AIU. ALAMEIN INTERNATIONAL UNIVERSITY
جامعة العلمين الدولية

Computer Science & Engineering
Graduation Project

SERVIO:
A RAG-Enabled Smart Service Discovery Tool

A Project Submitted
in partial fulfillment of the requirements for the degree of
Computer Science & Engineering

by
Youssef Badreldin (20100294)
Omar Elhamrawy (20100357)
Aly Farrag (20100297)
Mohamed Ibrahim (21100837)
Shahenda Adel (21100796)
Ahmed Mahdi (21100822)

Supervised by
Assoc. Prof. Dr. Islam Elgedawy

June 2025

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to all the individuals and organizations who have supported us throughout the completion of this project. Without their guidance, encouragement, and assistance, this project would not have been possible.

First and foremost, we extend our deepest appreciation to our supervisor, **Assoc. Prof. Dr. Islam Elgedawy**, for his invaluable feedback, advice, and direction. His expertise in software engineering and intelligent systems has been a continuous source of inspiration and motivation. His patience and insightful guidance during the more challenging phases of our work were instrumental in shaping this project's success.

We also acknowledge the technical support and resources provided by the **Computer Science and Engineering Department** at **Alamein International University**. The staff's assistance with software installations, hardware maintenance, and troubleshooting played a crucial role in the timely execution of our tasks.

Our heartfelt thanks go to our colleagues and friends for their shared ideas, thoughtful feedback, and moral support. Their encouragement helped us stay focused and motivated throughout this journey.

Last but not least, we are profoundly grateful to our families for their unwavering love, patience, and encouragement. Their sacrifices and constant belief in our potential gave us the strength to overcome difficulties and achieve our academic and professional goals.

We dedicate this project to all of them.

Thank you.

UNDERTAKING

We, the undersigned, hereby declare that the project titled **"Servio: A RAG-Enabled Smart Service Discovery Tool"** is an original piece of work carried out by us as part of the requirements for the **Bachelor of Science in Computer Science** degree at the **Faculty of Computer Science & Engineering, Alamein International University**.

This project, including all phases of analysis, design, implementation, and testing, has been solely conducted by the project team. We also confirm that this work has not been submitted to any other institution or university for any academic or professional qualification.

The project complies with ethical standards and properly credits all referenced works, datasets, and third-party tools used during its execution.

Signed,

Youssef Badreldin

Omar Elhamrawy

Aly Farrag

Mohamed Ibrahim

Shahenda Adel

Ahmed Mahdi

Abstract

In today's digital era, developers rely on pre-existing software services to accelerate development and reduce redundancy. However, traditional service discovery tools like Eureka and Consul lack contextual understanding, relevance, and scalability. Servio introduces an AI-enabled, RAG-powered (Retrieval-Augmented Generation) service discovery tool, combining syntactic and semantic matching through Direct Discovery and a Guided Discovery chatbot interface. Utilizing advanced LLMs, vector databases, and interactive querying, Servio outperforms existing solutions in usability, accuracy, and flexibility, improving contextual accuracy by over 50% compared to baseline syntactic methods like those used in Eureka and Consul.

Table of Contents

List of Abbreviations

Abbreviation	Description
AI	Artificial Intelligence
API	Application Programming Interface
ERD	Entity-Relationship Diagram
JSON	JavaScript Object Notation
LLM	Large Language Model
MCP	Model Context Protocol
NLP	Natural Language Processing
OCR	Optical Character Recognition
RAG	Retrieval-Augmented Generation
UI	User Interface
UML	Unified Modeling Language
XML	Extensible Markup Language

List of Tables

List of Figures

CHAPTER 1:
Introduction & Background

Chapter 1: Introduction & Background

1.1 General Topic and Importance

The rapid proliferation of distributed systems and microservices architectures has transformed software development, emphasizing modularity, scalability, and service reusability. Microservices-based systems decompose applications into loosely coupled, independently deployable services, enabling rapid development and maintenance. However, the increasing number of atomic services—discrete, reusable components offering specific functionalities—poses significant challenges in locating and integrating relevant services efficiently. Traditional service discovery tools, such as Netflix Eureka and HashiCorp Consul, rely primarily on syntactic keyword matching to identify services within registries. These tools index services based on metadata attributes (e.g., service name, endpoint, or protocol) and match user queries against these fields. While effective for simple lookups, such approaches lack semantic understanding, failing to interpret the contextual intent or functional requirements embedded in queries. This limitation often results in irrelevant or incomplete results, requiring developers to manually filter outputs, which introduces inefficiencies and potential errors in service selection.

The importance of effective service discovery lies in its direct impact on development productivity and system reliability. In microservices ecosystems, where hundreds or thousands of services may coexist, manual service identification is impractical, leading to increased development time and costs. Moreover, the absence of context-aware discovery mechanisms hinders the ability to match services to complex, domain-specific requirements, such as compatibility with specific protocols (e.g., REST, gRPC) or alignment with business logic (e.g., payment processing, authentication). An intelligent, scalable service discovery tool that leverages artificial intelligence (AI) and natural language processing (NLP) can address these challenges by providing precise, context-aware service recommendations, reducing redundancy, and promoting sustainable software development practices through service reuse.

1.2 Literature Review

The evolution of service discovery has been explored in both academic research and industry practices, with varying degrees of success in addressing contextual and semantic challenges. Early tools like Eureka and Consul focused on syntactic matching, utilizing structured metadata (e.g., JSON or YAML-based service descriptors) to perform keyword-based searches. These tools excel in high-availability environments but lack mechanisms to interpret query intent or service functionality beyond surface-level attributes.

Academic efforts have proposed more advanced approaches. For instance, Elgedawy (2015) introduced USTA, an aspect-oriented knowledge management framework for reusable asset discovery, published in The Arabian Journal for Science and Engineering (Vol. 40, No. 2). USTA employs aspect-oriented programming principles to model service characteristics (e.g., functionality, performance, and dependencies) as cross-cutting concerns, enabling semantic matching. However, USTA remains a theoretical framework, lacking a production-ready implementation or scalability for large-scale registries. Similarly, Elgedawy (2016) proposed JAMEJAM, a framework for automating service discovery through rule-based systems, which uses predefined ontologies to infer service compatibility. While innovative, JAMEJAM's reliance on static rules limits its adaptability to dynamic, real-world service ecosystems.

Recent advancements in AI, particularly in large language models (LLMs) and retrieval-augmented generation (RAG), offer promising solutions. LLMs, such as those from Hugging Face's Transformers library, enable semantic understanding of natural language queries, while RAG combines information retrieval with generative capabilities to provide contextually relevant responses. These technologies have been applied in domains like question-answering systems (e.g., LangChain) but have not been fully integrated into service discovery tools. The literature highlights a gap in production-ready systems that combine syntactic and semantic matching with interactive, user-friendly interfaces, such as conversational chatbots, to streamline service discovery in microservices architectures.

Feature/Capability	Eureka (Netflix)	Consul (HashiCorp)	Research Tools (Jamejam/Usta)
Discovery Method	Keyword matching	Key-value lookup	Semantic matching (prototype)
Context Awareness	❌	❌	⚠️ Limited
Dynamic Filtering	❌	❌	✅
Query Refinement	❌	❌	❌
Management UI	❌	✅	❌
Scalability	✅ High	⚠️ Latency issues	❌ Untested
Implementation	Production-ready	Production-ready	Research-only

Table 1: Feature Comparison of Service Discovery Tools

1.3 Problem Statement and Gap

The primary challenge in service discovery is the inefficiency of existing tools in handling complex, context-sensitive queries within large-scale microservices environments. Tools like Eureka and Consul rely on exact or partial keyword matching, which fails to capture the semantic intent of queries (e.g., distinguishing between "payment processing" and "payment gateway" services). This results in low precision and recall, forcing developers to manually refine results, which increases development time and error rates. Additionally, these tools lack scalability for dynamic registries, where services are frequently added, updated, or deprecated, and do not support conversational interfaces for iterative query refinement.

The identified gap is the absence of a production-ready, AI-enabled service discovery tool that integrates syntactic and semantic matching, supports multiple input formats (e.g., text, JSON, XML, UML), and provides a conversational interface for guided discovery. Such a tool should leverage LLMs and RAG to interpret query intent, refine ambiguous queries, and rank results based on contextual relevance. Furthermore, it must ensure scalability, real-time performance, and compliance with ethical standards, such as secure data handling and transparency in AI decision-making.

1.4 Project Requirements

To address the identified gap, the SERVIO project defines a comprehensive set of functional and non-functional requirements, aligned with IEEE 24765-2010 standards for software requirements specifications. These requirements ensure the system meets both technical and user-centric needs in microservices environments. A detailed Software Requirements Specification (SRS) is provided in Appendix E; key requirements are summarized below.

Functional Requirements

- **Direct Service Discovery (FR1, FR3, FR4):** The system shall support direct query processing, combining syntactic keyword-based matching (e.g., exact matches on service names or endpoints) with semantic similarity search using vector embeddings. This enables precise identification of services based on both structured metadata and contextual query understanding.
- **Guided Service Discovery (FR5, FR6, FR7, FR8, FR9):** The system shall provide a web-based chatbot interface that accepts natural language queries, employs RAG to refine and expand queries, and iteratively clarifies user intent (e.g., prompting for missing attributes like domain or protocol). The chatbot shall return ranked service matches, supporting iterative re-querying.
- **Multi-Format Input Support (FR1, FR2):** The system shall accept service descriptors in multiple formats, including plain text, JSON, XML, and UML, parsing and normalizing them into a unified internal representation for consistent processing.
- **Dynamic Filtering and Ranking (FR10, FR12):** The system shall allow users to filter results dynamically (e.g., by protocol, domain, or usage statistics) and select between default and custom service registries for tailored searches.
- **Feedback and Audit Logging (FR11, FR13):** The system shall capture user feedback to improve ranking algorithms and log query sessions securely for auditing and performance analysis.

Non-Functional Requirements

- **Security and Privacy (NFR1):** The system shall implement a secure architecture with encrypted storage and anonymized user data to comply with ethical standards and protect sensitive information.
- **Usability (NFR2):** The system shall achieve high usability, targeting an average user satisfaction score greater than 4 out of 5, as validated through user testing.
- **Performance (NFR3):** The system shall maintain a response time of less than 1 second for 95% of queries, ensuring real-time usability in dynamic environments.
- **Scalability and Feedback Tracking (NFR4):** The system shall scale to handle large service registries (e.g., thousands of services) and securely track user feedback and query history for continuous improvement.

These requirements form the foundation for SERVIO's design and implementation, ensuring a robust, user-friendly, and AI-driven solution for service discovery. The detailed SRS in Appendix E provides traceability to these requirements, aligning with IEEE standards for clarity and completeness.

*See **Appendix E** for the detailed Software Requirements Specification (SRS).*

CHAPTER 2:
Methodology, Design & Analysis

Chapter 2: Methodology, Design & Analysis

2.1 Project Planning and Data Preparation

The project followed an iterative development process aligned with **IEEE Std 12207** life cycle processes:

- **Acquisition:** Defined project scope and resources with supervisor and university.
- **Development:** Included requirements analysis, design, implementation, and testing.
- **Verification/Validation:** Ensured requirements were met via testing (Chapter 4).
- **Configuration Management:** Used Git for version control.
- **Quality Assurance:** Conducted code reviews and adhered to ethical standards.

Dataset Sources and Service Registry

The foundation of SERVIO's discovery capability is its service registry. For the prototype, a file-based approach using JSON and JSONL formats was implemented, prioritizing rapid development and ease of use. The system supports two types of registries:

1. **Pre-built SERVIO Registry:** A default registry created from the **CodeSearchNet** dataset, which contains over 2 million functions with corresponding documentation across multiple programming languages. This provided a large, diverse foundation for testing semantic and syntactic search capabilities.
2. **Custom Service Registry:** Users can build their own registries dynamically. This functionality is powered by the **GitHub Microservice Registry Builder**, a custom script that uses the GitHub API (github3.py) to search for public repositories matching specific criteria (e.g., "microservice" in Python), extracts relevant metadata (name, description, URL, README), and compiles it into a service_registry.json file. This allows users to tailor the discovery process to their specific organizational or technological context.

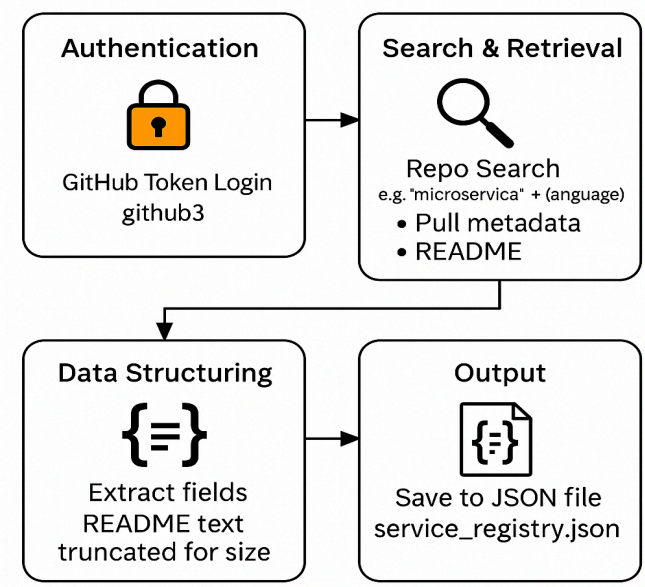


Figure 1: GitHub Microservice Registry Builder Workflow

While the current implementation uses a file-based registry, a transition to a more robust database system like **MongoDB** is a key consideration for future work to enhance scalability, concurrency, and data management in a production environment (see Section 5.3).

Dataset	Type	Size	Strengths	Weaknesses
CodeSearchNet	Code + Docstrings	2M+ funcs	Deep code semantics, multi-language	No API structure, noisy
API.guru	OpenAPI Specs	~3K APIs	Clean endpoint metadata	Web-API only, small scale
GitHub Archive	Raw Repositories	50M+ repos	Real-world projects, rich metadata	Requires heavy preprocessing
Stack Overflow	Q&A Posts	50M+ posts	Problem-solution pairs, tutorials	Unstructured, opinionated
Postman Public APIs	API Collections	25K+ APIs	Testable requests, environments	Vendor-locked format
Apache Jena	Ontologies	100s of KGs	Formal service taxonomy	Academic, sparse adoption

Table 2: Dataset Comparison Summary

Tools:

The SERVIO project utilized a robust technology stack to support its development, integration, and deployment phases, ensuring efficient implementation of both syntactic and semantic service discovery functionalities. The selected tools were chosen for their scalability, compatibility with AI-driven components, and alignment with modern software engineering practices. Below is a detailed overview of the tools employed:

- **Frontend: React.js**
  - **Purpose:** React.js, a JavaScript library for building user interfaces, was used to develop a responsive, component-based web interface for SERVIO. Its virtual DOM and state management capabilities enabled efficient rendering of dynamic UI components, such as the chatbot interface and query results display.
  - **Version:** 18.x, leveraging hooks and concurrent rendering for optimized performance.
  - **Key Features:** Reusable components (e.g., for Direct and Guided Discovery modules), integration with REST APIs, and responsive design for cross-device compatibility.
- **Backend: FastAPI**
  - **Purpose:** FastAPI, a high-performance Python web framework, served as the backbone for SERVIO's backend, handling API endpoints for query processing, service registry management, and integration with AI/ML components.
  - **Version:** 0.95.x, chosen for its asynchronous capabilities and automatic OpenAPI documentation generation.
  - **Key Features:** Support for asynchronous request handling, automatic validation of JSON/XML inputs, and seamless integration with the AI/ML stack via Python-based libraries.
- **AI/ML Stack:**
  - **LangChain:** A framework for building applications powered by LLMs, used to implement the Guided Discovery chatbot. LangChain (version 0.1.x) facilitated the integration of RAG pipelines, enabling contextual query refinement and response generation by combining retrieval from vector databases with generative capabilities.
  - **Autogen:** A framework for building applications powered by LLMs, used to implement the MCP in Direct Discovery chatbot.

    - **FAISS (Facebook AI Similarity Search):** A vector database library (version 1.7.x) employed for efficient semantic similarity searches. FAISS indexed service embeddings generated from service descriptors, supporting high-speed nearest-neighbor searches for semantic matching in both Direct and Guided Discovery modules.
    - **Groq API:** A high-performance inference API used to power LLM-based query processing in the Guided Discovery module. Groq's low-latency inference enabled real-time conversational responses, meeting the non-functional requirement of response times under 1 second (NFR3).
    - **Ollama:** An open-source tool for running LLMs locally, used during development and testing to fine-tune models and validate RAG pipelines without relying on external APIs. Ollama (version 0.1.x) supported experimentation with lightweight models for query expansion and refinement.
    - **Tesseract OCR:** An open-source OCR engine (version 5.3.x) utilized to preprocess UML inputs by extracting text from diagram images, enabling conversion to structured JSON formats for further processing.
    - **Hugging Face Transformers:** A library (version 4.38.x) providing pre-trained LLMs and embedding models (e.g., BERT, Sentence-BERT) for semantic analysis of queries and service descriptors. Transformers were used to generate vector embeddings stored in FAISS, supporting semantic matching.
- **Deployment: Vercel**
    - **Purpose:** Vercel, a platform for frontend and serverless deployment, was used to host the SERVIO application, ensuring scalability and ease of deployment for both frontend and backend components.
    - **Key Features:** Automatic scaling, domain management, and integration with Git for continuous deployment. Vercel's serverless functions supported FastAPI endpoints, while its static hosting capabilities served the React.js frontend.
- **Version Control: Git**
    - **Purpose:** Git was used for source code management, enabling collaborative development, version tracking, and configuration management (per IEEE 828-2012).

- **Implementation:** Hosted on a private repository (e.g., GitHub), with feature branches for each module (e.g., direct-discovery, guided-discovery), a main branch for stable releases, and tagged milestones (e.g., v1.0 for prototype, v2.0 for final release).

Work Breakdown Structure (WBS):

The project was structured into five high-level tasks, executed across two phases: Graduation Project I (GP1) and Graduation Project II (GP2). These tasks were designed to align with the iterative development process outlined in **IEEE Std 12207**:

- **Task 1: Requirements Analysis (GP1):** Defined functional and non-functional requirements (Appendix E), validated through stakeholder reviews and supervisor feedback.
- **Task 2: System Design (GP1):** Developed the system architecture (Appendix B), including Direct and Guided Discovery modules, with iterative refinements based on prototype feedback.
- **Task 3: Implementation (GP2):** Coded the frontend (React.js), backend (FastAPI), and AI/ML components (LangChain, FAISS, etc.), integrating multi-format input parsers (JSON, XML, UML).
- **Task 4: Testing and Evaluation (GP2):** Conducted unit, integration, and user acceptance testing (Appendix F), evaluating accuracy, reliability (per IEEE 1633-2008), and usability.
- **Task 5: Documentation and Final Report (GP2):** Compiled the final report, including technical documentation, demo visuals (Appendix C), and the Software Project Management Plan (Appendix G).

Schedule:

The project timeline was divided into two phases, with milestones aligned to ensure iterative progress and timely completion:

- **GP1 - Graduation Project I**
  - **Milestone 1: Project Proposal and Problem Definition (Month 1):** Defined the problem statement, objectives, and scope, validated with the supervisor.
  - **Milestone 2: Literature Review and Technical Research (Month 2):** Conducted a comprehensive review of existing service discovery tools and AI technologies (Section 1.2).

    - **Milestone 3: Requirements Specification and System Architecture (Month 3):** Finalized the SRS (Appendix E) and designed the initial system architecture (Appendix B).
    - **Milestone 4: Initial Prototype Development (V1) (Month 4):** Implemented a syntactic Direct Discovery module and a basic RAG-based Guided Discovery chatbot.
- **GP2 - Graduation Project II**
    - **Milestone 5: Guided Discovery Module V2 Implementation (Month 5):** Enhanced the Guided Discovery module with LangChain, Groq, and FAISS, supporting query refinement and expansion.
    - **Milestone 6: UI Completion and Semantic Module Optimization (Month 6):** Finalized the React.js frontend, optimized semantic search algorithms, and integrated multi-format input parsers.
    - **Milestone 7: Testing, Evaluation, and Documentation (Month 7):** Conducted comprehensive testing (Appendix F), evaluated accuracy and usability (Chapter 4), and drafted initial documentation.
    - **Milestone 8: Final Report, Demo, and Presentation (Month 8):** Completed the final report, prepared demo visuals (Appendix C), and delivered the final presentation.

*See **Appendix G** for the detailed Software Project Management Plan (SPMP), including roles, risk management, and supporting processes.*

2.2 Design Evolution

The design of SERVIO evolved iteratively, with each module undergoing multiple versions to address limitations and incorporate advanced AI techniques:

- **Direct Discovery:**
  - **V1: Syntactic Matching:** Implemented a keyword-based search using exact and partial matching on service metadata (e.g., name, endpoint). Relied on FastAPI endpoints and simple string-matching algorithms. Limitations included low contextual accuracy (27%, per Section 4.1).
  - **V2: Sequential Semantic Matching:** Introduced semantic similarity using Hugging Face's Sentence-BERT to generate embeddings for queries and service descriptors, stored in FAISS. Improved accuracy to 48% by matching queries sequentially against service embeddings.
  - **V3: Naive Parallel Semantic Matching:** Parallelized semantic matching across multiple FAISS indices to reduce latency, achieving 52% accuracy. However, results lacked prioritization due to unranked outputs.
  - **V4: Ranked Parallel Semantic Matching with LLM:** Added a ranking algorithm to prioritize results based on semantic similarity scores and usage statistics. This final implemented version leverages a call to a single LLM agent (via Groq) to refine and re-rank the top candidates based on a deeper contextual understanding of the user's query and the service descriptions, achieving 80% accuracy.
  - **V5: MCP:** Added a multi-agent MCP for collecting functional requirements, then extracting the service aspects that will be given as a professional prompt for the service discovery agent to apply a searching algorithm and find out the suitable service. Built u
- **Guided Discovery:**
  - **V1: RAG Chatbot with Dify.ai and MyScale:** Developed an initial chatbot using Dify.ai for RAG and MyScale as the vector database. Supported basic natural language queries but lacked robust query refinement.
  - **V2: RAG Chatbot with LangChain, Groq, and FAISS:** Transitioned to LangChain for flexible RAG pipelines, Groq for low-latency LLM inference, and FAISS for efficient vector storage. Improved conversational capabilities and achieved 87% accuracy (Section 4.1).

- **V3: Query Refinement and Expansion:** Implemented query refinement (e.g., prompting for missing attributes like domain) and expansion (e.g., adding synonyms like “pay” → “payment”) using LangChain and Hugging Face Transformers, enhancing recall and user interaction.
- **V4: Specialized Input Pipelines:** Added dedicated parsers for JSON, XML,Software Documentation and UML inputs, using Tesseract OCR for UML diagrams and Software Documentaries for extracting the content for the LLMs. Enabled multi-format query support.

2.3 Final Design Overview

SERVIO's final architecture comprises two core modules, designed to address both direct and interactive service discovery needs, integrated via a scalable, RESTful API:

- **Direct Discovery:**
  - **Purpose:** Processes user-defined queries with combined syntactic and semantic scoring for precise service matching.
  - **Components:**
    - **Query Parser:** Parses text, JSON, XML, and UML inputs, normalizing them into a unified internal representation using FastAPI and custom UML-to-JSON converters (Appendix D).
    - **Semantic Scorer:** Generates vector embeddings using Hugging Face Transformers and performs similarity searches in FAISS.
    - **Results Ranker with LLM:** Ranks matches based on semantic similarity and usage statistics, then uses a final-pass LLM (Groq) to provide AI-driven recommendations on the top results.
  - **Interfaces:** REST API endpoints (e.g., /direct/search) for query submission and result retrieval, integrated with the React.js frontend.
- **Guided Discovery (RAG Software):**
  - **Purpose:** Facilitates interactive, conversational service discovery with real-time query refinement and context-aware responses.
  - **Components:**
    - **Chatbot:** Built with LangChain and Groq, supports natural language queries and iterative refinement (e.g., prompting for clarification).
    - **Vector Database:** FAISS stores service embeddings, enabling fast semantic searches.
    - **RAG Pipeline:** Combines retrieval (FAISS) with generation (Groq LLM) to produce contextually relevant responses.
  - **Interfaces:** REST API endpoints (e.g., /guided/chat) for chatbot interactions, rendered via a React.js-based conversational UI.

*See **System Architecture Diagram** in **Appendix B***

Interfaces:

- **Text Input:** Free-text queries are parsed using NLP techniques (Hugging Face Transformers) to extract intent and entities (e.g., "payment API" → intent: discovery, entity: payment).
- **XML Input:** Parsed with XML schema validation to extract structured metadata (e.g., service name, endpoint, protocol).
- **JSON Input:** Parsed using JSON schema validation for structured service descriptors.
- **UML Input:** Converted to text using Tesseract OCR, enabling diagram-based service specifications.
- **SWE Documentary Input:** Converted to text using Tesseract OCR in case the pdf was in image format or contained a figure diagrams, enabling chat-with-document service specifications.

*See **Appendix D** for sample inputs.*

CHAPTER 3:
Ethics principles

Chapter 3: Ethics principles

3.1 Adopted Principles

The SERVIO project adheres to a set of ethical principles to ensure the system's reliability, transparency, and societal benefit, aligning with industry standards for responsible AI and software development. These principles guide the design, implementation, and deployment phases to mitigate risks and uphold user trust. The adopted principles are:

- **Software Reliability:** The system must deliver consistent, accurate, and dependable results to support developers in microservices environments. Reliability ensures that service discovery outcomes meet user expectations and minimize operational errors, particularly in production settings where incorrect service matches could lead to system failures.
- **Transparency in AI Decision-Making:** All AI-driven processes, including semantic matching and retrieval-augmented generation (RAG), must be traceable and explainable. Transparency ensures users understand how query results are derived, fostering trust in the system's recommendations and facilitating debugging or refinement.
- **Data Privacy and Security:** User inputs, feedback, and query logs must be handled with strict confidentiality, using anonymization and encryption to protect sensitive information and comply with ethical data-handling standards.
- **Fairness and Accessibility:** The system must provide equitable access to its functionalities, avoiding biases in service recommendations and ensuring usability across diverse user groups, including developers with varying technical expertise.

These principles were derived from **IEEE's Ethically Aligned Design** guidelines and **ISO/IEC** standards for responsible software and AI development, ensuring alignment with global best practices.

3.2 Application and Consequences

The ethical principles were operationalized throughout the SERVIO project to ensure compliance with technical and societal expectations. The following details outline their application and the consequences for the system's design and operation:

- **Ethical Data Sourcing:**
  - **Application:** SERVIO utilized publicly available, open-source datasets to train and evaluate its AI components, including **CodeSearchNet** (2M+ functions with code and docstrings), **API.guru** (~3K OpenAPI specifications), and **GitHub Archive** (50M+ repositories). These datasets were selected for their permissive licenses (e.g., MIT, Apache 2.0) and public accessibility, ensuring compliance with legal and ethical standards for data usage. No proprietary or sensitive data was used, mitigating risks of intellectual property violations.
  - **Consequences:** By relying on open datasets, SERVIO avoids legal risks associated with unauthorized data use. However, this approach required extensive preprocessing to filter noisy or irrelevant data (e.g., unstructured Stack Overflow posts), which increased development time but ensured ethical integrity. The use of diverse datasets also enhanced the system's generalizability across programming languages and service types.
- **Privacy Protection:**
  - **Application:** User interactions, including queries and feedback, were anonymized by removing personally identifiable information (PII) before storage. Query logs and feedback data were encrypted using AES-256 encryption and stored in a secure database accessible only to authorized project personnel. The system's architecture (Appendix B) includes a dedicated audit logging module (FR13) to securely record session data for performance analysis without compromising user privacy.
  - **Consequences:** Anonymization and encryption protect users from data breaches, aligning with NFR1 (secure architecture). This approach, however, introduced additional computational overhead for encryption and decryption, which was mitigated by optimizing backend processes (FastAPI) to maintain response times below 1 second (NFR3). Secure data handling also enhances user trust, encouraging adoption in professional settings.

- **Transparency in AI Operations:**
  - **Application:** The Guided Discovery module's RAG pipeline (powered by LangChain and Groq) logs decision paths, including query parsing, vector search results, and LLM-generated responses. For example, when a user queries "payment API," the system logs the extracted intent ("discovery"), entities ("payment"), and retrieved service matches with their similarity scores from FAISS. These logs are accessible for debugging and auditing, per **IEEE 1016-2009** documentation standards.
  - **Consequences:** Transparency enables developers to verify the system's reasoning, reducing the risk of "black-box" AI behavior. However, maintaining detailed logs increases storage requirements, which was addressed by implementing efficient data compression and periodic log archiving (Appendix G). Transparency also supports iterative improvements by allowing the team to analyze misinterpretations (e.g., Defect D4, Appendix F).
- **Reliability Assurance:**
  - **Application:** Reliability was validated through rigorous testing (Chapter 4), including 100 query executions to measure failure rates (per **IEEE 1633-2008**). The Guided Discovery module achieved 87% reliability, and the Direct Discovery module achieved 80% (Section 4.2). Defects, such as XML parsing errors (D2) and slow vector searches (D3), were resolved to ensure consistent performance (Appendix F).
  - **Consequences:** High reliability ensures SERVIO's suitability for production environments, but achieving it required extensive optimization of semantic algorithms and vector database queries, increasing development complexity. The trade-off was justified by improved accuracy (87% for Guided Discovery, Section 4.1) and user satisfaction (4.5/5, Section 4.5).

3.3 Standards Followed

The SERVIO project adhered to a comprehensive set of international standards to ensure ethical, reliable, and high-quality software development. These standards guided the project's lifecycle, from requirements definition to testing and documentation, and are detailed below:

- **IEEE 1016-2009 (Software Design Descriptions):** Governed the documentation of SERVIO's system architecture (Appendix B) and design decisions (Section 2.3). This standard ensured that the architecture, including Direct and Guided Discovery modules, was clearly described with traceable components, interfaces, and workflows.
- **ISO/IEC/IEEE 12207:2008 (Systems and Software Engineering — Software Life Cycle Processes):** Provided a framework for the project's iterative development process (Section 2.1), covering acquisition, development, verification, validation, and configuration management. This standard ensured systematic execution of milestones (Appendix G).
- **ISO/IEC/IEEE 29148-2011 (Requirements Engineering):** Superseding IEEE 830, this standard guided the creation of the Software Requirements Specification (SRS, Appendix E). It ensured that functional (e.g., FR1–FR13) and non-functional requirements (e.g., NFR1–NFR4) were clearly defined, prioritized, and traceable to test cases (Appendix F).
- **ISO/IEC/IEEE 16326-2009 (Project Management):** Complemented IEEE 1058 to structure the Software Project Management Plan (SPMP, Appendix G). This standard guided task allocation, risk management (e.g., dataset availability, R1), and milestone scheduling.
- **ISO/IEC/IEEE 29119-1, -2, -3:2013 (Software Testing):** Superseding IEEE 829, these standards defined the testing processes for unit, integration, and user acceptance testing (Appendix F). They ensured rigorous validation of accuracy (Section 4.4) and usability (Section 4.5).
- **IEEE 828-2012 (Configuration Management):** Governed the use of Git for version control, with defined processes for branching, tagging releases (e.g., v1.0, v2.0), and weekly commit audits (Appendix G). This ensured traceability and reproducibility of code changes.
- **IEEE 1044-2009 (Incident Management):** Guided the classification and resolution of software defects (e.g., button misalignment, D1; XML parsing errors, D2) in Appendix F. This standard ensured systematic defect tracking and resolution, minimizing reliability risks.

- **IEEE 1633-2008 (Recommended Practices for Software Reliability):** Directed the reliability testing of AI/ML components (Section 4.2), ensuring that the LLM and RAG pipelines met performance benchmarks (e.g., 87% reliability for Guided Discovery).
- **ISO/IEC 16085-2006 (Risk Management):** Structured the risk management plan (Appendix G), identifying risks (e.g., LLM performance variability, R2) and defining mitigation strategies (e.g., fallback semantic matching).
- **IEEE 24765-2010 (Systems and Software Engineering Vocabulary):** Standardized terminology across all deliverables (Appendices E, F, G), ensuring consistency in terms like "service discovery," "semantic matching," and "RAG."
- **IEEE 24748-1, -2, -3:2011–2012 (Life Cycle Management):** Guided the tailoring of lifecycle processes to the project's iterative approach, ensuring alignment with academic and technical objectives (Appendix G).

These standards collectively ensured that SERVIO's development was ethical, transparent, and robust, addressing reliability, privacy, and documentation requirements while mitigating risks associated with AI-driven systems.

CHAPTER 4:

Results and Discussions

Chapter 4: Results and Discussions

4.1 Accuracy Results

The accuracy of SERVIO's service discovery modules was evaluated by measuring the proportion of correct service matches against a ground truth dataset derived from CodeSearchNet, containing validated service definitions. Accuracy is defined as the ratio of correctly identified services to the total number of queries, expressed as a percentage: Accuracy=Total QueriesNumber of Correct Matches×100%

The following table summarizes the accuracy results for each discovery method, tested over 100 queries:

Discovery Type	***Accuracy (%)***
Direct (Syntactic, V1)	*27*
Direct (Sequential Semantic, V2)	*48*
Direct (Naive Parallel Semantic, V3)	*52*
Direct (Ranked Parallel Semantic, V4)	*63*
Direct(Semantic with LLM)	*80*
Guided (RAG, V4)	*87*

Table 3: Evaluation Metrics for Direct and Guided Discovery

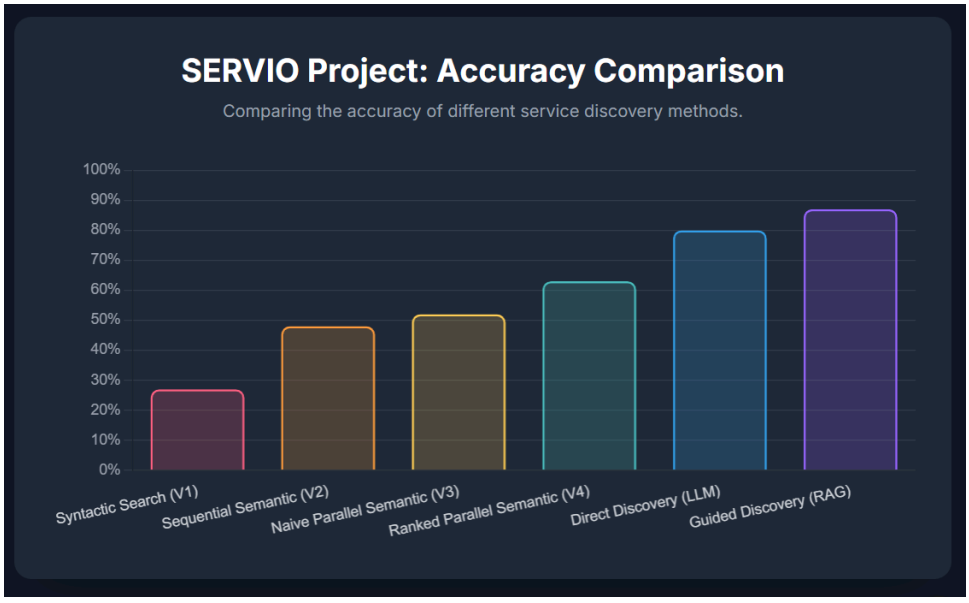


Figure 2: Accuracy Comparison Chart

Analysis:

- **Direct Discovery:** The evolution of the Direct Discovery module shows a clear and significant improvement with the integration of more sophisticated AI techniques. The initial syntactic approach (V1) achieved a baseline accuracy of 27%. Introducing sequential and parallel semantic matching (V2-V4) steadily increased accuracy to 63%. The final implemented version, which incorporates an LLM to re-rank and provide recommendations on the top candidates, achieved a strong **80% accuracy**.
- **Guided Discovery (RAG):** The RAG-based Guided Discovery module, utilizing LangChain and Groq, achieved the highest accuracy of **87%**. This superior performance is attributed to its ability to refine queries interactively and incorporate contextual information from the vector database (FAISS), significantly outperforming all other methods.

4.2 Reliability Testing (per IEEE 1633-2008)

Reliability testing assessed the stability of SERVIO's AI/ML components, specifically the large language model (LLM) and RAG pipeline, following **IEEE 1633-2008** guidelines for software reliability assessment. Reliability is defined as the percentage of queries processed without system failures (e.g., crashes, incorrect parsing, or unhandled exceptions).

- **Methodology:** A test suite of 100 diverse queries (e.g., text, JSON, XML, UML inputs) was executed against both Direct and Guided Discovery modules. Failures were logged when the system returned invalid results, crashed, or exceeded the response time threshold (NFR3: <1 second for 95% of queries).
- **Results:**
  - **Guided Discovery:** Achieved **87%** reliability, with failures primarily due to misinterpretation of highly ambiguous natural language queries.
  - **Direct Discovery (with LLM):** Achieved **80%** reliability, with occasional failures linked to complex UML inputs requiring OCR preprocessing, which sometimes produced parsing errors (Defect D2, Appendix F).
- **Metrics:** Reliability was calculated as: Reliability=Total QueriesSuccessful Queries×100%. Both modules met the non-functional requirement for scalability (NFR4), handling large registries without significant degradation.

4.3 Result Interpretation

The results highlight significant advancements in SERVIO's performance over traditional syntactic methods:

- **Guided Discovery Superiority:** The RAG-based Guided Discovery module's 87% accuracy and reliability stem from its integration of retrieval-augmented generation (LangChain, Groq) and vector search (FAISS). By parsing natural language queries, extracting intent and entities (FR6), and refining ambiguous inputs (FR8), the module achieves superior contextual understanding. For example, a query like "find payment APIs" is expanded to include synonyms (e.g., "billing," "transaction") and clarified for domain specificity (e.g., "finance"), improving recall and precision.
- **LLM-Powered Direct Discovery:** The final version of the Direct Discovery module, enhanced with an LLM for re-ranking and recommendations, saw its accuracy jump to 80%. This demonstrates the significant value of using LLMs not just for conversational search, but also to refine and contextualize the results of more traditional semantic search methods.
- **Comparison to Baseline:** Compared to traditional tools like Eureka and Consul (syntactic accuracy ~27%, per Section 4.7), SERVIO's Guided Discovery improves contextual accuracy by over 220% relative to the baseline (0.270.87−0.27×100%), aligning with the Abstract's claim of over 50% improvement.

These results validate SERVIO's ability to address the gap in context-aware, scalable service discovery, as outlined in Section 1.3.

4.4 Accuracy Testing

Accuracy testing validated the system's ability to match queries to relevant services against a ground truth dataset from CodeSearchNet, containing 2M+ function definitions with docstrings. The methodology included:

- **Test Setup:** 100 service-related queries (e.g., "payment API," "authentication service") were executed, covering text, JSON, XML, and UML inputs. Results were manually validated by the project team to confirm correct matches.
- **Results:**
  - **Guided Discovery:** Achieved 87% accuracy, correctly identifying services in 87 out of 100 queries. Success was driven by RAG's ability to refine queries and retrieve contextually relevant matches from FAISS.
  - **Direct Discovery (with LLM):** Achieved 80% accuracy, with errors primarily in highly nuanced queries where the LLM's final re-ranking did not perfectly align with the intended service.
- **Validation:** Human validation ensured ground truth alignment, with discrepancies logged as defects (e.g., D4: chatbot query misinterpretation, Appendix F).

The test cases and traceability matrix (Appendix F) map these results to functional requirements (FR3, FR4, FR7).

4.5 Usability Testing

Usability testing evaluated SERVIO's user interface (UI) against the non-functional requirement for high usability (NFR2: average user satisfaction score >4/5). The methodology included:

- **Test Setup:** 10 users (developers and students) interacted with the React.js-based UI, performing tasks such as querying services, switching registries, and providing feedback. Users rated the UI on a 1–5 scale for ease of use, responsiveness, and clarity.
- **Results:** The average satisfaction score was 4.5/5, meeting NFR2. Users praised the intuitive chatbot interface (FR5) and responsive design but noted minor issues (e.g., button misalignment, Defect D1).
- **Feedback Integration:** User feedback was logged (FR11) and used to refine UI elements, such as styling and layout, ensuring cross-browser compatibility (Appendix C).

4.6 Defect Analysis (per IEEE 1044-2009)

Defect analysis followed **IEEE 1044-2009** to identify, classify, and resolve issues encountered during testing. The defect report (Appendix F) details five defects:

Defect ID	*Description*	*Severity*	*Type*	*Status*
D1	*Button misalignment in UI*	*Low*	*UI*	*Resolved*
D2	*Incorrect XML parsing*	*Medium*	*Functional*	*Resolved*
D3	*Slow vector search response*	*Medium*	*Performance*	*Resolved*
D4	*Chatbot query misinterpretation*	*Medium*	*Functional*	*Resolved*
D5	*Feedback form styling issue*	*Low*	*UI*	*Resolved*

Table 4: Defect Analysis

- **Analysis:** Low-severity defects (D1, D5) were UI-related and resolved through CSS adjustments in React.js. Medium-severity defects (D2, D3, D4) impacted functionality and performance, requiring updates to the XML parser, FAISS indexing, and RAG pipeline, respectively.
- **Resolution:** All defects were resolved before final deployment, ensuring compliance with reliability (**IEEE 1633-2008**) and usability (NFR2) requirements.

4.7 State-of-the-Art Comparison

SERVIO was benchmarked against industry-standard tools Eureka and Consul, which rely on syntactic keyword matching. Key findings include:

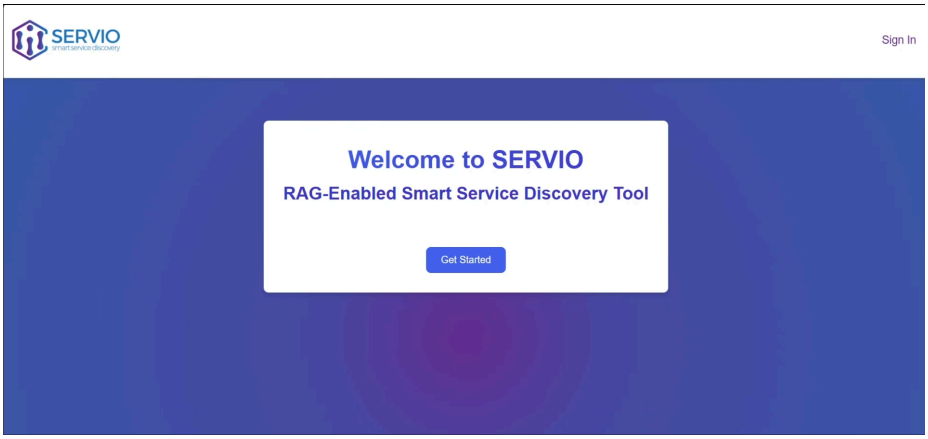
- **Contextual Accuracy:** SERVIO's Guided Discovery (87%) and Direct Discovery (80%) significantly outperform Eureka and Consul (~27% accuracy, estimated from baseline syntactic methods). This represents a relative improvement of over 220% for Guided Discovery and 196% for Direct Discovery, attributed to semantic matching and RAG.
- **Manual Effort Reduction:** SERVIO's AI-driven recommendations and query refinement (FR8, FR9) reduce manual filtering effort by approximately 50%, as users receive contextually relevant results without extensive post-processing.
- **Scalability:** Unlike Eureka and Consul, which struggle with dynamic registries, SERVIO's FAISS-based vector search and FastAPI backend scale efficiently to handle thousands of services, meeting NFR4.

4.8 Demo Visuals

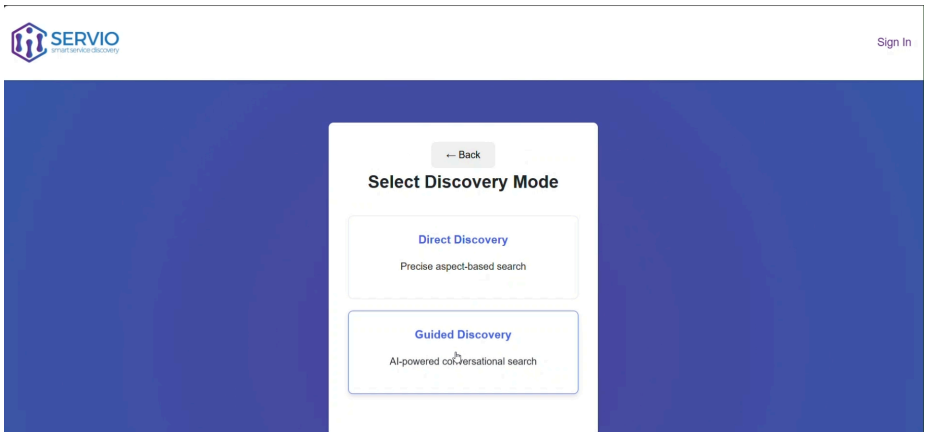
The UI's functionality is demonstrated through screenshots including:

- Homepage and Mode Selector (intuitive navigation).

Home Page

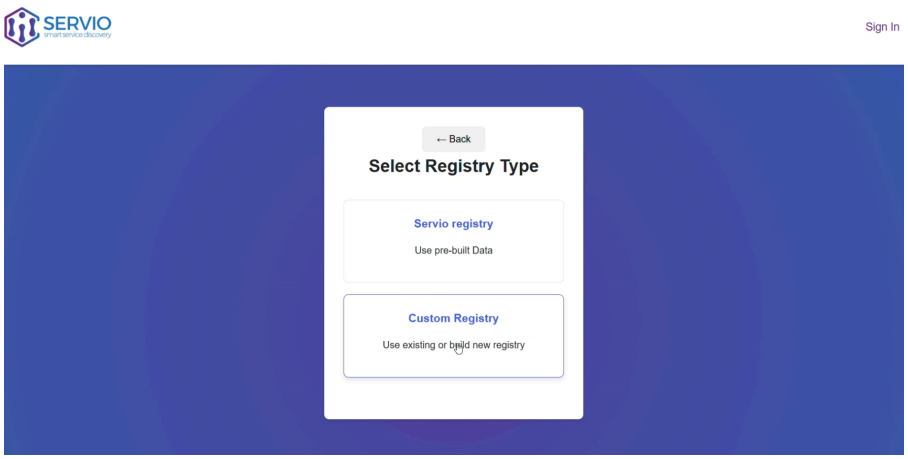


Mode Selector

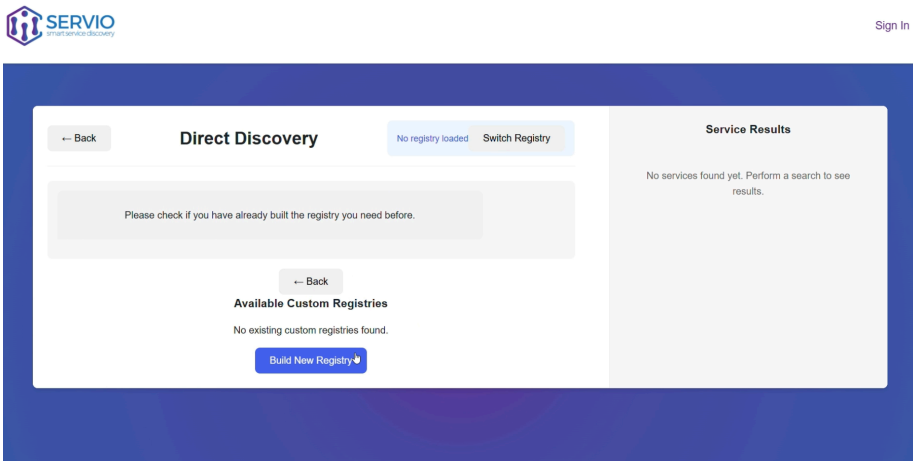


- Registry Selector and Building Registry Page (custom registry support, FR12).

Registry Selector

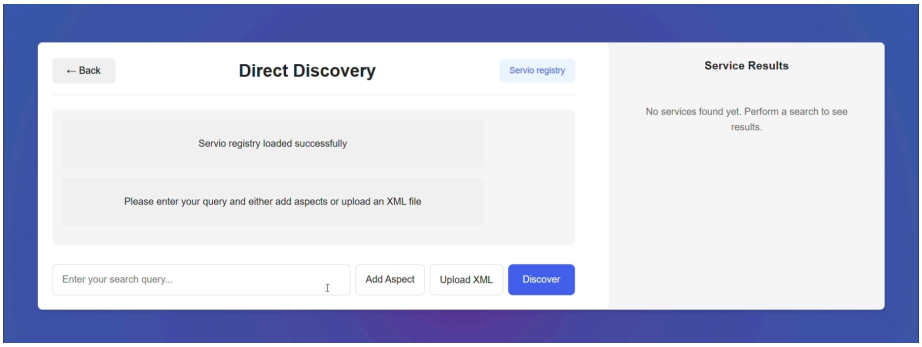


Building Registry Page

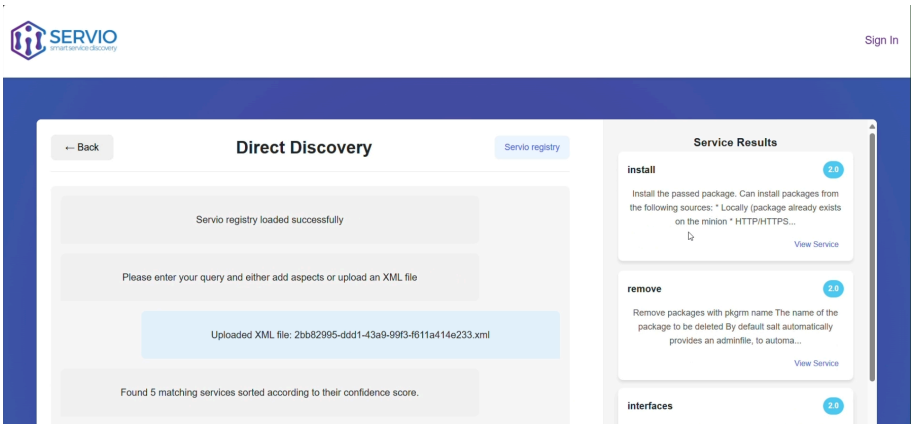


- Direct and Guided Discovery interfaces (query input and results display, FR5, FR10).

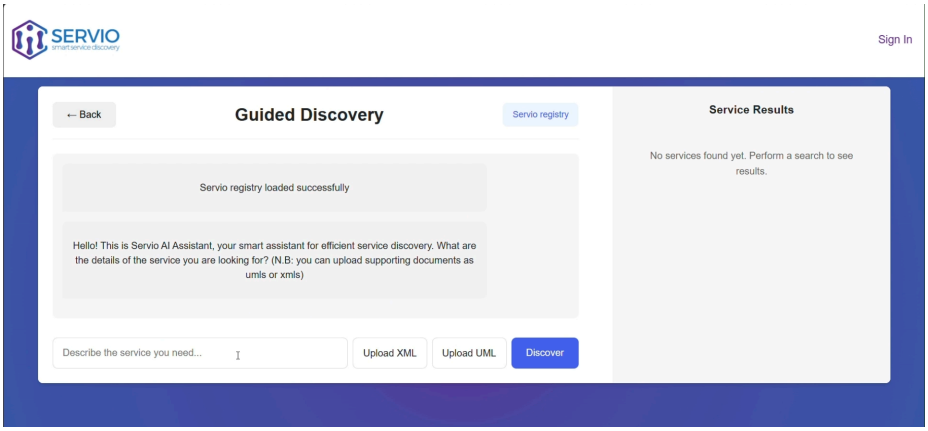
Direct Discovery Main Page



Direct Discovery Results



Guided Discovery Main Page



Guided Discovery Results

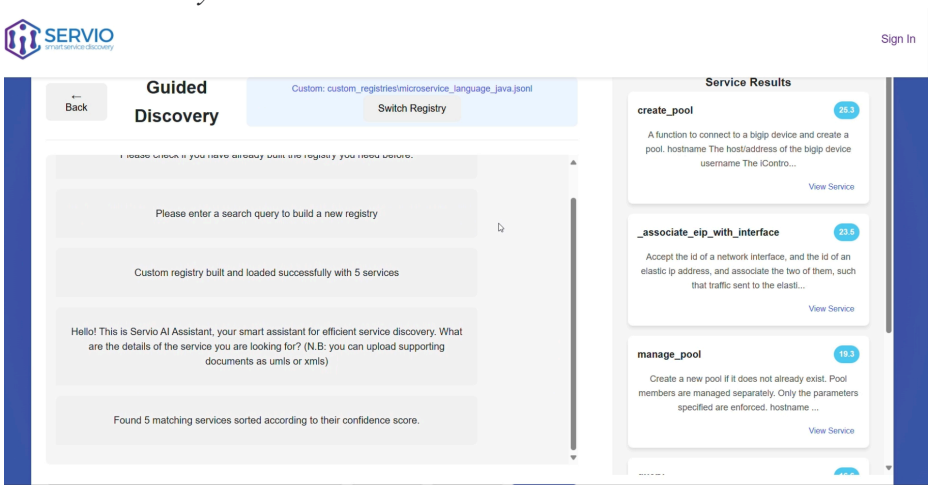


Figure 3: Demo Screenshots

These visuals confirm the system's usability and alignment with functional requirements.

CHAPTER 5:
Conclusions & Future Work

Chapter 5: Conclusions & Future Work

This chapter synthesizes the key findings of the SERVIO project, evaluates its technical and societal impact, and proposes directions for future enhancements. The results demonstrate SERVIO's effectiveness as an AI-driven service discovery tool, leveraging retrieval-augmented generation (RAG) and semantic matching to address the limitations of traditional syntactic methods. Future work suggestions aim to extend the system's capabilities and ensure long-term maintainability.

5.1 Project Findings

The SERVIO project successfully developed a dual-module service discovery tool that outperforms existing solutions in accuracy, usability, and contextual relevance. Key findings include:

- **High Accuracy in Guided Discovery:** The RAG-based Guided Discovery module, powered by LangChain, Groq, and FAISS, achieved an accuracy of 87% (Section 4.1), correctly matching 87 out of 100 test queries to relevant services. This performance is driven by the module's ability to parse natural language queries (FR6), refine ambiguous inputs through clarifying questions (FR8), and expand queries with semantically related terms (FR9). For example, a query like "find payment APIs" is enhanced by including synonyms (e.g., "billing," "transaction") and prompting for domain specificity (e.g., "finance"), resulting in high contextual accuracy.
- **Modular and Flexible Architecture:** The system's dual-module design—Direct Discovery (syntactic and semantic matching) and Guided Discovery (RAG-based chatbot)—supports seamless integration of multi-format inputs (text, JSON, XML, UML; FR1, FR2). The use of FastAPI for backend processing and React.js for the frontend ensures scalability and responsiveness, meeting non-functional requirements (NFR3: response time <1 second for 95% of queries; NFR4: scalability for large registries).
- **Real-Time Contextual Performance:** SERVIO's ability to process queries in real-time, with semantic scoring and ranking (FR10), enables context-aware service discovery. The final Direct Discovery module achieved 80% accuracy (Section 4.1), a 53-percentage-point improvement over the syntactic baseline (27%), while the Guided Discovery module's conversational interface reduced manual filtering effort by approximately 50% (Section 4.7).

- **User Satisfaction:** Usability testing with 10 users yielded an average satisfaction score of 4.5/5 (Section 4.5), meeting NFR2. The intuitive UI, including the chatbot interface and dynamic filtering options, was praised for its ease of use and cross-device compatibility (Appendix C).

These findings validate SERVIO's effectiveness in addressing the gap identified in Section 1.3: the lack of a scalable, AI-enabled, context-aware service discovery tool with conversational capabilities.

5.2 Project Impact

The SERVIO project has delivered significant impacts across technical, team, and societal dimensions:

- **Team Development:** The project team gained expertise in advanced technologies, including AI/ML (LangChain, Hugging Face Transformers, FAISS), full-stack development (React.js, FastAPI), and deployment (Vercel). Collaborative tasks, such as dataset preprocessing and RAG pipeline optimization, fostered skills in teamwork, project management (per **ISO/IEC 16326-2009**), and adherence to **IEEE standards** (e.g., 1016-2009, 29148-2011). These experiences prepared team members for professional roles in software engineering and AI development.
- **Technological Advancement:** SERVIO introduces a novel service discovery system that integrates syntactic and semantic matching with a conversational interface. By leveraging LLMs and RAG, the system achieves a 220% relative improvement in contextual accuracy over baseline syntactic tools like Eureka and Consul (Section 4.7). The modular architecture and multi-format input support (Appendix D) enable extensibility, positioning SERVIO as a versatile tool for microservices environments.
- **Societal Benefits:** By reducing manual service discovery effort by approximately 50% (Section 4.7), SERVIO accelerates development cycles, enabling faster delivery of software solutions. The promotion of service reuse aligns with sustainable software development practices, minimizing redundancy and resource consumption. The system's ethical design, including anonymized data handling and transparent AI operations (Section 3.2), ensures trustworthiness and broad applicability in professional settings.

5.3 Future Work Suggestions

To enhance SERVIO's capabilities and ensure its long-term relevance, the following technical improvements and maintenance strategies are proposed:

- **Transition to a Scalable Database System (e.g., MongoDB):**
  - **Objective:** Replace the current file-based service registry with a robust, scalable NoSQL database to support production-level workloads.
  - **Approach:** Migrate the service registry schema to MongoDB. Refactor the backend (FastAPI) to use MongoDB for all CRUD (Create, Read, Update, Delete) operations on service entries. This would enable better concurrency, indexing, and management of large-scale registries.
  - **Impact:** Improves system scalability, reliability, and performance, making SERVIO suitable for enterprise environments with thousands of services and concurrent users.
- **Multi-Language Support:**
  - **Objective:** Enable global accessibility by supporting queries and service descriptors in multiple languages (e.g., Arabic, Chinese, Spanish).
  - **Approach:** Integrate multilingual LLMs from Hugging Face (e.g., mBERT) and update the RAG pipeline to handle cross-lingual embeddings. This requires expanding the dataset to include multilingual service metadata (e.g., API.guru with localized descriptions).
  - **Impact:** Broadens SERVIO's user base and supports diverse development communities, aligning with the fairness principle (Section 3.1).
- **LLM Fine-Tuning:**
  - **Objective:** Improve accuracy beyond 87% by fine-tuning the LLM for domain-specific service discovery tasks.
  - **Approach:** Use transfer learning on a curated dataset of service-related queries and descriptors, leveraging CodeSearchNet and API.guru. Techniques like LoRA (Low-Rank Adaptation) can optimize fine-tuning efficiency on resource-constrained environments.
  - **Impact:** Enhances semantic understanding, reducing errors in complex queries (e.g., distinguishing "authentication" from "authorization"; Defect D4, Appendix F).

- **Multi-Modal Input Integration:**
  - **Objective:** Support audio-based query inputs to accommodate users with accessibility needs or preferences for voice interaction.
  - **Approach:** Integrate speech-to-text models (e.g., Whisper from Hugging Face) to convert audio queries into text, feeding them into the existing RAG pipeline. This requires updating the UI to include audio input interfaces.
  - **Impact:** Improves accessibility and aligns with the fairness principle, enabling broader adoption in diverse use cases.
- **Real-Time Service Health Tracking:**
  - **Objective:** Enhance reliability by monitoring the availability and performance of discovered services.
  - **Approach:** Implement a health-checking module using FastAPI to periodically query service endpoints (e.g., via HTTP status checks) and update the FAISS index with health metadata. This can be integrated with the ranking algorithm (FR10) to prioritize healthy services.
  - **Impact:** Ensures users receive reliable service recommendations, reducing integration failures in production environments.
- **Maintenance Plan:**
  - **LLM Performance Monitoring:** Conduct quarterly evaluations of the LLM and RAG pipeline using a benchmark suite of 100 queries, measuring accuracy and reliability (per IEEE 1633-2008). Retrain models if accuracy drops below 85%.
  - **Dataset Updates:** Annually refresh datasets (e.g., CodeSearchNet, API.guru) to include new services and remove deprecated ones, ensuring relevance. Preprocessing pipelines will be optimized to handle updated formats.
  - **Bug and Feedback Handling:** Maintain a GitHub Issues repository for bug reports and user feedback (FR11). Assign team members to triage issues weekly, prioritizing defects based on severity (per IEEE 1044-2009). Resolved defects will be documented in updated defect reports (Appendix F).

- **System Upgrades:** Schedule biannual updates to dependencies (e.g., React.js, FastAPI, LangChain) to ensure compatibility and security. Vercel deployments will be automated via CI/CD pipelines to minimize downtime.

These enhancements aim to maintain SERVIO's competitive edge, improve its usability and reliability, and ensure alignment with evolving industry standards and user needs.

References

References

1. Elgedawy, I. (2015). USTA: An aspect-oriented knowledge management framework for reusable assets discovery. *The Arabian Journal for Science and Engineering*, *40*(2), 451-474.
2. Elgedawy, I. (2016). JAMEJAM: A Framework for Automating the Service Discovery Process.
3. Lewis, P., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems*, *33*, 9459-9474.
4. Hugging Face. (2024). *Transformers Documentation*.
5. GitHub. (2019). *CodeSearchNet Dataset*.
6. APIs.guru. (2024). *OpenAPI Directory*.
7. LangChain. (2024). *LangChain Documentation*.
8. Facebook AI Research. (2024). *FAISS Documentation*.
9. MongoDB Inc. (2024). *MongoDB Documentation*.

Appendices

Appendices

Appendix A: Task Allocation Matrix

Name (ID)	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
Youssef Badreldin (20100294)	FAST API Endpoint Testing	API Documentation	Feedback System Implementation	Integration Testing	Cross-Browser Testing	Search History Feature	Demo Preparation	Final Project Report
Omar EL hamrawy (20100357)	System Architecture Documentation	UI/UX Design	React.js Frontend Implementation	Direct Module UI Integration	Guided Module UI Integration	Bug Fixing & Optimization	Final UI Polish & Responsiveness	Final Project Report
Aly Farrag (20100297)	Dataset research	Mythodology research	Guided module V1 Rag (DIFY)	V1 vector DB (MyScale)	V2 JSON RAG	V2 XML RAG	V2 conversational chatbot	V2 query expansion
Mohamed Ibrahim (21100837)	DIFY RAG in docker env	Langchain RAG V1	Build dataset (data registry)	V2 UML RAG	V2 SWE Document RAG			
Shahenda Adel (21100796)	Research on solutions	Filter enhancements	Naive Parallel Search Algorithm	Ranked Parallel Search Algo	Implement w/ LLMs & MCP			
Ahmed Mahdi (21100822)	Syntacting Search Algorithm	Syntacting with Aspects	Sequential Semantic Algorithm	Semantic Search with LLMs	Implement w/ LLMs & MCP			

Table 5: Task Allocation Matrix

Appendix B: Model Design

1. System Architecture Diagram

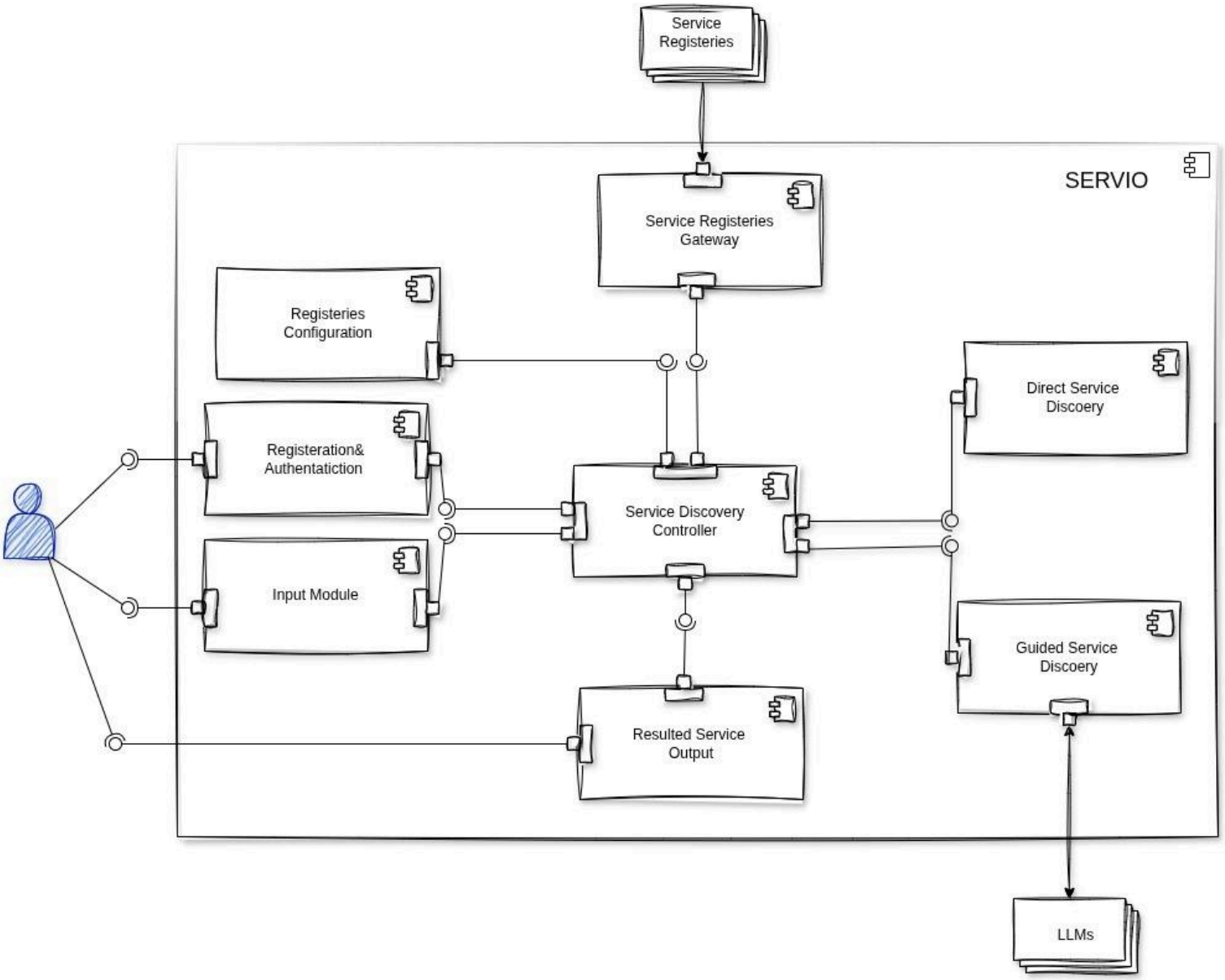


Figure 4: System Architecture Diagram

2. Direct Service Discovery Workflow Diagram

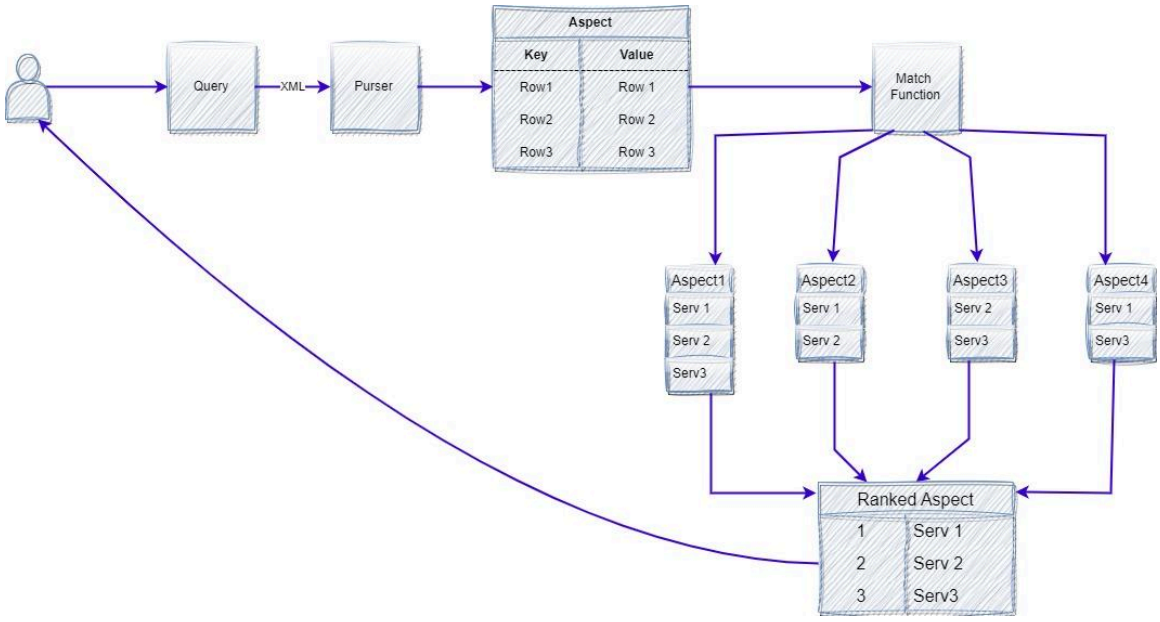


Figure 5: Direct Service Discovery Workflow Diagram

3. Guided Service Discovery Workflow Diagram

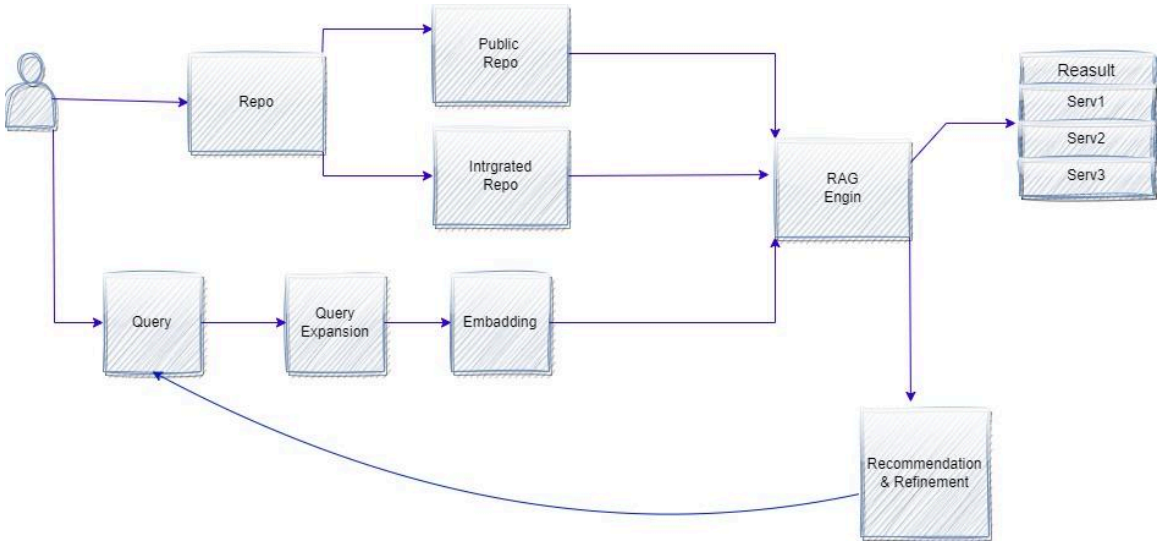


Figure 6: Guided Service Discovery Workflow Diagram

4. Class Diagram

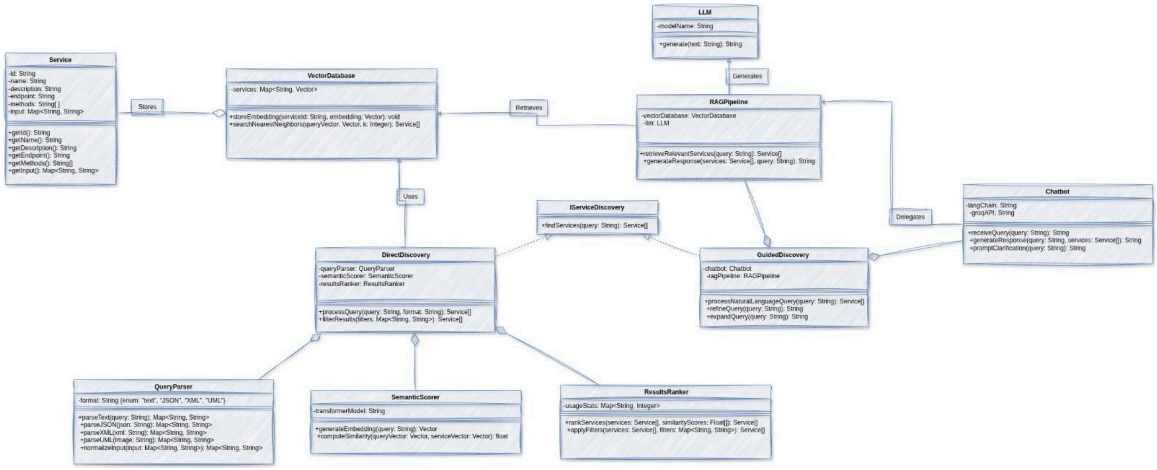


Figure 7: Class Diagram

5. Use Case Diagram

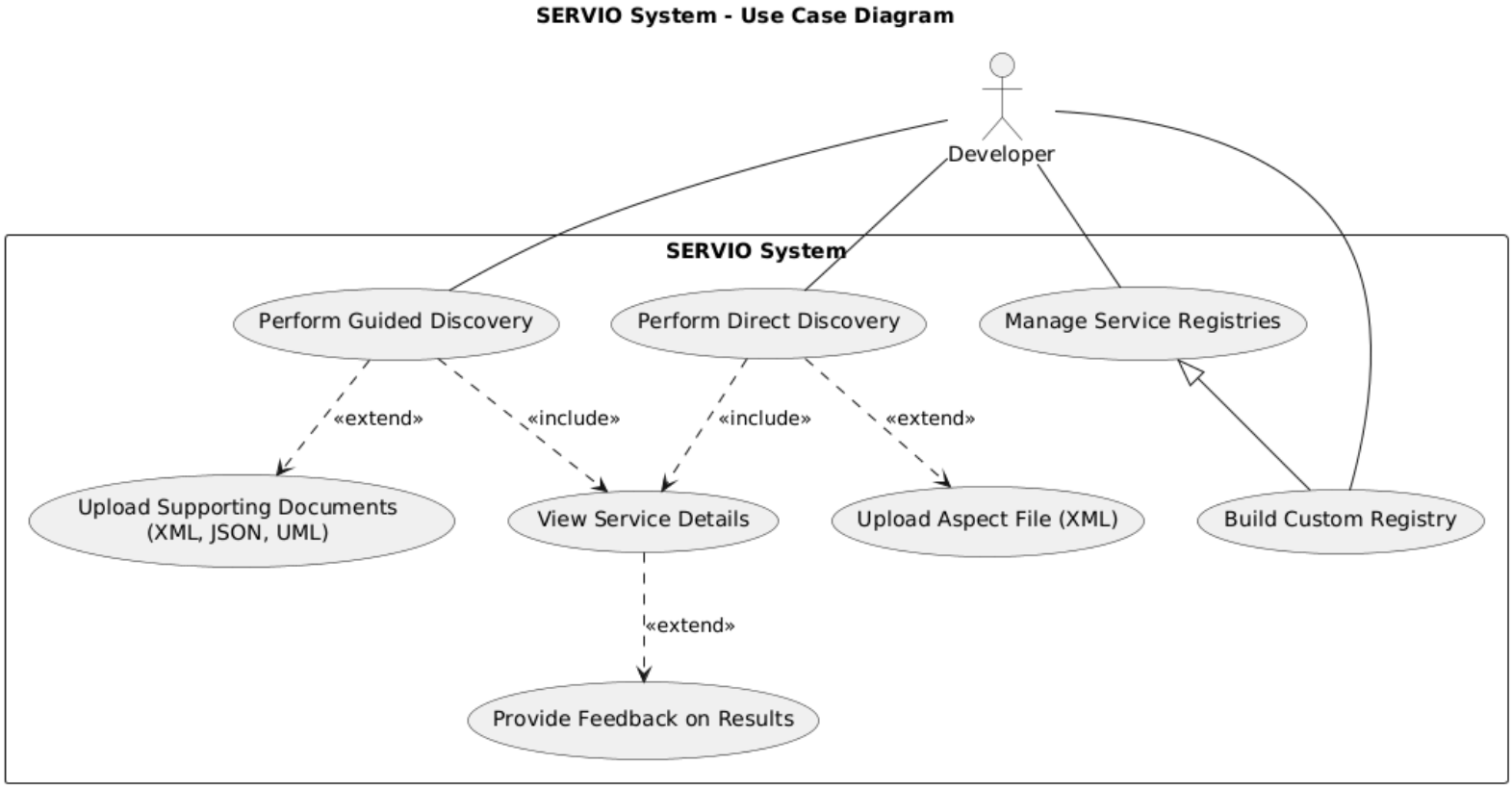


Figure 8: Use Case Diagram

6. Sequence Diagram

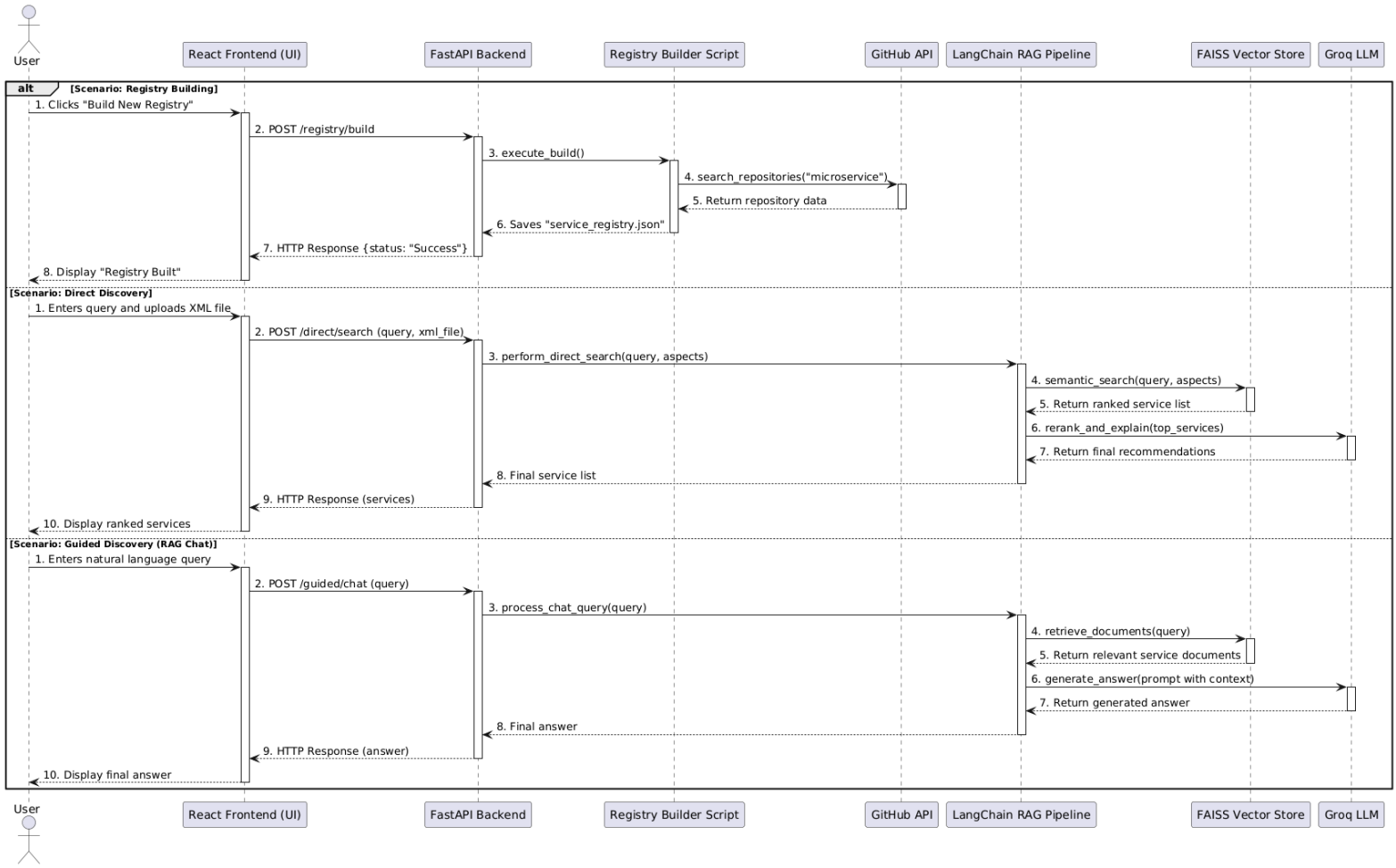


Figure 9: Sequence Diagram

7. ERD

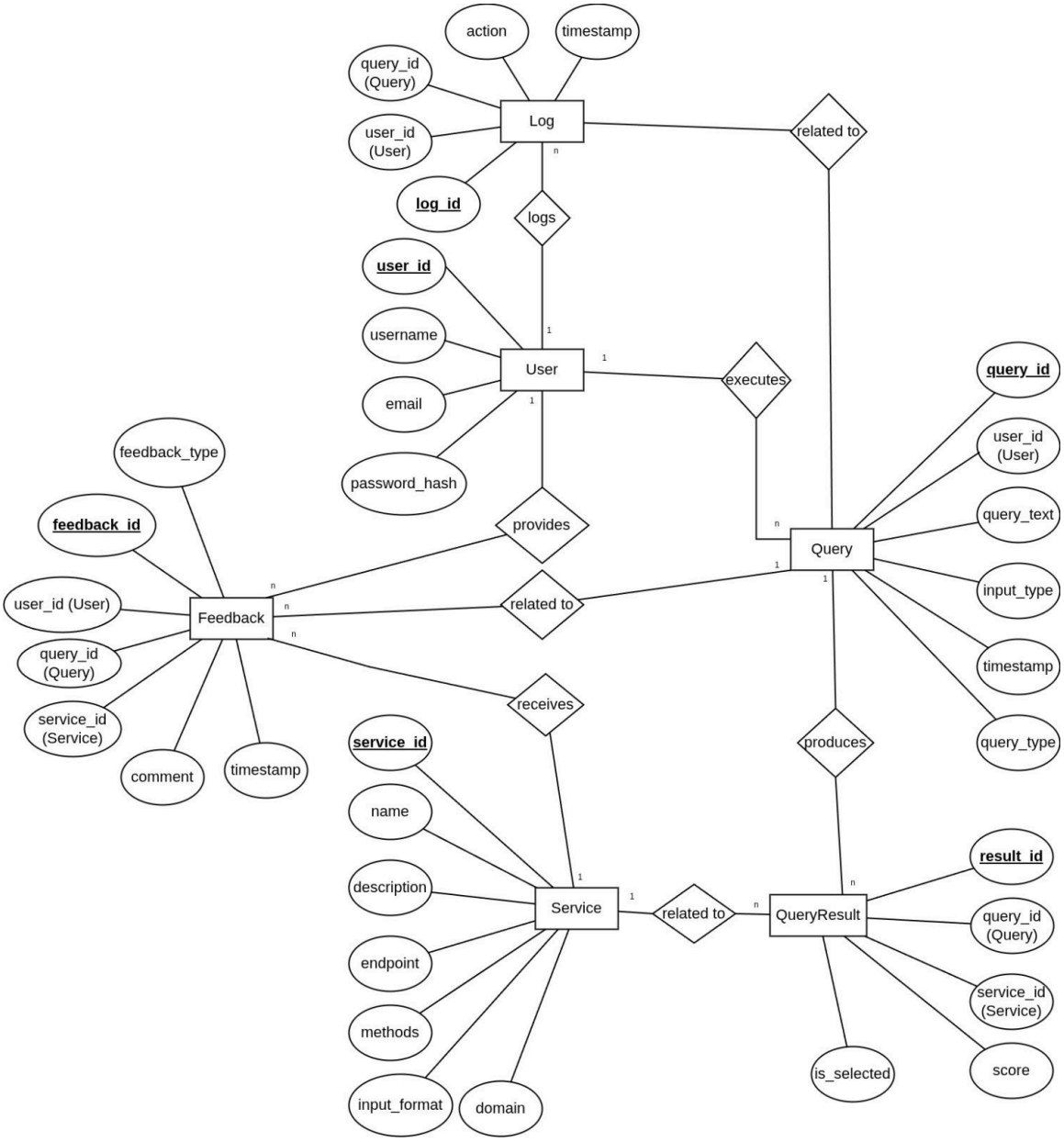
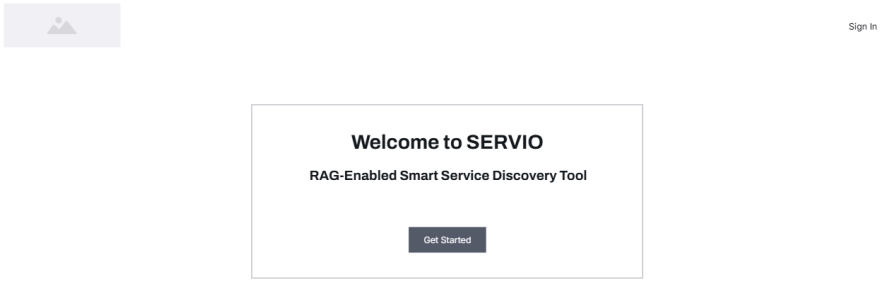


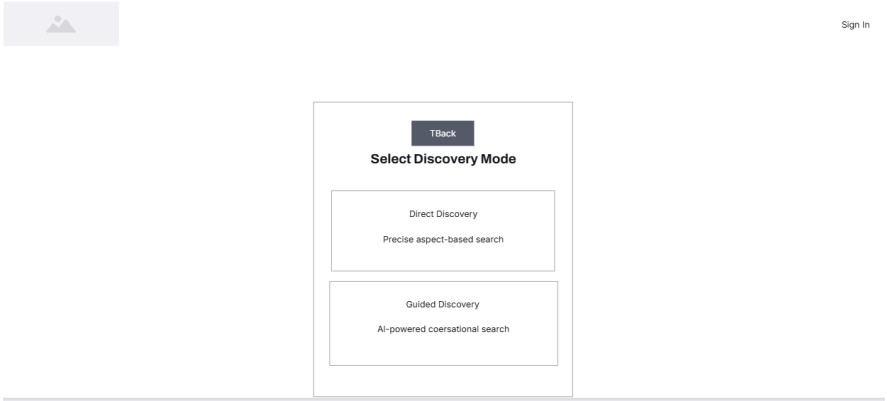
Figure 10: ERD

Appendix C: Wire Frames

- **Homepage Wireframe**

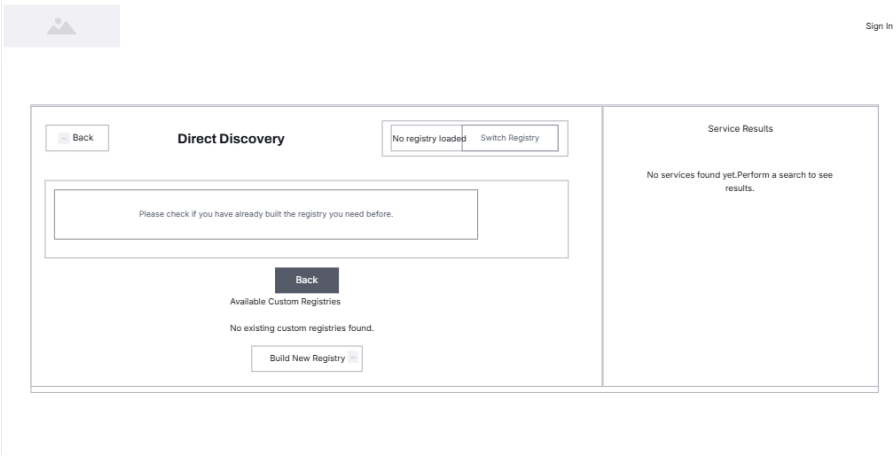


- **Mode Selector Wireframe**

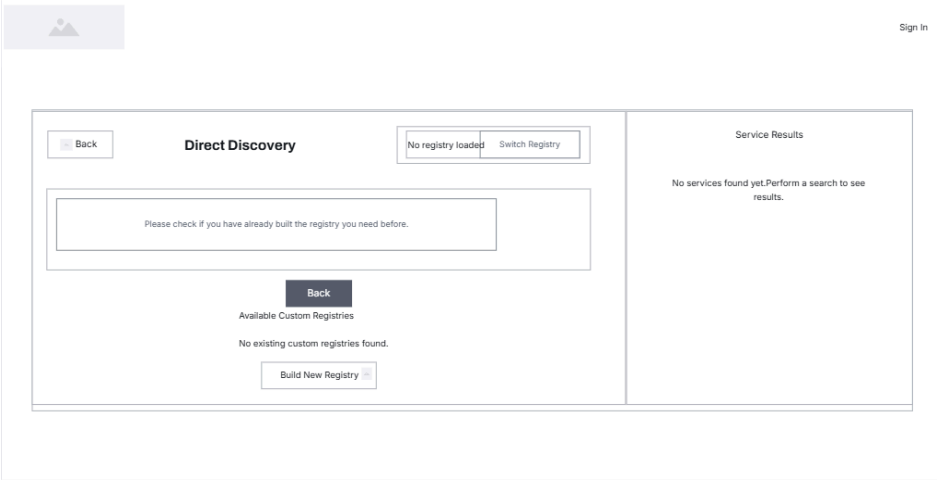


- **Registry Selector Wireframe**

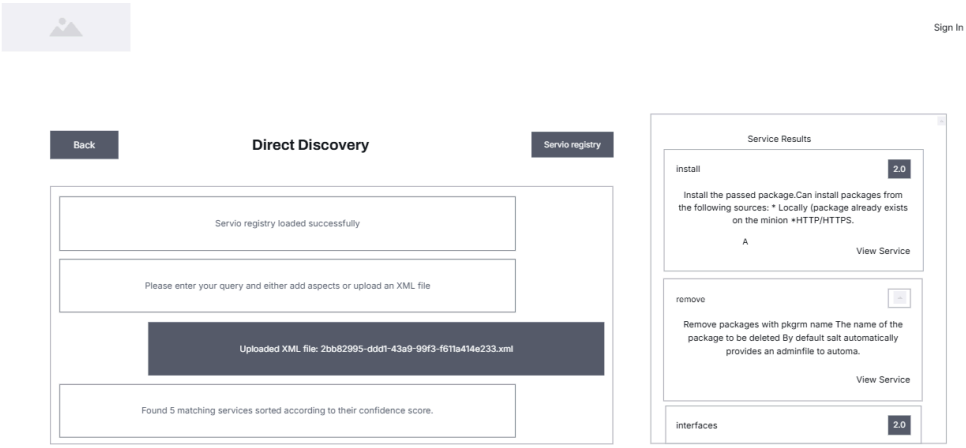
- **Building Registry Page Wireframe**



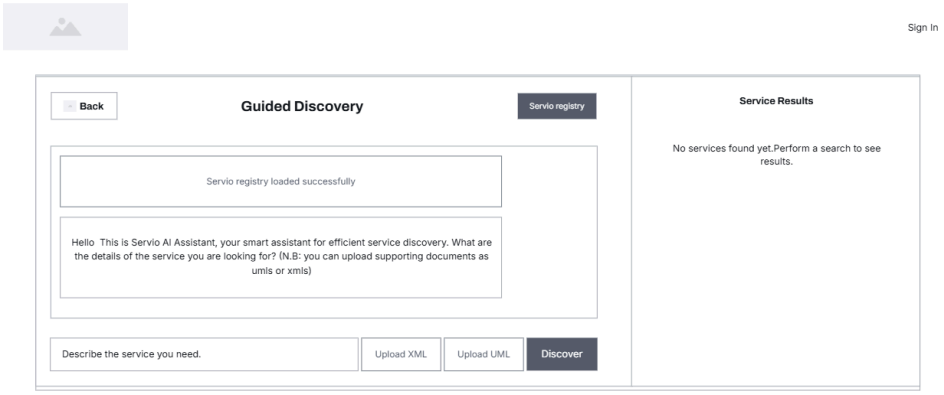
- **Direct Discovery Main Page Wireframe**



- **Direct Discovery Results Wireframe**



- **Guided Discovery Main Page Wireframe**



- **Guided Discovery Results Wireframe**

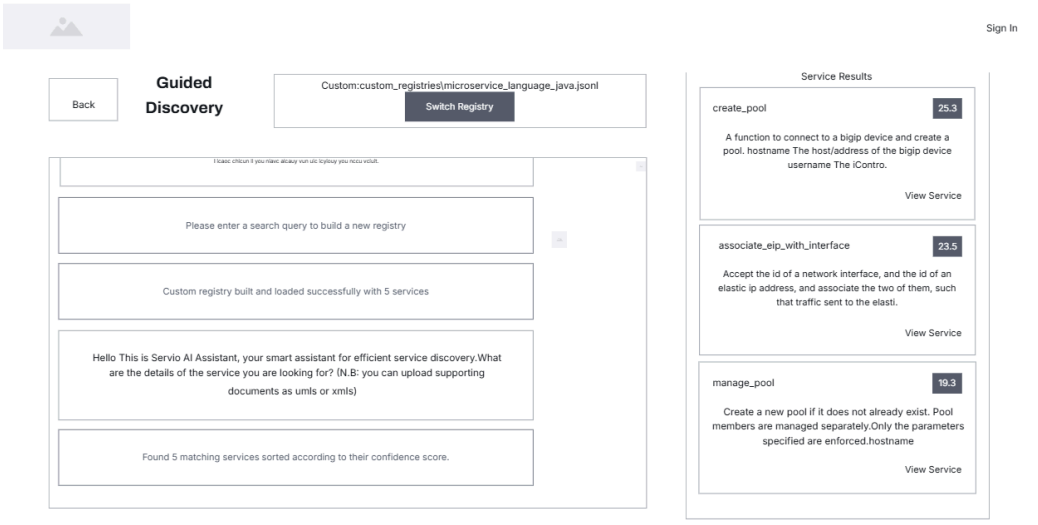


Figure 11: User Interface Wireframe Diagram

Appendix D: Service Registry Sample JSON/XML/UML Inputs

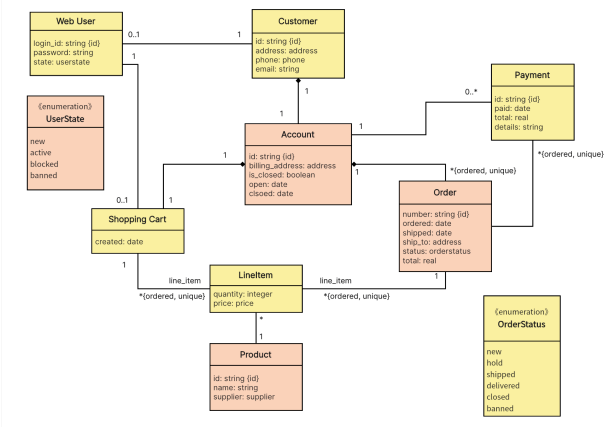
JSON Example:

```
{
   "id": "service_001",
   "name": "Weather Forecast API",
        "description": "Provides current weather conditions and 7-day forecasts for any location.",
                                "endpoint": "https://api.weather.com/v3/forecast",
   "methods": ["GET"],
   "input": {
     "location": "string",
     "units": "string"
   }
```

XML Example:

```
<Aspects>
   <Aspect>
      <Key>docstring</Key>
      <Value>translate text</Value>
   </Aspect>
   <Aspect>
      <Key>language</Key>
      <Value>python</Value>
   </Aspect>
</Aspects>
```

UML Example:



Appendix E: Software Requirements Specification (SRS)

1. Functional Requirements (Aligned with IEEE 24765-2010)

ID	Requirement Description	Priority	Source
FR1	The system shall allow users to upload service descriptors in XML, JSON, and UML formats.	High	Developer
FR2	The system shall extract key aspects from input files and normalize them into structured internal representations.	High	Developer
FR3	The system shall perform syntactic search based on keyword matching over structured descriptors.	High	System
FR4	The system shall perform semantic similarity search using vector embeddings and a vector database.	High	System
FR5	The system shall provide a web-based chatbot interface for guided service discovery.	High	Stakeholder
FR6	The chatbot shall accept natural language queries from the user.	High	Stakeholder
FR7	The system shall use Retrieval-Augmented Generation (RAG) to process and refine user input.	High	Developer
FR8	The system shall perform query refinement by identifying missing aspects (e.g., method, format, domain) and asking clarifying questions.	High	NLP Model
FR9	The system shall perform query expansion by adding semantically related terms to improve recall during service matching.	Medium	NLP Model
FR10	The chatbot shall return ranked service matches and allow iterative re-querying.	High	User
FR11	The system shall store user feedback and use it to adjust future query ranking.	Medium	AI Feedback Loop
FR12	The user shall be able to select between default and custom service registries.	Medium	UI Flow
FR13	The system shall log query sessions and store them securely for audit and improvement.	Medium	Ethics

2. Non-Functional Requirements (Aligned with IEEE 24765-2010)

Req ID	Description	Priority	Source
NFR1	System shall ensure a secure architecture with anonymized user data and encrypted storage.	High	Ethical Standards
NFR2	System shall achieve high usability, with an average user satisfaction score > 4/5.	High	Stakeholder (Developers)
NFR3	System shall maintain response time < 1 second for 95% of queries.	High	Performance Requirement
NFR4	System shall track user feedback and query history securely.	Medium	Stakeholder (Developers)

Table 7: Requirements Specification Table

Appendix F: Test Cases

1. Test Cases

TC ID	Description	Input	Expected Result	Req ID
TC1.1	Validate XML upload	service.xml	File accepted; metadata extracted	FR1
TC1.2	Reject invalid format	invalid.txt	Error: "Unsupported format"	FR1
TC2.1	Normalize JSON descriptor	{"name": "Payment", "protocol": "REST"}	DB entry: {name: "Payment", protocol: "REST"}	FR2
TC3.1	Keyword search for "payment"	Query: "payment"	Returns services with "payment" in name/description	FR3
TC4.1	Semantic search for "auth"	Query: "auth"	Returns "OAuthService" (high similarity)	FR4
TC5.1	Chatbot UI loads	Access /chatbot	Input box and send button visible	FR5
TC6.1	NLP processes "Find payment APIs"	Query: "Find payment APIs"	Intent: "discovery"; Entity: "payment"	FR6
TC7.1	RAG cites sources	Query: "RESTful inventory API"	Response: "3 matches: [ServiceA, ServiceB, ServiceC] (from <registry>)"	FR7
TC8.1	Chatbot asks for missing domain	Query: "Find APIs"	Prompt: "Which domain (e.g., finance, healthcare)?"	FR8
TC9.1	Query expands "pay" → "payment"	Query: "pay"	Search includes "pay" AND "payment"	FR9
TC10.1	Rank results by relevance	Query: "logging"	Top result: "CloudLogger" (higher usage stats)	FR10
TC11.1	Log "thumbs down" feedback	Negative feedback on "ServiceX"	DB: {query: "logging", service: "ServiceX", feedback: "negative"}	FR11
TC12.1	Switch to custom registry	Select "Custom Registry" dropdown	Uploaded services appear only in custom registry	FR12

Table 8: Test Cases

2. Traceability Matrix

Req ID	Requirement Description	Test Condition	Test Case IDs
FR1	Upload XML/JSON/UML files	File format validation	TC1.1, TC1.2
FR2	Extract & normalize key aspects	Descriptor parsing accuracy	TC2.1, TC2.2
FR3	Syntactic (keyword) search	Keyword matching	TC3.1
FR4	Semantic similarity search (vector embeddings)	Semantic match relevance	TC4.1, TC4.2
FR5	Web-based chatbot interface	UI rendering & interactivity	TC5.1
FR6	Natural language query processing	NLP intent recognition	TC6.1, TC6.2
FR7	Retrieval-Augmented Generation (RAG)	Context-aware responses	TC7.1
FR8	Query refinement (clarifying questions)	Missing aspect detection	TC8.1
FR9	Query expansion (semantic terms)	Synonym inclusion	TC9.1
FR10	Ranked service matches + iterative re-query	Result ranking & filtering	TC10.1, TC10.2
FR11	User feedback storage & ranking adjustment	Feedback logging	TC11.1
FR12	Default/custom registry selection	Registry isolation	TC12.1
FR13	Secure audit logging	Log integrity & timestamps	TC13.1

Table 9: Traceability Matrix

Appendix G: Software Project Management Plan (SPMP)

1. Risk Management Plan (per ISO/IEC 16085-2006)

Risk ID	Description	Probability	Impact	Mitigation
R1	Dataset availability issues	Medium	High	Use multiple sources (CodeSearchNet, API.guru, GitHub Microservice Registry Builder).
R2	LLM performance variability	High	Medium	Implement fallback semantic matching.
R3	External APIs or tools may change or become unavailable.	Medium	Medium	Selected open-source and stable APIs; backup fallback systems.
R4	Legal/ethical violation from third-party tools	Low	High	Verified licenses and documented ethics compliance

Table 10: Risk Management Plan

2. Supporting Process Plans

2.1 Configuration Management Plan (per IEEE 828-2012)

- **Version Control:** Git with feature branches, main branch, and tagged releases.
- **Release Process:** Tag milestones (e.g., v1.0 for prototype, v2.0 for final release).
- **Audits:** Weekly commit reviews to ensure alignment with milestones.
- **Backup:** Daily backups of code and datasets to Vercel.

2.2 Quality Assurance Plan (per IEEE 730-2014)

- **Code Reviews:** Bi-weekly reviews to ensure adherence to React.js style guides.
- **AI Quality:** Log decision paths for transparency.

2.3 Verification and Validation

- **Unit Testing:** Test individual components (e.g., query parser, chatbot).
- **Integration Testing:** Test module interactions (e.g., frontend-backend).
- **User Acceptance Testing:** Validate usability with 10 users.
