetes distributions and installation tools, such as kubespray[5] or RKE2[6].
3. **Kubernetes Distribution and Installation**
   RKE2 is planned as the Kubernetes distribution, with a minimal cluster installation performed initially. We will use the default CIDR settings for service and pod networking (see Network Segmentation below), but override the `node-cidr-mask-size` from the default 24 to 22 to allow for more pods per node. Following capabilities must be configured before the first workload that depends on persistent storage and networking:
   - Storage using the Ceph cluster.
   - Networking using Cilium with eBPF enabled for optimal performance. No tunneling is needed because all the nodes are on the same layer 2 domain. WireGuard encryption will be used for inter-node communication.
4. **Argo CD Deployment**
   Argo CD[7] will be installed as the first optional component. This will be used to manage not only the applications later, but also all additions and Kubernetes plugins from this moment further.
5. **GitOps Setup**
   Argo CD will be connected to an in-advance prepared Git repository with all the other infrastructure components, such as cert-manager, ingress, proxy, DNS, load balancer, network policies, etc.

Supporting services are described later in the section 5.4 Operational Architecture.

5.2.2 Network Segmentation

The following table shows the proposed subnets and VLANs for our primary site:

VLAN ID	Subnet	Description
10	10.10.10.0/24	**Out-of-Band Management**, for IPMI, isolated from other networks.
20	10.10.20.0/24	**Provisioning / Admin**, used to SSH into Proxmox, hypervisors, and base OS.
30	10.10.30.0/24	**VM Internal Network**, internal VM traffic, isolated from WAN.
35	10.10.35.0/24	**Proxmox Sync**, ideally isolated, dedicated network used only for Corosync traffic.
40	10.10.40.0/24	**Kubernetes Nodes**, used by K8s control/worker nodes for inter-node traffic.
50	10.10.50.0/24	**Ingress / Load Balancer**, for public-facing services accessible via ingress or LB.
80	10.10.80.0/24	**Storage (Ceph)**, dedicated Ceph traffic.
90	10.10.90.0/24	**Monitoring/Logging**, isolate telemetry traffic (this is optional, and may be dropped).
N/A	10.42.0.0/16	**Kubernetes Pods**, allocated via cluster CIDR (used by CNI); used for internal pod-to-pod communication across the cluster; not externally routable.
N/A	10.43.0.0/16	**Kubernetes Services**, allocated via cluster CIDR (used by CNI); not externally routable.

From the workload deployment perspective, Kubernetes networks are used most heavily, but they are largely abstracted away through the use of service names. Additionally, the Ingress network is used to expose software services to clients outside the cluster. Access from the public Internet is handled by the edge firewall, which routes traffic to the Ingress network.

The backup/Zabbix server on the secondary site will have its own uplink with a separate public IP. It will not be part of any VPN/VLAN, it will be heavily firewalled, and will be accessed over secure protocols (SSH, HTTPS).

5.3 CI/CD Pipeline

Continuous Integration (CI) and **Continuous Delivery** (CD) are two related terms from a family of "continuous" terms. This is underlined by the fact that CD can also mean **Continuous Deployment** which in some context is clearer as a complement to CI. All these terms imply various levels of automation - the higher, the better, but this also requires higher maturity level of the organization.

It is important to focus on the "value stream" - what is going in (e.g. source code), how is it transformed, what are the key artefacts, and where they should go. The pipeline can be roughly split into the CI-part which, in simplified terms, corresponds to the "build" phase - and CD-part, here with focus on deployment. CI part is mostly done with dev/build tools, e.g. Maven, or, in Kubernetes environment, tools for building container images. The result is an artifact that is either consumed by other CI/CD pipelines (in case of a library) or ultimately deployed. A deployable artifact can be deployed to multiple environments, including testing environments for end-to-end and other kinds of tests.

5.3.1 CI/CD Requirements

These are the identified requirements for our CI/CD pipeline:

- Building midPoint (Java, Maven), this is an existing capability, currently we use Jenkins for this.
- Building container images - all the services and AI workloads will be delivered as container images.
- Deployment to Kubernetes, both automatic and with manual intervention, including selection of the target environment. This may be handled directly by the CI/CD pipeline or delegated to GitOps tools such as Argo CD.
- The pipeline should scan artifacts for dependency vulnerabilities using Dependency-Track[8], which is already established in our company. This means, CI/CD must be able to send Software Bill of Materials (SBOM) to Dependency-Track.

At this stage, we don’t expect sophisticated pipelines incorporating AI workflows, as we mostly use existing LLMs for inference. Also, AI development and testing initially require more manual intervention. This may change in the future and tools such as MLflow or kubeflow may be used by the primary CI/CD pipeline.

The following types of artifacts must be supported by our CI/CD processes:

- Existing Java artifacts, namely midPoint. It is normally run on customers' infrastructure, but for testing purposes we will also deploy it to our Kubernetes cluster.
- Our services, packaged as containers and deployed as pods to Kubernetes. Helm or Kustomize are both acceptable to describe the deployment as long as the environment-specific configuration is clearly externalized.
- Third-party software for development (such as various AI tools), and operations (monitoring, logging, and other qualities). This will be managed in GitOps fashion, utilizing Helm charts or Kustomize, ideally already available from 3rd parties.

5.3.2 Artifact Registry

Beyond the main CI/CD tools for build and deployment, another component is of utmost importance - artifact registry (also called a repository). There are many types of artifact repositories, often specific to a programming language ecosystem, for instance, a Maven repository for Java.

Based on the known requirements from the development team, we will not need Python or other language-based repositories initially. But we must introduce a robust and reliable container image registry (Docker registry).

We are long-term users of Nexus OSS repository software, however, recent restrictions - specifically, a 24-hour request limit - have made it unreliable. It is a capable universal registry/repository supporting all the important artifact formats. But we decided not to use Nexus OSS because it becomes non-functional when the request limit is exceeded, and the commercial version is very costly.

The following alternatives are considered:

- JFrog Artifactory OSS[9] - a direct competitor of Nexus, universal registry, free version available.
- Harbor[10] - OCI-compliant registry, Docker images and binary files (via ORAS[11]), OIDC support; no Maven and npm support.
- GitLab also offers a built-in Docker registry. While we use GitLab already, it is debatable whether this responsibility should belong here, or whether a focused project like Harbor would be more appropriate.

5.3.3 CI/CD Solution

Let’s discuss our current CI/CD ecosystem and how it can serve the needs of this project:

- **Source repositories**
  GitHub and internal GitLab; virtually any Git repository would work the same way in our ecosystem. Both solutions support webhooks to trigger the build in CI/CD, but polling would also be an acceptable solution. Generally, we treat Git repositories purely as source code storage, not tying ourselves to the additional features, especially for GitHub, which we do not host ourselves.
- **CI/CD product**
  Jenkins with Jenkinsfile-style pipelines builds most of our software, predominantly midPoint and related libraries. With current CI/CD requirements, any generic CI/CD solution would suffice as long as it can be integrated with other tools we use to check source code quality and detect potential vulnerabilities, that is Dependency-Track and SonarQube. These tools are often integrated primarily on the build tool level. Given our familiarity with Jenkins and the fact that CI/CD is not the core focus of this project we may keep using it not to disrupt the development flow. We should improve our current scripted pipelines, which often duplicate Kubernetes setup code. We will therefore try to standardize and reuse our Kubernetes setup templates and adopt declarative pipelines where feasible. For short development- or test-

oriented pipelines, GitLab CI/CD is also under consideration.

- **Artifact repository** - Nexus, covered in previous section.
- **Kubernetes deployment**
  Argo CD; this is currently used predominantly by the infrastructure team to deploy 3rd-party products for the needs of the company. In the scope of the project, Argo CD will be used much more by the development team as well, be it directly or via another tool, to deploy the services into Kubernetes.

Other CI/CD solutions were reviewed, with a focus on OSS, free, self-hosted options. Naturally, there is significant overlap in features across tools, though some offer unique approaches. If strong pipeline modeling, e.g. as a directed acyclic graph (DAG), was required, we would consider options such as Concourse CI[12] or Argo Workflows[13]. GitLab CI/CD and GitHub Actions also support DAG-style job dependencies. However, we decided to stay with our current solutions and focus on improving pipeline standards instead of migrating to a new tool.

5.3.4 GPU Workloads

GPUs are hosted on one of the three physical nodes in the cluster. All GPUs are made available to the Kubernetes worker VM through PCIe passthrough. This worker VM runs the required GPU drivers and Kubernetes plugins (e.g. NVIDIA device plugin for Kubernetes[14]). The Kubernetes node is labeled with GPU-related tags to enable GPU-aware pod scheduling[15]. Otherwise, the AI workload is managed like any other service, apart from the need to monitor additional GPU-specific metrics.

5.4 Operational Architecture

This section describes the operational architecture supporting the system, focusing on how services are deployed, accessed, and monitored. It includes considerations for running AI components, along with the supporting infrastructure such as observability tooling and API routing. Generally, we should follow well-known practices inspired by the Twelve-Factor App[16] methodology.

5.4.1 Observability Architecture

The observability architecture is designed to provide a unified monitoring and logging solution for all workloads across the project. This approach ensures consistent visibility and operational awareness, enabling rapid incident response. Since the project will be deployed on our company hardware, there is a natural synergy with company-wide compliance and security initiatives. Rather than maintaining a separate observability platform, company observability solution will be redesigned to meet the needs of the project.

Key design aspects include:

- **Shared company-wide monitoring and logging stack:**

A centralized observability platform will be deployed using industry-standard tools such as Prometheus[17] for metrics collection, Grafana[18] for visualization, and an OpenSearch[19] -based logging platform for log aggregation and analysis. OpenSearch replaces the well-known ELK stack (Elasticsearch, Logstash, Kibana), which we want to avoid due to their less OSS-friendly licensing. This stack will be co-deployed on the company Kubernetes infrastructure and shared across projects for operational consistency.

- **Namespace-aware data collection and logical isolation:**
  The monitoring infrastructure will support separation of metrics and logs by Kubernetes namespace, enabling clear operational boundaries between projects or teams. This approach simplifies access control while retaining a unified observability backend for the whole organization. The setup remains compatible with stricter isolation models if future workloads or compliance requirements demand them.
- **Developer empowerment and collaboration:**
  The infrastructure team will maintain the core monitoring and logging platform. Developers will be empowered to create and manage their own dashboards and alerts within their namespaces. This enables agile and responsive operational insights for each team or project without compromising centralized governance.
- **Integration with CI/CD and security tooling:**
  Observability will extend to monitor CI/CD processes (covered in section 5.3), including build health, artifact vulnerability scans, and SBOM generation. Security-related telemetry and scan results will feed into the monitoring platform to provide feedback for the developers, as well as for the security team.
- **Alerting and incident escalation:**
  Alerts will be defined based on service-level objectives and operational thresholds, with integration to incident management workflows. In addition to Prometheus Alertmanager[20], the Zabbix[21] server deployed at the secondary site will provide complementary monitoring and escalation capabilities for infrastructure-level events. The coordination between these systems will be defined to avoid alert duplication and to leverage long-term monitoring strengths of Zabbix.

Deploying all these solutions is just a start and can quickly become a burden if not managed with high level of automation and GitOps practices. Once the observability platform is well established, the proposed tools provide a solid foundation for integrating Wazuh[22], an open-source SIEM (Security Information and Event Management) system. Wazuh leverages components like OpenSearch, allowing us to build on existing knowledge and tools without starting from scratch. This would offer a more comprehensive, holistic view of our collected data and can extend to integrate with our firewall for enhanced security monitoring.

5.4.2 Workload Organization and Namespace Strategy

Kubernetes namespaces offer a way to logically separate and manage workloads within the same cluster. One option is to create a dedicated namespace for each service or functional component, e.g. Code generator, Integration catalog, etc. This approach allows for fine-grained access control, isolated resource quotas, and clearer separation of concerns. This is especially beneficial in large-scale systems or when multiple teams are involved. It can also simplify observability, network policies, and operational boundaries, but introduces additional overhead in terms of configuration and coordination.

In contrast, placing all project workloads into a single shared namespace reduces complexity at the beginning. This makes it easier to configure networking, access control, and CI/CD pipelines during early development and prototyping. It also reflects the current reality of a small, co-developed system where most components are tightly integrated and maintained by the same team (or closely related teams with a high level of trust).

We will start with a single shared namespace for all workloads to keep the setup lean and manageable. As the architecture matures and services evolve, we may introduce namespace boundaries where operational or security requirements justify them.

5.4.3 API Gateway and Ingress Management

API Gateway has the following responsibilities:

- Act as the primary boundary between internal services and external clients, hiding internal service details and protecting the cluster from direct external access.
- Manage incoming traffic with routing, TLS termination, and basic authentication.
- Provide advanced API management features such as authentication, rate limiting, etc.
- Monitor key metrics like request rates, latency, and errors to identify and resolve issues proactively.
- Trigger alerts on anomalies (e.g., codegen timeouts) to minimize user impact and maintain service reliability.

The initial plan is to use the NGINX Ingress Controller[23] to manage incoming traffic, providing essential functions such as routing, TLS termination, and basic authentication. This setup ensures reliable and secure access to services hosted within the Kubernetes cluster.

Looking ahead, we plan to evaluate alternative API gateway solutions such as Kong[24] or Traefik[25]. These platforms offer advanced capabilities beyond basic ingress, including API management features like authentication and rate limiting. Selecting the optimal gateway will be driven by evolving project requirements and ease of integration.

We will monitor gateway metrics like request rates, latencies, and error counts to detect bottlenecks or failures early. For example, if the codegen service starts timing out, alerts can trigger investigations before users notice impact. This proactive monitoring helps keep APIs stable and performant throughout development and production.

5.4.4 Workload Standards and Best Practices

To ensure consistent quality, security, and maintainability across project workloads, a set of standards and best practices will be established and followed.

- **Container Images:**
  All container images must be based on vetted, minimal base images to reduce attack surface and improve startup times. Images will undergo security scanning as part of the CI/CD pipeline, with Software Bill of Materials (SBOM) generated for traceability and compliance (orchestrated by CI/CD pipeline).
- **Observability Instrumentation:**
  Services must expose standardized endpoints to facilitate monitoring and health checks:

  - `/metrics` endpoint conforming to the Prometheus exposition format[26] for metrics collection. This is a de facto standard supported by many client libraries.
  - Kubernetes health (`/health`) and readiness (`/ready`) probes to enable proper lifecycle management by the platform.

  Examples for backend languages commonly used in the project:

  - **Java (Spring Boot):** Use the Micrometer[27] library, which integrates seamlessly with Spring Boot Actuator to expose Prometheus-compatible metrics. Spring Boot Actuator also supports built-in health and readiness endpoints out of the box.
  - **Python:** Recommended libraries include prometheus-client[28] for metrics exposure, along with frameworks like FastAPI[29] or Flask[30] combined with health check packages (e.g., py-healthcheck[31]) for readiness and liveness probes.

  Using these libraries will help standardize telemetry while minimizing custom implementation, ensuring consistency across services. The final choice of libraries for observability instrumentation may evolve over time.

- **Logging and Log Forwarding:**
  Logs should follow a consistent, structured format and include key metadata such as timestamps, service name, and request or correlation IDs. This consistency enables effective troubleshooting and traceability across distributed services. Guidelines will be established to ensure logs are forwarded reliably and securely to the centralized observability platform based on OpenSearch.

  Although we do not plan to use OpenTelemetry[32] for the project at this moment, we may consider it as our observability solution matures. OpenTelemetry, as a CNCF-hosted project, provides widely accepted standard formats for telemetry data including traces, metrics, and logs.

  Candidate libraries for structured logging include Logstash Logback Encoder[33] for Java (often used with Spring Boot) and structlog[34] or python-json-logger[35] for Python.

As typical in Kubernetes or cloud environments, logs should be written to standard output (stdout).

- **Configuration and Lifecycle Management:**
  Workloads will be configured through environment variables and externalized configs, as this improves portability and eases operational management. Graceful shutdown mechanisms must be implemented to allow pods to terminate cleanly, releasing resources and avoiding request disruptions.
- **Resource Management:**
  Clear guidelines for specifying resource requests and limits (CPU, memory) will be enforced to ensure stable cluster operation and prevent resource contention. This includes recommendations for initial values and adjustments based on observed usage, enabling efficient autoscaling and scheduling.

Together, these standards form a foundation for robust, secure, and observable workloads that align with the overall project and company infrastructure goals.

5.5 Security Operations

Security is an integral part of the operational lifecycle and is applied consistently across the infrastructure, CI/CD pipelines, and deployed workloads.

The following practices and tools are considered as our security operations baseline:

- **Access control** is enforced using Kubernetes RBAC, with further restrictions applied through GitOps workflows (e.g., Argo CD) to ensure auditable and declarative changes. Authentication is integrated with our Keycloak instance, enabling centralized identity and access management.
- **Secrets management** is implemented using Kubernetes Secrets initially, but we consider to advance to HashiCorp Vault[36].
- **Container security** is addressed via:
  - Minimal base images.
  - Regular vulnerability scans of built artifacts using Dependency-Track[37].
  - Generation of SBOMs using tools like Trivy or Syft, and tracking of image provenance and metadata in CI/CD.
- **Infrastructure hardening** includes baseline OS configuration, restricted administrative access, and restricted network ingress.
- **Auditability** is achieved through logging of CI/CD actions, Kubernetes events, and changes to GitOps-managed infrastructure.

Security practices will evolve with the project, depending on the scope of exposed APIs, or regulatory demands. An internal security review will be part of the final staging/deployment process for any public-facing services.

Because the Kubernetes cluster and workloads are fully under our control, excessive security measures such as mTLS are not required. Pods can use HTTP within the Kubernetes pod network. Cilium network policies will restrict communication between unrelated pods. Communication between Kubernetes nodes will be encrypted by Cilium to prevent sniffing on the VM network. Outside the cluster, all communication will use HTTPS, with Let's Encrypt serving as a sole certificate authority.

5.6 Backup and Disaster Recovery

Our infrastructure incorporates a multi-layered backup and disaster recovery (DR) strategy, designed to minimize data loss and service downtime.

The redundancy is provided on these levels:

- **Ceph-based storage** with replication across nodes provides immediate fault tolerance.
- **Secondary site** with a dedicated backup server, this ensures geographic redundancy and protection from catastrophic failures.
- An additional secondary backup may be introduced if needed; however, this requires data encryption at rest and can increase risk if not carefully managed.

We are familiar with Backy2[38] for Ceph RBD image backups and BorgBackup[39] for structured, file-level or application-aware backups. This combination creates a strategy in which important data are backed up in two complementary forms, each with its pros and cons.

- Backing up whole images (or volumes) shortens recovery times, but the backup process is unaware of application state, which may result in inconsistencies.
- Recovering business data, e.g. database dumps and selected directories on the filesystem, is more involved. The system has to be reinstalled first (and reconfigured if the configuration is not backed up) and then filled with the data from backup.

Since our future infrastructure will use Proxmox for VM clustering, it is natural to utilize the Proxmox Backup Server (PBS)[40] solution for VM-level backups. PBS can effectively replace Backy2-style backups, which are application-unaware, but PBS also supports "quiescence" via QEMU guest agent, enabling more consistent backups.

In any case, this kind of backup consumes more disk space, but at the same time enables faster recovery. Naturally, occasional full backups will be combined with incremental backups to save some space. But if anything goes wrong with this kind of backup, e.g. inconsistent image, one deals with a binary blob that may be useless in the worst case, or very difficult to put together at least.

We plan to use Kubernetes-oriented backup tools such as Velero or Stash for backing up cluster resources and persistent volume data.

Application-aware backups remain valuable even when wholesale backups of underlying data are performed. At this moment, we intend to continue using Borg for this type of backup. These backups are typically more space-efficient and suitable for long-term retention.

Backups will be **scheduled and verified regularly**, with attention to recovery time objectives (RTO) and recovery point objectives (RPO) appropriate to each workload type. This setup is a solid foundation, but formal DR testing and procedures will be defined in later operational stages.

[1] https://kubernetes.io/
[2] https://ceph.io/
[3] https://www.proxmox.com
[4] https://www.enisa.europa.eu/publications/good-practices-for-supply-chain-cybersecurity
[5] https://github.com/kubernetes-sigs/kubespray
[6] https://docs.rke2.io/
[7] https://argo-cd.readthedocs.io/en/stable/
[8] https://dependencytrack.org/
[9] https://jfrog.com/community/download-artifactory-oss/
[10] https://goharbor.io/
[11] https://oras.land/
[12] https://concourse-ci.org/
[13] https://argoproj.github.io/workflows/
[14] https://github.com/NVIDIA/k8s-device-plugin
[15] https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/
[16] https://12factor.net/
[17] https://prometheus.io/
[18] https://grafana.com/
[19] https://opensearch.org/
[20] https://prometheus.io/docs/alerting/latest/alertmanager/
[21] https://www.zabbix.com/
[22] https://wazuh.com/
[23] https://github.com/kubernetes/ingress-nginx
[24] https://konghq.com/
[25] https://traefik.io/traefik/
[26] https://prometheus.io/docs/instrumenting/exposition_formats/
[27] https://micrometer.io/
[28] https://github.com/prometheus/client_python
[29] https://fastapi.tiangolo.com/
[30] https://flask.palletsprojects.com/
[31] https://pypi.org/project/py-healthcheck/
[32] https://opentelemetry.io/
[33] https://github.com/logstash/logstash-logback-encoder
[34] https://www.structlog.org/
[35] https://pypi.org/project/python-json-logger/
[36] https://developer.hashicorp.com/vault

[37] https://dependencytrack.org/

[38] https://github.com/wamdam/backy2

[39] https://www.borgbackup.org/

[40] https://pbs.proxmox.com/wiki/index.php/Main_Page

6. Compliance & Governance

Our organization is in the process of establishing an information security management system aligned with ISO/IEC 27001. Preparations for certification are underway, with the goal of completing the process within the next few months. This ensures that security and compliance are treated as ongoing effort, not just one-time activities.

The following sections focus on governance and regulatory aspects specific to AI systems, including standards like ISO/IEC 42001 and legal obligations under the AI Act and GDPR.

6.1 AI Governance and ISO/IEC 42001

ISO/IEC 42001 is the international standard for Artificial Intelligence Management Systems (AIMS), designed to help organizations manage risks and responsibilities associated with AI technologies. We do not seek this certification, so it is not mandatory for us, but its principles offer a structured way to align AI development and deployment with safety, transparency, and accountability goals.

As we want to improve our AI governance practices, we will use ISO/IEC 42001 as a guiding framework. Some of the relevant areas it covers, and how it relates to our project, include:

- **AI-specific risk management:**
  Risks associated with AI-based suggestions, such as hallucinated connector configurations or misapplied access policies, are being assessed and tracked. Manual approval steps are enforced in sensitive workflows, for instance, code must be reviewed by the user. Prompt injection vectors are included in threat modeling activities.
- **Human oversight and accountability:**
  Accountability for AI-assisted decisions is distributed across designated roles (e.g., platform, data, and security teams). Logs include traceable metadata and auditability for actions influenced by AI. Human approval is required for all changes with security or infrastructure impact.
- **Transparency and explainability:**
  Users interacting with AI-generated suggestions are shown contextual information, including explanatory tooltips and links to source prompts or documentation. Internally, all AI-related assets (prompt templates, model parameters, intended usage) are documented in version-controlled repositories.
- **Data quality and integrity:**
  Currently, it is not clear whether we will perform training tasks, but if so, training or reference data used by internal AI components must be curated from valid, up-to-date configuration sources. Inference components must also rely on validated and current configuration sources to ensure meaningful output.
- **Monitoring and impact assessment:**

AI-related features are instrumented with custom telemetry such as suggestion adoption rates and user feedback indicators. Logs about rejected or modified AI outputs support regular audits and downstream analysis of potential misalignment with user expectations.

- **Lifecycle management:**
  AI model usage is tracked across services, with support for prompt versioning, staged rollout, and rollback procedures (this is related to Transparency and Data quality controls above). All deployments involving model updates are reviewed and tested as part of normal CI/CD processes.

Where relevant, we will review the controls from ISO/IEC 42001 and prepare a related statement of applicability.

6.2 EU AI Act Alignment and Obligations

The EU Artificial Intelligence Act introduces binding regulatory requirements for the development and deployment of AI systems within the European Union. In contrast to voluntary frameworks such as ISO/IEC 42001, AI Act establishes legal obligations based on the risk category of each AI system. As an EU-based organization, we are subject to this regulation with compliance obligations expected to phase in between late 2025 and 2026.

Our current use of AI focuses on assisting identity engineers (which are users from the API's perspective), specifically generating code snippets and suggesting attribute mappings in the context of midPoint connector development and configuration. These suggestions are subject to human review and approval, and they do not autonomously affect end-users or identity data. Nevertheless, we aim to align our practices with the Act's core principles.

Key considerations include:

- **General-purpose AI usage:**
  If using open-source or third-party LLMs, we must ensure they originate from providers that offer sufficient transparency and documentation. This includes licensing, model information, and technical details to meet applicable legal and ethical requirements, especially as defined by the EU AI Act and associated Codes of Practice.
- **Transparency and documentation:**
  Any internal models or prompt templates must be documented, including their intended use, limitations, and version history. This helps satisfy the Act's expectations for general-purpose AI traceability.
- **Human oversight and approval:**
  All AI-assisted suggestions (code changes, access control mappings) remain subject to human review. No AI-generated content is applied without user confirmation, reducing regulatory exposure.
- **CI/CD governance:** Outputs from AI systems are traceable through logging and versioning. Manual approval gates and automated validation steps in CI pipelines help prevent unsafe

or non-compliant code from reaching production.

- **Avoiding high-risk classification:** We monitor the evolving definition of high-risk AI under the Act. Should our AI tooling ever influence decisions in regulated domains (e.g., critical infrastructure, biometric ID systems), a formal risk assessment and compliance process will be initiated.

We want to ensure alignment with the AI Act's requirements without sacrificing developer productivity and control. To achieve this, we will implement transparency and human oversight (both for developers and users, as relevant) into our processes.

6.3 GDPR Considerations

Our project is designed to minimize the processing of personal data. Typically, all user data remains within the customer's environment and is processed locally by midPoint, consistent with established data privacy best practices.

However, we recognize scenarios where AI-assisted mapping suggestions could benefit from access to personal data to improve accuracy. In such cases, only after local heuristics fail, users will be explicitly asked to consent (opt-in) before any personal data is processed or transmitted externally.

We will try to avoid such scenarios, but if no better alternative exists, the system must be designed to minimize spread of such data inside our operational infrastructure. Wherever personal data may appear in telemetry (logs, traces, etc.), it may be necessary to omit or replace it with pseudo-data. While this may limit troubleshooting for this category of API calls, we accept these consequences where GDPR compliance requires it.

We commit to maintaining strict compliance with GDPR principles, ensuring transparency, user control, and data minimization throughout all AI-related workflows. This approach balances innovation with privacy and legal responsibility.

Bibliography

Ahmad, W. U., Chakraborty, S., Ray, B., & Chang, K. W. (2021). Unified pre-training for program understanding and generation. arXiv preprint arXiv:2103.06333.

Allamanis, M., Barr, E. T., Devanbu, P., & Sutton, C. (2018). A survey of machine learning for big code and naturalness. ACM Computing Surveys (CSUR), 51(4), 1–37.

Alon, U., Zilberstein, M., Levy, O., & Yahav, E. (2019). code2vec: Learning distributed representations of code. Proceedings of the ACM on Programming Languages, 3(POPL), 1–29.

AI Infrastructure Landscape (2023), available at https://ai-infrastructure.org/ai-infrastructure-landscape/, retrieved at 5th June 2025.

AI Observability: Ensuring Trust and Transparency in Machine Learning Pipelines (2025), available at https://www.computer.org/publications/tech-news/trends/ai-observability-trust-transparency, retrieved at 6th June 2025.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., … & Sutton, C. (2021). Program synthesis with large language models. arXiv preprint arXiv:2108.07732.

Banerjee, S., & Lavie, A. (2005, June). METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization (pp. 65–72).

Chandrachood, A. (2023). The Importance of Observability in Modern Software Applications. J Artif Intell Mach Learn & Data Sci 2023, 1(2), 472–475.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., … & Zaremba, W. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

Denkowski, M., & Lavie, A. (2014, June). Meteor universal: Language specific translation evaluation for any target language. In Proceedings of the Ninth Workshop on Statistical Machine Translation (pp. 376–380).

Dhamankar, R., Lee, Y., Doan, A., Halevy, A., & Domingos, P. (2004, June). iMAP: Discovering complex semantic matches between database schemas. In Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (pp. 383–394).

Dibia, V., Fourney, A., Bansal, G., Poursabzi-Sangdeh, F., Liu, H., & Amershi, S. (2022). Aligning offline metrics and human judgments of value for code generation models. arXiv preprint arXiv:2210.16494.

Do, H. H., & Rahm, E. (2002, January). COMA—a system for flexible combination of schema matching approaches. In VLDB'02: Proceedings of the 28th International Conference on Very Large Databases (pp. 610–621). Morgan Kaufmann.

Ellis, K., Ritchie, D., Solar-Lezama, A., & Tenenbaum, J. (2018). Learning to infer graphics programs from hand-drawn images. Advances in Neural Information Processing Systems, 31.

Feng, L., Li, H., & Zhang, C. J. (2024). Cost-Aware Uncertainty Reduction in Schema Matching with GPT-4: The Prompt-Matcher Framework. arXiv preprint arXiv:2408.14507.

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., … & Zhou, M. (2020). CodeBERT: A pre-trained model for programming and natural languages. arXiv preprint arXiv:2002.08155.

Fuxman, A., Miller, R. J. (2009). Schema Mapping. In: LIU, L., ÖZSU, M. T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA.

Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., … & Zhou, M. (2020). GraphCodeBERT: Pre-training code representations with data flow. arXiv preprint arXiv:2009.08366.

Kang, J., & Naughton, J. F. (2003, June). On schema matching with opaque column names and data values. In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (pp. 205–216).

Kementsietsidis, A. (2009). Schema Matching. In: LIU, L., ÖZSU, M. T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA.

Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). Machine Learning Operations (MLOps): Overview, Definition, and Architecture. IEEE Access, 11, 31866–31879

Lai, H., Liu, X., Iong, I. L., Yao, S., Chen, Y., Shen, P., … & Tang, J. (2024, August). AutoWebGLM: A Large Language Model-based Web Navigating Agent. In Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (pp. 5295–5306).

Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., … & de Vries, H. (2023). Starcoder: may the source be with you!. arXiv preprint arXiv:2305.06161.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., … & Vinyals, O. (2022). Competition-level code generation with alphacode. Science, 378(6624), 1092–1097.

Li, Y., Li, J., Suhara, Y., Doan, A., Tan, W. C. (2020). Deep entity matching with pretrained language models. arXiv preprint arXiv:2004.00584.

Lin, Ch.-Y. (2004). ROUGE: A package for automatic evaluation of summaries. In Proceedings of the Workshop on Text Summarization Branches Out, 2004.

Lozhkov, A., Li, R., Allal, L. B., Cassano, F., Lamy-Poirier, J., Tazi, N., … & de Vries, H. (2024). Starcoder 2 and the stack v2: The next generation. arXiv preprint arXiv:2402.19173.

Lu, Y., & Wang, J. (2025). KARMA: Leveraging Multi-Agent LLMs for Automated Knowledge Graph Enrichment. arXiv preprint arXiv:2502.06472.

Madhavan, J., Bernstein, P. A., & Rahm, E. (2001, September). Generic schema matching with cupid. In VLDB (Vol. 1, No. 2001, pp. 49–58).

Melnik, S., Garcia-Molina, H., & Rahm, E. (2002). Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, pp. 117–128.

MLOps at a Reasonable Scale \[The Ultimate Guide] (2024), available at https://neptune.ai/blog/mlops-at-reasonable-scale, retrieved at 6th June 2025.

Nijkamp, E., Hayashi, H., Xiong, C., Savarese, S., & Zhou, Y. (2023). Codegen2: Lessons for training llms on programming and natural languages. arXiv preprint arXiv:2305.02309.

Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002, July). BLEU: a method for automatic evaluation of machine translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (pp. 311–318).

Popović, M. (2015, September). chrF: character n-gram F-score for automatic MT evaluation. In Proceedings of the Tenth Workshop on Statistical Machine Translation (pp. 392–395).

Rahm, E., & Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. The VLDB Journal, 10, 334–350.

Ren, S., Guo, D., Lu, S., Zhou, L., Liu, S., Tang, D., ... & Ma, S. (2020). CodeBLEU: a method for automatic evaluation of code synthesis. arXiv preprint arXiv:2009.10297.

Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., ... & Synnaeve, G. (2023). Code llama: Open foundation models for code. arXiv preprint arXiv:2308.12950.

Shraga, R., Gal, A., Roitman, H. (2020). Adnev: cross-domain schema matching using deep similarity matrix adjustment and evaluation. Proc. VLDB 13(9), 1401–1415.

Sutanta, E., Wardoyo, R., Mustofa, K., & Winarko, E. (2016). Survey: Models and Prototypes of Schema Matching. International Journal of Electrical & Computer Engineering (2088-8708), 6(3).

Tran, N., Tran, H., Nguyen, S., Nguyen, H., & Nguyen, T. (2019, May). Does BLEU score work for code migration? In 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC) (pp. 165–176). IEEE.

Wang, S., Ding, L., Shen, L., Luo, Y., He, Z., Yu, W., & Tao, D. (2024). \$\mathbb{USCD}\$: Improving Code Generation of LLMs by Uncertainty-Aware Selective Contrastive Decoding. arXiv preprint arXiv:2409.05923.

Wang, Y., Wang, W., Joty, S., & Hoi, S. C. (2021). CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. arXiv preprint arXiv:2109.00859.

Xu, F. F., Alon, U., Neubig, G., & Hellendoorn, V. J. (2022, June). A systematic evaluation of large language models of code. In Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming (pp. 1–10).

Zhang, J., Shin, B., Choi, J. D., & Ho, J. C. (2021). SMAT: An attention-based deep learning solution to the automation of schema matching. In Advances in Databases and Information Systems: 25th European Conference, ADBIS 2021, Tartu, Estonia, August 24–26, 2021, Proceedings 25 (pp. 260–274). Springer International Publishing.

Zhang, Y., Ma, Z., Ma, Y., Han, Z., Wu, Y., & Tresp, V. (2024). Webpilot: A versatile and autonomous multi-agent system for web task execution with strategic exploration. arXiv preprint arXiv:2408.15978.