Frankfurt University of Applied Sciences

Faculty of Computer Science and Engineering

Implementation and Evaluation of an Enterprise Architect Chatbot
Using a RAG-Based Approach

Thesis to Obtain the Academic Degree

Master of Science (M.Sc.)

Submitted by

Hendrik Gruber

Matriculation Number: 1458240

Advisor	:	Prof. Dr. Jürgen Jung
Co-Advisor	:	Dr. Rainer Schlör

DECLARATION (ERKLÄRUNG)

I hereby assure that I wrote the present work independently and that I did not use any other sources than those given in the bibliography.

All passages that are taken verbatim or correspondingly from published or not yet published sources are marked as such.

The drawings or images in this work were created by myself or provided with a corresponding source reference.

This work has not been submitted to any other examination authority in the same or a similar form.

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Frankfurt a.M., 16. April, 2026

Hendrik Gruber

ABSTRACT

Lorem ipsum ...

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AI	Artificial Intelligence
APOC	Awesome Procedures on Cypher
AR	Action Research
EA	Enterprise Architect / Architecture
EAM	Enterprise Architecture Management
GenAI	Generative Artificial Intelligence
IT	Information Technology
MCP	Model Context Protocol
PAR	Participatory Action Research
RAG	Retrieval-Augmented Generation
UI	User Interface
UML	Unified Modeling Language

USAGE OF GENERATIVE AI

OpenAI GPT-5 and GPT-5.2 (OpenAI, 2025) was used in order to aid in the creation of this thesis as described below.
OpenAI GPT-5 and GPT-5.2 (OpenAI, 2025) were used in order to find sources and summarize them.
OpenAI GPT-5 and GPT-5.2 (OpenAI, 2025) were used in order to build both transpilers and the node.js backend server.
OpenAI GPT-5 and GPT-5.2 (OpenAI, 2025) were used in order to assist in the writing process. Textual information was not generated with the assistance of AI. Rather, it was used to proof read and give the writing an academic style.

Part I

THESIS

1 INTRODUCTION

1.1 MOTIVATION AND THESIS QUESTION

In the field of define that we are building a RAG system which will be referred to as a chatbot. it is not an agent, as it is not executing anything.

1.2 RESEARCH METHOD

Action Research was applied in order to gain scientific value out of the developed prototype. The advantage of this research method is that it is very supportive of the development process for information systems. According to Baskerville (1999), all types of Action Research have the following four characteristics in common: An orientation towards developing, a focus on a specific problem, an iterative process, and a collaboration amongst participants. This is applicable to the work at hand because a prototype is being developed for a specific problem. The development cycle is conducted iteratively and collaboratively with different stakeholders. More details on this are described in chapter **??** [3]

2 TERMINOLOGY AND TECHNOLOGY

This chapter goes in detail on the terminology and technology that will be relevant for the reader to have a foundational understanding of the rest of this thesis. Later chapters will build upon these concepts and pieces of technology.

2.1 TERMINOLOGY

2.1.1 *Enterprise Architecture Management*

Enterprise Architecture Management (EAM) can be summarized as being the bridge between the business and IT departments of an enterprise. The goal is to implement information technology that is aligned with the business needs of the company. This is in contrast to the IT department implementing information technology for the sake of implementing information technology, which people in IT are often fond of doing [1]. An unwanted situation would then be when the IT department falls into a siloed way of thinking where they are decoupled from the rest of the company. EAM helps to ensure that the implemented information technology is achieving the right things, namely supporting the business capabilities and processes. [13, pg. 2-3]

Enterprise Architecture can benefit a company in various ways.

Diese Quelle hierfür vielleicht verwenden: [12] Die Quelle hat vor allem eine Menge Fragen aus der echten welt, was EAs sich im alltag fragen. diese liste kann als referenz dienen für meinen copilot. Diese Quelle hilft da vielleicht auch nochmal um in die Tiefe zu gehen. [4]

2.1.2 *Enterprise Architect*

A common challenge for an EA is dealing with the heterogenous nature of an application landscape [13, pg. 6]

Erwähnen, dass eine Herausforderung des EAs es ist, dass die vorzunehmenden Änderungen zwar von einem high-level POV einfach aussehen, aber in den details viele herausforderungen stecken. zB stakeholder management (jede Applikation hat einen eigenen verantwortlichen, viele schnittstellen der applikationen, etc.)

2.1.3 *Business Capabilities*

2.1.4 *Architecture Diagrams*

2.1.4.1 *Application Landscape*

2.1.4.2 *EAM Modeling Tool - Archi*

2.2 TECHNOLOGY

2.2.1 *Large Language Models*

LLMs are capable of supporting in language-related tasks where text needs to be generated, translated, summarized, analysed, or questions answered [10].

[24] describes what llms are and why they are not good with domain specific information and how that causes them to hallucinate.

2.2.2 *Graph Database*

2.2.3 *Retrieval Augmented Generation*

2.2.4 *neo4j and the Cypher Query Language*

2.2.5 *Model Context Protocol*

Also explain (maybe in the SOTA) how it is able to understand the schema of the graphdatabase (by calling get-schema when initializing the MCP it knows how my graph database is setup).

2.2.6 *XML and Transpilers*

As will be shown later in section xyz, the working prototype uses a domain-specific, source-to-source transpiler to turn the archimate-exported XML data into Cypher queries to insert the EA data into the graph database.

This source has a bit of good information on why XML is good as a structure language[23] "Simply speaking, eXtensible Markup Language (XML) is a data architecture connecting meta-data and data. The architecture's defining feature is the hierar-chical network of nodes. Every node in the XML structure is connected somehow to any other node; also, being a hierarchy, every node is either subordinate or su-perordinate to another node, as shown in the tree structure in Figure 1. Further, XML "provides a standard syntax for the mark-up of data and documents" (Watt 2002:1). The syntax along with the hierarchical network structure make XML doc-uments exhaustively searchable and therefore useful for linguistic research."

More detailed information on the transpiler is described in section xyz.

2.2.7 *APOC and XPATH*

Describe how apoc + xpath work and what this has to do with the neo4j cyphers. this might be better in the methodology section as a quick paragraph, but it has to be described somewhere.

3 CURRENT STATE OF THE ART

Briefly explain why a literature analysis is important. Define the scope (what fields you looked at, which databases, what keywords). Define the research method and how you narrowed it down from x sources to y sources.i

3.1 ENTERPRISE ARCHITECTURE MANAGEMENT

theories, digital twin efforts, EA tool landscapeStandards or frameworks (e.g., TOGAF, ArchiMate, IATA ONE Record, LeanIX). Theoretical foundations (auch auf prozessmanagment eingehen, wie der aktuelle Prozess aussieht, wenn die Landschaft geändert werden soll) Current tools and methods Research prototypes in EA

Authors Jung and Fraunholz 2021 [13] lay foundational work from which many EAM concepts can be derived.

3.2 LARGE LANGUAGE MODELS, CONVERSATIONAL AGENTS, AND RETRIEVAL-AUGEMENTED GENERATION

strengths, hallucination issues, graph-RAG enhancements Theoretical foundations Current tools and methods

This paper covers how ai tools are more scalable than manual expertise analysis of things. The source is highly relevant. Look at the summary in notebookLM. 05.10.25 [9]

This 2025 paper has ideas on how changing knowledge graphs (e.g. through updates) can be handled [14]. It looks at temporal data and how to handle it. This might be relevant since addressing how a changing application landscape can be handled will probably be a challenge.

This paper gives an overview on how to control the dialog sequence and also notes 4 types of dialog options for chatbots in the related works section: [15].

This paper [29] covers how a chatbot can support in task-planning and output generation. Might be helpful in understanding how my chatbot can tell the EA how to conduct changes in the application landscape.

This paper [6] states how proactive dialogue systems work and can be improved. It goes into detail on 3 types of dialogue systems: clarification, target-guided, and non-collaborative dialogues. All 3 of these have a certain relevance for the EAM Chatbot.

RAG: Geh darauf ein, was es für unterschiedliche Chunking methoden gibt, wie man ein Buch runterbringt, und was das alles für vor und nachteile hat. auch welche tools es gibt (neo4j) ist wichtig.

3.3 TEXT-TO-CYPHER GENERATION

Text-to-cypher: this source is OP [25]. it goes into detail on why text to cypher is beneficial, how it is done, how to prompt the system, and how they used a 3 step system to preprocess the text, map the preprocessed text to the entities in the db, generate the cypher based on all previous information and context.

3.4 COMPARABLE PROJECTS AND PROTOTYPES

Proof-of-concepts, research prototypes, industry whitepapers, GitHub projects.

Tools like ChatEA, LeanIX AI features, or Microsoft Copilot integrations in architecture/governance.

A prototypical graph-based RAG approach for text-summarization has been created by Microsoft: y[7]. The accompanying paper is here: [8]

Dragon1 needs to be detailed here!

3.5 EVALUATIONS AND LIMITATIONS

Studies analyzing strengths/weaknesses of RAG, embedding quality, hallucination mitigation.

Papers about user interaction with EA tools, chatbot evaluation frameworks, usability challenges

This paper [26] gives a standardized method and framework for evaluating coversational AI agents.

This paper [16] proposes a benchmark for open-ended multi-turn conversational agents. I think this paper focuses more on evaluating agents and comparing their results, but maybe i can copy their evaluation methods and benchmarks?

4 METHODOLOGY

This chapter describes how the chatbot “Masutā” (pronounced “mah-soo-taa” - a phonetic adaptation of the English term "master" into Japanese katakana) was developed. It provides further insight into the applied research method and the process through which the prototype was created. This gives the reader the understanding of how the research and implementation was conducted before discussing the implementation details in chapter 5.

4.1 ACTION RESEARCH DESIGN

Action research is a research method which is highly applicable when developing an information system such as the one presented in this paper [3]. The advantage of Action Research is that a large focus can be laid on the development of a system while still achieving an academic benefit. This allows for a very explorative approach to developing an information system.

Todo

Cyclical phases are central to the concept of Action Research. Baskerville [3] describes Action Research as an iterative process consisting of five steps within a single cycle. Other sources, such as Cornish et al. [5], propose variations with fewer (usually three) phases within a cycle; however these models also boil down to the same concepts. Across the literature, Action Research cycles follow the same structure: planning what should be done in the new cycle, taking action, and evaluating the outcome of the completed cycle before moving on to the next one [3, 5].

The paper at hand applied a cycle using the following five steps according to Baskerville [3]: diagnosing, action planning, action taking, evaluating, and specifying learning. The reason this research applies five cycles instead of three is the benefit of describing the development process in more detail. This five-step-cycle including the preliminary and subsequent steps built the basis of the conducted Action Research..

Each cycle lasted between three and four weeks. The sources used do not mention how long a single cycle should last. However, a timeframe of two to three weeks for each cycle was deemed as a reasonable for the development of Masutā because status updates were held at the end of each cycle and with the given amount of time for the cycle, there was enough progress to discuss in these status update meetings.

Action Research typically leaves the main theorizing up to the researcher. However, an extended form of Action Research named Participatory Action Research goes a step further in creating a more collaborative environment between the researcher and client participants. Instead of leaving the theorizing up to the researcher, new information and ideas are thought up together with the other participants, giving both parties an active role. This is beneficial because the other participants often have both theoretical and practical knowledge of the subject matter being worked on. [3] From here on out, the term Participatory Action Research will be used interchangeably with Action Research.

The following subsections explain the research design of the applied Action Research. For the exhaustive Action Research protocol, refer to chapter A of the Appendix.

4.2 ACTION RESEARCH SETUP

The domain of EAM was focused on within this research. In particular, this study addressed mostly application landscapes, business capability maps, as well as the relationships between these two architectural artifacts. These artifacts are commonly used to support documenting an enterprise's landscapes and are used to align an enterprise's IT with its strategic business objectives.

Due to their structural complexity with heterogenous data sources and multiple stakeholders involved, these artifacts are often large and difficult to interpret. Maintaining an overview can be especially challenging for junior level enterprise architects. This challenge motivates the exploration of AI-supported solutions that enable conversational interaction with the architecture, rather than manually navigating the complex diagrams.

The research was conducted in close collaboration with the academic supervisor and the co-advisor. The academic supervisor gave academic guidance and supported the structuring of the research process to ensure academic relevance. The co-advisor acted as the domain expert and practitioner, contributing practical insights to ground the research with real-world relevance. The author of this thesis assumed the role of the researcher, implementing the Action Research cycles, validating findings, and planning subsequent steps in coordination with the other stakeholders.

At the outset of the research, the general problem space was clear, but the potential solution was only vaguely defined. The co-advisor initialized the research with a vague vision of a centralized system containing all enterprise architecture information, which can be interacted with via natural language. The motivation for this was to reduce the effort required to interpret the enterprise architecture artifacts.

However, in the early stages of the project, not only were the technical details of the potential solution unclear, but also the feasibility of such a system. Early on it was mutually agreed upon that LLMs would play a key role in realizing this, even though the data structures, mechanisms, storage options, and interaction patterns were still open. Consequently, the initial problem statement was intentionally formulated at a high level to provide a suitable starting point for iterative exploration. Through the iterative development cycles, this high level problem statement without a planned technical concept was refined into a concrete problem definition and technical solution.

4.3 ACTION RESEARCH CYLCES

The above mentioned vague details of the implementation became clear during the development cycles, leading to a final architecture with a clear structure. Each cycle began with a meeting between all three stakeholders. The end of one cycle was the incubator for the next cycle and was conducted in the same meeting. Table 4.1 summarizes these cycles.

Todo: note in the cycles where i attemtped implementation x, which didn't work out because of reason y. any path taken should be noted. also, note at what state the system as a whole was at the end of each cycle. you have this in the learnings for cycle 4 that it was able to answer all 3 categories of questions. do something similar for each cycle.

4.3.1 *Cycle 1*

Diagnosis: The initial problem statement lacked concrete technical formulation and the feasibility of natural language interaction with enterprise architecture data was unclear.

The co-advisor outlined typical enterprise architecture workflows and the tools used to document and interact with architectural artifacts in practice. An initial idea was propsed of an AI-based black-box system capable of ingesting architecture data and deriving its own internal representations. The achievability as well as academic applicability of such a black-box system were critically questioned, particularly because of the limited transparency of the innern mechanisms and how to test this. As an alternative, the academic supervisor proposed an explicit knowledge graph approach in which a knowledge graph is built and used to supplement LLM-generated answers.

The key distinction between the black-box and white-box approaches is the transparency. While the black-box approach autonomously creates an internal representation of the information, the white-box approach requires the manual design and implementation of an explicit technical architecture.

These discussions were necessary in order to scope the solution space. Early visions of an end goals included a chatbot that would support enterprise architects in exploring and improving application landscapes, for example by identifying inconsistencies or incomplete application landscapes.

At this stage, the research methodology had not yet been explicitly defined as Action Research, and it was initially assumed that the resulting prototype would be evaluated through an expert-interview.

Action Planning: The first cycle aimed to develop a proof of concept that enables conversational access to a knowledge graph consisting of enterprise architecture knowledge grounded in textbook-based domain information. It was decided that a single-agent architecture will be used, as multi-agent architectures were considered unnecessarily complex for an initial proof of concept.

Action Taken: An initial knowledge graph was constructed based on content from the textbook *Masterclass Enterprise Architecture Management* [13]. The textual content was iteratively preprocessed and transformed into graph representations, with successive refinements applied to improve the mapping of domain concepts into nodes and relationships. This mapping was implemented using an LLM to parse the file and add it to the knowledge graph. The parser was individually fit to each file. After integrating the full textbook into the knowledge graph, additional prototypical data was incorporated in the form of an application landscape and business capability map to enable querying the knowledge graph of concrete architectural data. Each of these files also received a customized LLM-based parser.

The querying method was implemented via hard-coded cyphers in the frontend. The user's input prompt was then run through an LLM which attempted to fit the input into the predefined cyphers.

Evaluation: The state of the proof of concept after the first cycle was demonstrated to both advisors and evaluated qualitatively through open discussions. Both advisors positively assessed the feasibility of the approach, and the co-advisor confirmed that an explicit knowledge graph-based solution will be a viable direction for further development.

Learning: Compared to the beginning of the cycle, in which the feasibility of a centralized knowledge graph was uncertain, the first cycle demonstrated that an explicit, white-box approach represents a practical path forward. The cycle also revealed key challenges in two critical layers of the application. These layers relate to importing heterogeneous data and transforming it into graph structures and also the effective querying of such representations. A challenge was parsing data via an LLM because this meant giving full control to the LLM and thus being dependent on the capabilities of the language model.

Finally, it became evident that further iterations would be required to systematically address these challenges with an exploratory development process.

4.3.2 *Cycle 2*

Diagnosis: Following the initial feasibility assessment, the next identified challenge concerned extending the knowledge graph with more information and positioning the prototype as a useful tool within the realm of enterprise architecture management.

One design consideration involved separating textbook-based domain knowledge from enterprise architecture data into two distinct databases. After discussion, a single integrated database was chosen to enable direct relationships between conceptual textbook knowledge and architectural data, with the expectation of improved contextual reasoning.

The challenge imposed by lack of real-world data to test the system was also diagnosed. Potential test-datasets for further development and evaluation were discussed in order to bridge the gap before real-world data would be ready. The real-world data would require more time to be prepared because the co-advisor's company policy constraints meant the data cannot be used directly and has to be sanitzed first. Instead, it was agreed upon to use data from a completed university assignment, consisting of an application landscape, business capability map, business capability support matrix, business object model, and cross-application data-flow diagram for a fictitious company named SpeedParcel.

Action Planning: The objective of the second cycle was to extend the knowledge graph with additional data from the SpeedParcel dataset. This included integrating the business capability support matrix on top of the already existent application landscape and business capability map. A second goal of this cycle was to improve the database querying method as soon as more data is available to test with.

Action Taken: The capability support matrix of the SpeedParcel dataset was integrated into the knowledge graph, requiring adjustments to existing query mechanisms. A large challenge that hindered advancements in development came up. The hard-coded cyphers that were being used to query the database were reaching their limit. A viable solution was not found to hard-code cyphers in such a way that the changing dataset can be dynamically queried.

The idea of a Model Context Protocol (MCP) server came up during development. This brought the advantage of a standardized interface between the application and the knowlede-graph. Implementing the MCP server helped achieve a higher quality in the retrieved answers, as the prompt was being transformed into custom cyphers during runtime. This allowed the cyphers being used to query the database to no longer be hard-coded in the frontend, but allowed them to be generated dynamically based on the user input. This text-to-cypher generation is achieved by using an LLM to transform the user's prompt directly into cyphers, as apposed to having predefined cyphers that get populated with the user's input by the LLM. A prerequisite for this to work, however, was that the schema of the knowledge graph was known. At this stage, the schema was hard-coded into the frontend.

Evaluation: The extended prototype was again evaluated qualitatively through a demonstration by the researcher and open discussions between all three stakeholders. Both advisors expressed strong interest in the approach and assured that the chatboat was being developed in the correct direction in order to achieve the end goal of allowing enterprise architects to interact with the knowledge graph via natural language.

Learning: As the prototype is starting to mature, more learnings are being pulled from each phase. Particularly, the hard-coded query method that was previously implemented was proved to be insufficient for flexible interaction with the evolving graph structure. The MCP implementation allows the entire system to be more dynamic, independent of the data in the knowledge graph.

Furthermore, three key categories of user questions were identified. The first category being conceptual questions related to enterprise architecture principles, concepts, and best practices found in textbooks. The second category being descriptive questions targeting concrete architectural elements and relationships. The third category being integrative questions combining conceptual knowledge with specific architectural contexts. Examples of these categories can be found in section **todo**.

Beyond the scope of this thesis, a broader vision by the advisors emerged in which the prototype could support project planning by assessing impacts on application interfaces, systems, and stakeholders. The co-advisor noted that the developments achieved thus far provide a starting point for exploring such directions in future work.

4.3.3 *Cycle 3*

Diagnosis: During this phase it became apparent that Action Research will be the most appropriate methodological framework for the project. The development process had evolved into an exploratory and iterative approach. This has been characterized by continuous development, reflection between all three stakeholders, and decision-making regarding which direction to take the prototype in. Consequently, the previously considered expert interview was no longer the methodology of choice for this thesis.

In parallel, the co-advisor began modelling real-world enterprise architecture data using the Archi tool. It was agreed upon that the real-world data would later be exported from Archi and imported into the knowledge graph in XML format. This meant that the format of the real-world data is finalized and can be prepared for in the prototype, allowing the integration of more realistic data into the finished prototype.

Additionally, it was decided that a final meeting will be conducted at the end of the fourth cycle. This meeting would require that the prototype be finished and be executable in a local environment. This meeting will also mark the end of the development phase of the prototype.

Action Planning: The objective of the third cycle was to further expand the information saved in the knowledge graph by creating additional datasets from the SpeedParcel dataset. The datasets existed in a third party architecture modelling tool and had to be replicated in Archi in order to simulate the real-world data which will later also be exported from Archi. Modelling the data in Archi allows it to be exported into XML format and imported into the knowledge graph via an XML parser. This step aimed to prepare the system for handling the more complex and structured architectural models expected later.

Action Taken: Both the business object model and the cross-application data-flow diagram were recreated in Archi, in order to test how exported XML files will behave when parsing them into the knowledge graph. The parsing at this step was still being conducted via a custom, per-file LLM-based parser. This cycle mostly dealt with expanding the knowledge graph and fine tuning the system.

Evaluation: The progress of the third cycle was reviewed with both advisors during open discussions. The current state of the prototype and the extended data integration were assessed positively, and the overall research trajectory was again confirmed as appropriate. The end goal is in sight and is being worked towards in an appropriate manner.

The system had trouble generating qualitative answers for the newly imported dataset. This is due to the new data having a different schema than the other datasets.

Learning: The third cycle revealed scaling issues. While the individual per-file parsers have worked well so far, this cycle showed that its limits have been reached. Parsing the more complex files during this cycle showed that LLMs are no longer the appropriate method moving forward. On top of this, this custom, per-file parsing approach would also not allow for parsing files without manual implementation specific for the file at hand; a problem that has to be solved before the final meeting where files will have to be parsed on site and on the fly.

Another scalability issue revealed was a discussion about the real-world data. While SpeedParcel's data contained hundreds of nodes and edges, the real-world data will potentially have thousands of nodes and edges. This gap revealed that a better parsing solution and a better querying solution will be necessary before the prototype is ready for the final on site meeting. This will have to be tested before the on site meeting to ensure that this does not cause performance issues within the system.

4.3.4 *Cycle 4*

Diagnosis: The fourth and last cycle diagnoses that the system in its current state relies too heavily on LLMs for core precessing tasks. Importing the XML data into the knowledge graph is a critical point of the application and relying on an LLM to handle this parsing is not a good choice because XML is a standardized format and many parsers exist to handle XML. It was agreed upon that a generalized parser should be used to parse the incoming XML files into the knowledge graph, replacing the LLM at this critical point within the system's architecture.

A second critical part of the system that was diagnosed to be insufficient was the querying method to retrieve the information from the knowledge graph. Although the cyphers are being LLM-generated custom to the user's input, it is having trouble dealing with the changing schema of the knowledge graph because a hard-coded schema is written into the frontend which it uses as a reference for what to query. With an evolving knowledge graph, this approach is no longer viable because it is guessing at the graph's structure and not able to retrieve the data this way.

Action Planning: The goal of this cycle was to finish the prototype and have it be ready for the on site meeting at the end of the cycle where all three stakeholders will be testing and discussing the system together. Achieving this goal means that the XML data must be parsed via a generalized parser, replacing the LLM implementation. On top of that, the strategy of how to handle the schema must be refactored. It will be required to dynamically understand the data, rather than have one schema hard-coded. Lastly, it also meant setting up a containerized local environment to allow running the prototype on the advisor's devices during the on site meeting.

Action Taken: The LLM-based data parser was replaced by an out of the box solution from Awesome Procedures On Cypher (APOC), which contains functions to parse file types and also how to better query a knowledge graph. Here, APOC takes the input XML files and parses them directly into cyphers. This allows any type of Archi-exported XML file to be saved into the knowledge graph.

The schema-handling strategy was updated from the hard-coded variant. Before passing the user-prompt and system context to the LLM, a call to the knowledge graph was made via neo4j's built-in call `db.schema.visualization()` [20] in order to get the graph's schema information. While this did improve the generated results, it still proved to be insufficient, as the LLM was not able to handle this schema information well enough to generate reliable responses.

This was also solved via APOC. Rather than using neo4j's built-in schema call, the schema retrieval was changed to a predefined APOC function `apoc.meta.schema()` [21]. This ensures that the LLM understands how the knowledge graph is structured and how it may query it for the information requested by the user's prompt.

The local environment is set up via Docker containers and can be run on any machine. It comes equipped with a frontend, backend, and two databases. The first database contains all data pertaining to SpeedParcel. The second database is a playground database and is only pre-filled with textbook information. This ensures that the on site meeting will allow the application to work.

Lastly, cosmetic changes to the UI were added as well as new features to support the on site meeting. For example, the user may now upload XML files via the web application as well as reset the playground database, and view the knowledge graph in an interactive way.

Evaluation:

Learning: Many learnings can be taken from this last cycle. The first being that a generalized parser to read XML files into the knowledge graph improves the quality of the import. Another added benefit of this is that the import is now deterministic, while before the LLM-based parser behaved in a non-deterministic way.

The second main learning from this cycle arose from refactoring the way that the queries get generated. Previously, the cyphers were straightforward to generate, as the schema of the SpeedParcel data was entirely known. However, during this cycle, new data was imported into the knowledge graph, which had an unknown structure, meaning the knowldge-graph's schema was unknown. The transformation of text to query has been improved by making a preliminary call to the database to fetch the current database schema. This schema information is then passed as context to the LLM in order to write more accurate cyphers. The results with this new querying strategy meant that for the first time the system was able to answer all three categories of questions.

With the conclusion of this cycle, Masutā was a finished prototype and ready to be tested during the on site meeting.

Table 4.1: Overview of Action Research Cycles

Cycle	Diagnosis	Action Planning	Action Taken	Evaluation	Learning / Outcome
Cycle 1	Technical feasibility.	Develop a proof of concept.	Built initial knowledge graph with textbook data. Hard-coded cyphers.	Advisors confirmed feasibility and viability of a knowledge graph approach.	White-box approach validated. Challenges identified in data import and querying.
Cycle 2	Need to extend data and improve queries.	Integrate SpeedParcel dataset and improve querying.	Added SpeedParcel data. Replace hard-coded cyphers with MCP-based text-to-cypher.	Advisors confirmed correct direction of development.	Dynamic querying identified as essential. Three categories of questions defined.
Cycle 3	Action Research identified as appropriate methodology. Scalability and data format issues emerged.	Expand datasets via Archi models to prepare for XML imports	Recreated SpeedParcel models in Archi. Parsed XML.	Progress and trajectory assessed positively. Schema-related answer quality issues observed.	Scalability limits of LLM-based parsing identified. Need for generalized parsing recognized.
Cycle 4	Over-reliance on LLMs for parsing and schema handling diagnosed as critical limiatation.	Replace LLM parsing with generalized XML parser. Refactor schema handling. Prepare executable prototype.	Implemented APOC-based XML parsing and schema retrieval. Containerized local setup. UI enhancements added.	todo	Deterministic parsing and dynamic schema handling achieved. System able to answer all three question categorizes. Prototype completed.

It is important to note that during each cycle, continuous system tests were being run in order to ensure that the system's requirements were being met. For example, when adding new data to the knowledge graph, the system was prompted to test if it is able to retrieve this new data. In most cases, it was not able to do so right after adding new data. These tests led to the system's prompt having to be continuously be fine tuned in order to be able to handle the data in the knowledge graph. This was a routine step conducted regularly during each iteration of development.

4.4 FINAL MEETING

Todo: den Termin vor Ort beschreiben Purpose of the meeting, who participated, what was validated, what kind of feedback was collected.

Table 4.2: Overview of Action Research Cycles condensed into three parts per cycle

Cycle	Diagnosis	Action Taken	Learning / Outcome
Cycle 1	Technical feasibility.	Develop a proof of concept.	Built initial knowledge graph with textbook data. Hard-coded cyphers.
Cycle 2	Need to extend data and improve queries.	Integrate SpeedParcel dataset and improve querying.	Added SpeedParcel data. Replace hard-coded cyphers with MCP-based text-to-cypher.
Cycle 3	Action Research identified as appropriate methodology. Scalability and data format issues emerged.	Expand datasets via Archi models to prepare for XML imports	Recreated SpeedParcel models in Archi. Parsed XML.
Cycle 4	Over-reliance on LLMs for parsing and schema handling diagnosed as critical limiatation.	Replace LLM parsing with generalized XML parser. Refactor schema handling. Prepare executable prototype.	Implemented APOC-based XML parsing and schema retrieval. Containerized local setup. UI enhancements added.

As described in this chapter, it becomes clear why Action Research was an invaluable methodology. From unclear beginnings containing only a vague vision for a final prototype, each development cycle contributed to the final architecture being clear and goal oriented. Each phase helped to examine what was possible from a technical standpoint as well as how to move forward. This supported the explorative nature of the project.

5 IMPLEMENTATION

The methodology chapter answers the question of what was built and why. This chapter contains all information on what the finished Masutā prototype is, including its architecture and a sequence diagram, data processing mechanisms, and features.

5.1 FINISHED ARCHITECTURE AND PROTOTYPE

Figure 5.1 provides a holistic overview of the system architecture underlying the developed prototype. The user (1) starts the interaction by inputting their prompt through a web-based frontend implemented in React (2). The frontend serves as the primary interface for the user to interact with the application and its underlying data. Interaction is done by prompting via natural text. This makes up the presentation layer of the architecture.

Incoming requests are forwarded to the Node.js backend (3) which orecheStrates the interaction between the frontend the LLM (4), and the MCP Server (5). The LLM is used to turn the prompt into cyphers and vice versa to turn the cypher's result back into natural language. This is the only external service of the entire architecture. To enable standardized access to the neo4j datbases (6 and 7), the MCP server sits as an intermediary between the application logic and the databases. These components constitute the orchestration layer of the architecture.

The MCP server is connected to both neo4j databases. The first being the playground database (6) which contains textbook data and the user's custom data that they upload (8). The second database is the SpeedParcel database (7) which contains the textbook data and the SpeedParcel example data (9). The separation of the two databases allows the user to toggle between their own playground instance, containing their own data, and the SpeedParcel instance, containing example data that is ready to be prompted. The databases make up the data access layer of the architecture.

Each of these application components are explained in more detail in section 5.2. With the exception of the external service from OpenAI, each component has a Docker icon at the top left. This indicates that the application component is part of the containerization. Each component runs in its own container.

5.2 COMPONENT DETAILS

This section describes each component in more detail with the goal of clarifying why the individual components are implemented the way that they are.

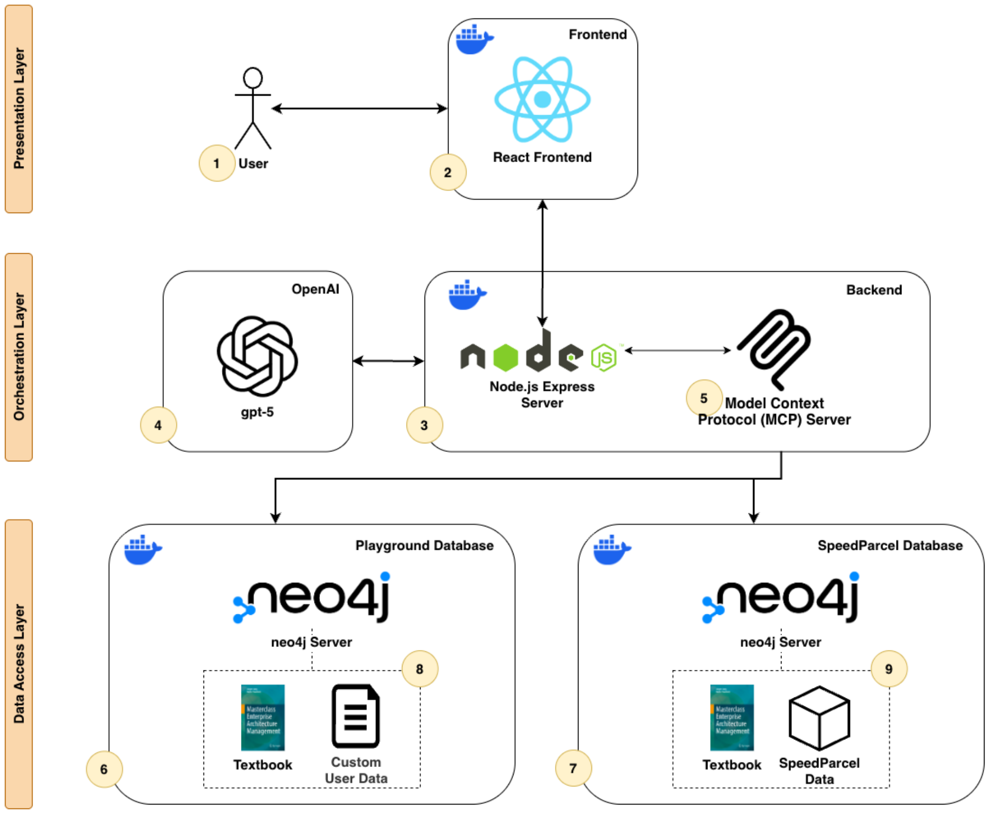

Figure 5.1: An architecture diagram showing each component and which components interact with one another. The Docker icon also indicates if the component is containerized.

5.2.1 *Frontend*

The frontend of the prototype is implemented as a single-page application using React with TypeScript. React was chosen due to its component-based architecture, which promotes modularity, reusability, and maintainability. TypeScript adds static typing to the JavaScript codebase, reducing runtime errors and improving development reliability in a growing application. For the user interface, the Chakra UI component library is implemented. Chakra UI provides a set of accessible, pre-styled components that significantly reduce the need for custom CSS while ensuring visual consistency across the application. This choice enabled rapid prototyping and allowed the development effort to focus on functionality rather than low-level UI styling concerns.

The frontend is containerized and exposed via `http://localhost:3000/`. It communicates with the Node.js backend through a set of RESTful HTTP endpoints. Each request contains the user's prompt as well as a session identifier, which is generated when the page is loaded or when the active database is toggled by the user. The session identifier is used by the backend to maintain chat state and conversation history. By regenerating the session identifier on page reloads and database switches, the system ensures a clear separation of chat contexts. This prevents unintended carry-over of conversation history between different databases or interaction sessions, thereby preserving data consistency and analytical validity.

Further details on the available frontend features and user interactions are provided in Appendix B.4.

5.2.2 *Backend*

The backend contains the entire application logic and is the most crucial part of the prototype. Here, requests are received from the frontend and processed before querying the data from the knowledge graph and subsequently returning it to the frontend. It also handles further logic, such as transpiling the user's uploaded XML files and inserting them into the knowledge graph and handles further features such resetting the playground database.

5.2.2.1 *Schema Information*

The ability for an LLM to perform a new task without explicitly retraining it is called in-context learning [27]. This is done via a meta-prompt that is supplied to the LLM where only a set of rules, inputs, or outputs are defined. The LLM then does the necessary reasoning based on this context in order to complete the new task. This is injected into the LLM at runtime [27]. In this case, when the LLM is tasked with generating a cypher for an unknown knowledge graph, the models uses the supplied context to infer the correct approach. In-context learning allows the LLM to learn how to generate the necessary cyphers on the fly. [25, 27]

Before the LLM is able to generate any meaningful cypher, it is necessary to get the knowledge graph's schema. The schema is a machine-readable summary of the graph structure containing information such as the node types and its relationships to other nodes. This is done by calling `apoc.meta.schema()` [21]. Directly after getting the schema, the backend also queries the indexes present in the knowledge graph. This enables the database to lookup nodes and relationships based on specific labels and properties [18].

With this information, the LLM will later understand how the knowledge graph is structured and how cyphers can be generated to query it.

5.2.2.2 *System Prompt*

The system prompt is one of the most crucial parts of the entire application. This is where the LLM is given its information for in-context learning in order to understand how to transform the natural language inputs into the cyphers and how to turn the raw cypher response back into natural language for the response to the user. The actual system prompts can be found in the chapter C.1 of the Appendix.

The incoming user prompt requires a finely tuned, well thought out system prompt. The goal of the first system prompt is to provide the LLM with all of the context that it needs in order to generate the specific cypher required to fulfil the user's prompt. It contains 4 sections, categorized to help generate the cypher. The first category in the system prompt is a set of rules on how it may generate the cypher, e.g. to infer meaning from the user's prompt to identify which labels in the knowledge graph to query. The second category is generic schema information and what type of cyphers it may use to fulfil different types of user requests. The third category in the system prompt is the task that the LLM is given, i.e. to generate exactly one cypher, only cypher, and no further text or explanations. The fourth and last category is a set of cypher syntax rules. These were added iteratively during development as the cyphers generated regularly had errors and required assistance to avoid.

This system prompt, along with the schema information, the user's current prompt, and the previous prompt and corresponding system answer are then passed to Open AI's gpt-5 LLM. The returned result is a single cypher which can be directly used to query the knowledge graph in order to fulfil the user's prompt.

The second system prompt supports in turning the cypher's response back into natural language. It is more straightforward and only contains two blocks. The first block tells the LLM to act as a senior enterprise architect supporting a junior enterprise architect. The second block gives the LLM more information on how to structure the response, e.g. to generally keep the answer within 1-6 sentences, depending on the complexity of the question.

This second system prompt, along with the original user prompt and the cypher's response are again passed to Open AI's gpt-5 LLM. The returned result is then a natural-language response containing the answer to the user's prompt. This is the final result that is returned to the frontend.

Tuning these system prompts greatly alters the quality of results of the entire system. While the second system prompt was rather straightforward, the first system prompt for incoming user prompts is the result of fine tuning it during development in all four action research cycles, as described in section 4.3. Any less information in this system prompt and the generated cypher's ability to retrieve the required information worsens.

This information is necessary for the LLM's ability to perform in-context learning specific to the knowledge graph's schema. Without this information, the LLM would not be able to generate cyphers based on the user's prompt, nor would it be able to return an answer in an appropriate wording designed for an enterprise architect.

5.2.2.3 *Ephemeral Conversation Memory*

The prototype has an ephemeral conversation memory, meaning that once the session concludes, the content of the conversation is discarded [28]. This is where the session ID, which is generated and sent from the frontend, comes into play. The backend logic holds an array of past user prompts and system answers. These are tied to the session ID. An incoming request's session ID from the frontend is compared to the current session ID stored in the backend. If the session IDs match, then the previous user prompt and system answer are used as further context for the new user prompt. This enables the user to build upon their previous prompt or the system's previous answer, which may be necessary during complex, multi-step interactions [28].

As the prototype stands, it saves only the most previous user prompt and corresponding system response. However, this can be scaled to contain more of the previous prompts and responses. It was kept to a minimum for this prototype as this can bloat the prompt passed to the LLM and quickly exceed Open AI's character limit, increase the amount of time required for it to generate a response, and cause confusion and irrelevant outputs [25].

5.2.2.4 *XML Transpiler*

Explain in detail here how the XML transpiler takes an XML file as the input and transpiles it into Cypher. it is model-to-model and thus touches on the subjects compiler construction, model-driven engineering, graph databases, and enterprise architecture tooling.

5.2.2.5 *Resetting the Database*

The backend contains an endpoint which allows the user to reset the playground database via interaction in the frontend. This reset can only be applied on the playground database, as the SpeedParcel database is read-only and is not to be modified. The playground database is reset in such a way, that only the user's uploaded data is removed and the textbook information is left remaining. This is done by deleting all nodes and relationships that are not explicitly part of the textbook data.

This covers all functionality in the backend. It is the central part of the prototype through which all data flows and all application logic is handled.

5.2.3 *Open AI*

The LLM used is made available by Open AI and can be accessed via its API [22]. The LLM has two tasks within the prototype. The first being to transform the user's prompt into cypher and secondly to turn the cypher's result back into natural language.

The LLM is an external component and is not hosted, trained, or fine tuned within the scope of this thesis. All interactions with the LLM were made via stateless API calls where the user's prompt, system prompt, schema, and conversation history were explicitly provided with each request. This allows the LLM to be decoupled from any application logic and can be replaced by other models.

5.2.4 *MCP Server*

The Model Context Protocol (MCP) is an open source, standardized communication protocol to allow client applications to communicate with AI applications. The advantage of having an MCP server sitting between the backend and the neo4j database is the standardized interface with external systems. [2, 11]

The MCP server employed in this prototype is the official neo4j MCP server. It communicates with the Node server via MCP's STDIO and comes with standard functions to read and write data to the neo4j databases. [19] This allows the prototype's backend to query the databases without requiring proprietary code.

5.2.5 *Neo4j Databases*

The need for two databases arose during development, as described in the Action Research in section 4.3. Before real-world data was made available, the system required data during development and testing. Because the SpeedParcel data was readily available from a university course during a previous semester and already resembled a real-world enterprise architecture, it was used as a starting point for the prototype.

Both databases contain textbook information from the 2021 book "Masterclass Enterprise Architecture Management" by Jung and Bardo [13]. The knowledge graph consisting of the textbook contains 13.724 nodes and 13.525 edges. The advantage of having a knowledge graph of the textbook information adjacent to the enterprise data is that user prompts of category 1 or 3 can directly query the textbook information as well in order to supplement the generated response with information specific to the role of an enterprise architect.

The SpeedParcel database contains further example data and serves as a starting point for the user to explore the prototype and its capabilities. SpeedParcel's knowledge graph consists of 566 nodes and 788 edges. Within this knowledge graph are SpeedParcel's application landscape, business capability map, business capability support matrix, business object model, and cross-application data-flow diagram. This data was read-in via the method described in section 5.2.7. The SpeedParcel database is entirely read-only.

Out of the box, the playground database only contains the knowledge graph of the textbook information. It is up to the user to fill the database with their own enterprise data. This database can also be reset by the user from the UI, allowing them to start over.

Both databases can be interactively viewed either via the application "neo4j Desktop" or via their web application under . After connecting to the respective database, the knowledge graph can be queried via custom cyphers and the results viewed in a visual and interactive way. Utilizing this tool along with the "copy cypher" feature described in the Appendix's section B.4.1 enables the user to inspect the raw information that the LLM received from the database before transforming it into a natural language response. This allows the user to fact check and retrace the answers generated.

5.2.6 *Local Environment via Docker*

A requirement that came up during the late stages of development was a local environment for the prototype to be able to run on any machine so that the final on-site meeting would be possible. This was made possible via Docker and Docker compose. A `docker-compose.yml` in the root of the project orchestrates all four containers necessary to run the prototype.

The frontend and backend also have their own Dockerfile to start the respective application. The backend has a volume defined within it where the user's uploaded XML files are saved.

The neo4j databases use a predefined neo4j image for the respective containers. They are filled via the services `speedparcel-restore` and `textbook-restore` which must be run ahead of starting the application for the first time. This is described in detail in the instruction in chapter B of the Appendix.

5.2.7 *Creating the SpeedParcel and the Textbook Knowledge Graphs*

As described in subsection 5.2.5, the databases are pre-filled with nodes and edges. Their initial knowledge graphs were originally developed during cycles 1, 2, and 3 of the Action Research, as described in section 4.3.

This implementation method quickly reached its limits as the imported datasets became more and more complex.

Todo: This source [17] talks about how natural language text can be used to **create** the knowledge graph. Def use this when explaining how i created my speedparcel imports.

5.3 DATA FLOW

The sequence of interactions and how data flows between each component is visualized in the sequence diagram in figure 5.2. After the user enters their prompt into the frontend, the prompt is forwarded to the backend. The backend then begins preprocessing the request by retrieving the current schema information. This information is vital for the LLM in order for it to know how the knowledge graph is structured and how to create the cyphers for it. Once the backend receives the schema information, the LLM converts the user's prompt into cyphers using all information that it has receives via the context, schema, and prompt. The cyphers are then used to query the selected knowledge graph stored in the neo4j server. The neo4j server returns raw data to the MCP server, which passes it back to the backend. The backend then converts this raw response back into natural language using the LLM. Once converted, the backend saves the user's prompt and the system's response into the chat history before returning the response to the frontend where it is displayed.

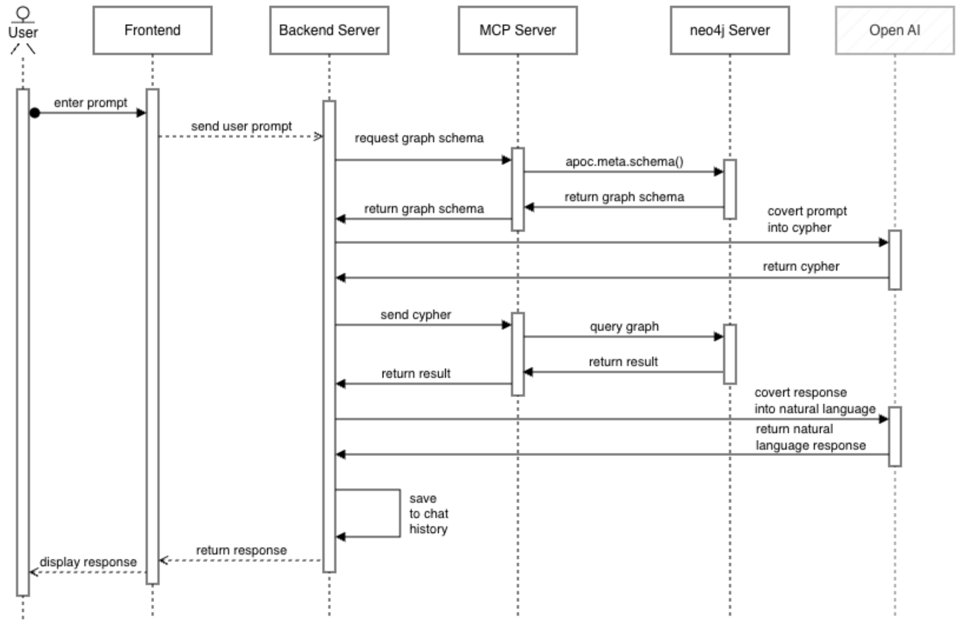

Figure 5.2: A UML sequence diagram showing the main flow of data when a user prompts the system.

6 EXPERIMENTS

What we did with the finished prototype and how we tested it. Prompts we used, cases we built, edge cases, etc.

show some test cases here. break down an answer like "i want to remove the application StatManPlus. What do i have to look out for as an enterprise architect?". the returned answer goes into a lot of depth as this is a level 3 question. break down the result of the query and how the result pulls information about the application but also pulls information from the Lehrbuch database.

Idea: create a list of questions over each category with expected answers and see how it performs. also run x amount of experiments and see how many errors came up (e.g. cypher errors) and count the percent of answers that were perfect, good but could be improved, and wrong / faulty.

```
MATCH (a:Application {name: 'StatManPlus'})
OPTIONAL MATCH (a)-[:SUPPORTS]->(cap:Capability)
WITH a, collect(DISTINCT cap) AS caps
UNWIND (CASE WHEN size(caps) > 0 THEN caps ELSE [NULL] END) AS cap
OPTIONAL MATCH (cap)-[:HAS_CHILD*0..]->(desc:Capability)
WITH a, caps, collect(DISTINCT desc) AS cap_impacted
OPTIONAL MATCH (a)-[:DATA_FLOW]->(outApp:Application)
WITH a, caps, cap_impacted, collect(DISTINCT outApp.name) AS outgoing_apps
OPTIONAL MATCH (inApp:Application)-[:DATA_FLOW]->(a)
WITH a, caps, cap_impacted, outgoing_apps, collect(DISTINCT inApp.name) AS
    incoming_apps
OPTIONAL MATCH (ch:Chunk)
WHERE toLower(coalesce(ch.text,'')) CONTAINS toLower(a.name)
   OR toLower(coalesce(ch.table_summary,'')) CONTAINS toLower(a.name)
   OR toLower(coalesce(ch.title,'')) CONTAINS toLower(a.name)
   OR toLower(coalesce(ch.definition,'')) CONTAINS toLower(a.name)
WITH a, caps, cap_impacted, outgoing_apps, incoming_apps,
     collect(DISTINCT {key: ch.key, title: ch.title, snippet: ch.
    table_summary, text: ch.text}) AS app_chunks
OPTIONAL MATCH (guid:Chunk)
WHERE toLower(coalesce(guid.text,'')) CONTAINS 'decommission'
   OR toLower(coalesce(guid.text,'')) CONTAINS 'retire'
   OR toLower(coalesce(guid.text,'')) CONTAINS 'sunset'
   OR toLower(coalesce(guid.title,'')) CONTAINS 'decommission'
   OR toLower(coalesce(guid.title,'')) CONTAINS 'retire'
   OR toLower(coalesce(guid.title,'')) CONTAINS 'sunset'
   OR toLower(coalesce(guid.definition,'')) CONTAINS 'decommission'
   OR toLower(coalesce(guid.definition,'')) CONTAINS 'retire'
   OR toLower(coalesce(guid.definition,'')) CONTAINS 'sunset'
   OR toLower(coalesce(guid.table_summary,'')) CONTAINS 'decommission'
   OR toLower(coalesce(guid.table_summary,'')) CONTAINS 'retire'
   OR toLower(coalesce(guid.table_summary,'')) CONTAINS 'sunset'
WITH a, caps, cap_impacted, outgoing_apps, incoming_apps, app_chunks,
     collect(DISTINCT {key: guid.key, title: guid.title, snippet: guid.
    table_summary, text: guid.text}) AS guidance_chunks
```

```
RETURN a.name AS application,
       [c IN caps WHERE c IS NOT NULL | c.id] AS supported_capability_ids,
       [g IN cap_impacted WHERE g IS NOT NULL | g.id] AS
    impacted_capability_ids,
       outgoing_apps AS outgoing_data_flows_to_apps,
       incoming_apps AS incoming_data_flows_from_apps,
       app_chunks AS chunks_referencing_application,
       guidance_chunks AS chunks_with_decommission_guidance
```

Listing 6.1: Cypher query to retrieve application impact and decommission guidance

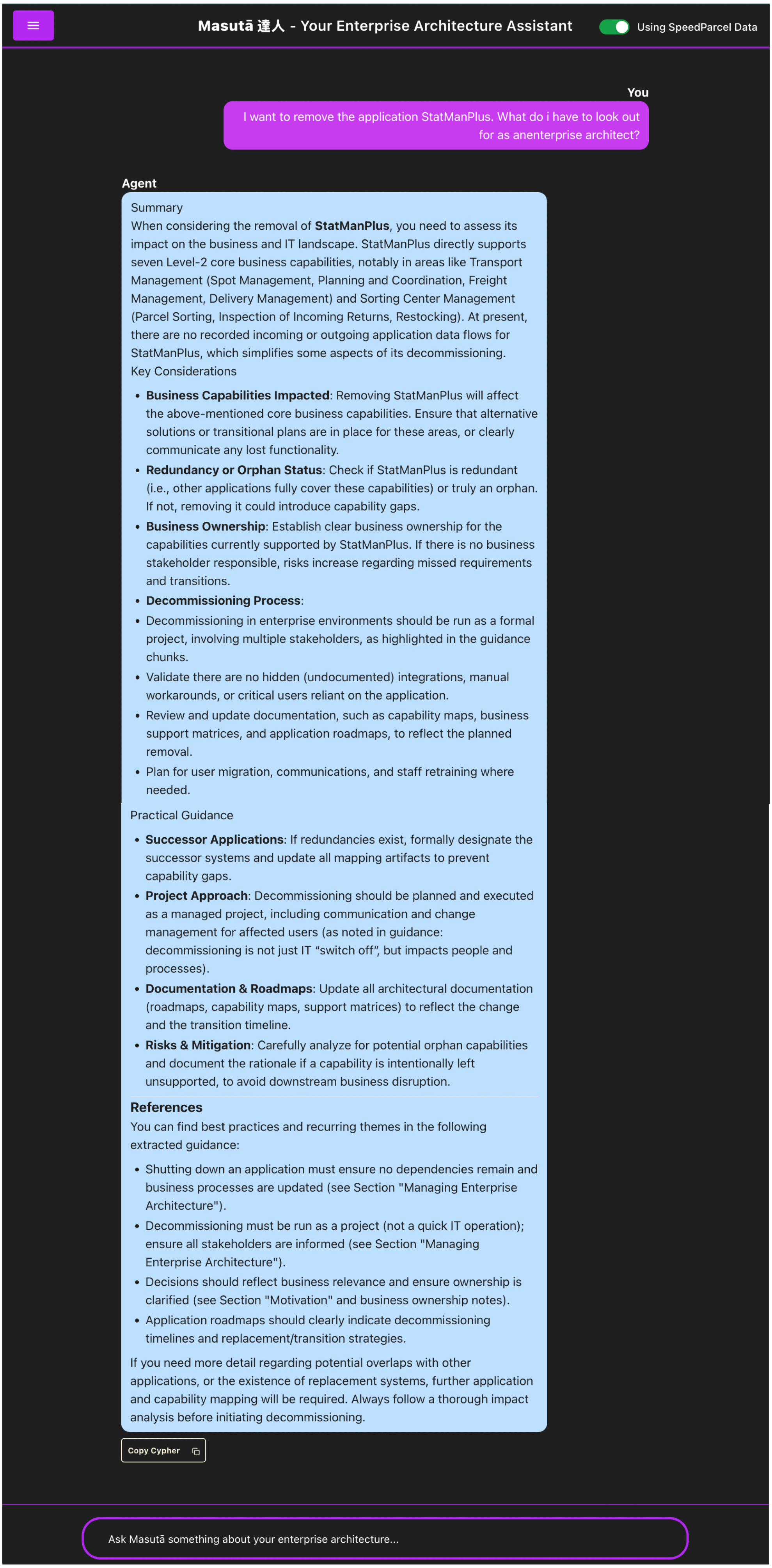

Figure 6.1: Raw response returned from the Cypher in listing ??. Notice that the

application	supported_capability_ids	impacted_capability_ids	outg	incon	chunks_referencing_application	chunks_with_decommission_guidance

"StatManPlus"	["cap_l2_51_spot_management", "cap_l2_52_planning_and_coordination", "cap_l2_53_freight_management", "cap_l2_54_delivery_management", "cap_l2_58_parcel_sorting", "cap_l2_59_inspection_of_incoming_returns", "cap_l2_60_restocking"]	["cap_l2_51_spot_management", "cap_l2_52_planning_and_coordination", "cap_l2_53_freight_management", "cap_l2_54_delivery_management", "cap_l2_58_parcel_sorting", "cap_l2_59_inspection_of_incoming_returns", "cap_l2_60_restocking"]	[]	[]	[{
					snippet:null,
					text:"Capability "Spot Management" (Level 2 in band "Core Capabilities", under L1 capability "Transport Management") is supported by: StatManPlus (flag "x").",
					title:null,
					key:"c7e0b12a9b94ca889c7e13d9e4cea9e504e5e0d2_sec_transport_management_chunk_spot_management"
					}, {
					snippet:null,
					text:"Capability "Planning and Coordination" (Level 2 in band "Core Capabilities", under L1 capability "Transport Management") is supported by: StatManPlus (flag "x"), TrackDB (flag "x").",
					title:null,
					key:"c7e0b12a9b94ca889c7e13d9e4cea9e504e5e0d2_sec_transport_management_chunk_planning_and_coordination"
					}, {
					snippet:null,
					text:"Capability "Freight Management" (Level 2 in band "Core Capabilities", under L1 capability "Transport Management") is supported by: StatManPlus (flag "x").",
					title:null,
					key:"c7e0b12a9b94ca889c7e13d9e4cea9e504e5e0d2_sec_transport_management_chunk_freight_management"
					}, {
						[{
						snippet:null,
						text:"This last difference is visible in a lot of companies today. There are software systems available and used by business people, but nobody on the business side feels responsible for them. Responsibility would include maintenance (i.e. conducting projects to perform changes), support and funding (e.g. for licenses). It, furthermore, refers to making decisions on the future of individual software applications, like its decommissioning if it does no longer provide relevant value. Making decisions based on business relevance implies an assessment from the business perspective. In fact, many systems are just handed over to IT so that they keep them running but there is no business ownership. IT people then need to drive changes driven by business requirements. Ideally, these should be driven by business stakeholders. We still have to convince people in business and IT to understand the relevance of each IT system and then make sure the IT system has business relevance. We further need clear ownership within the business to take the responsibility for decisions on this IT system.",
						title:null,
						key:"27be2852-75f0-44bc-a0d8-343ee3ed7ce9_10"

Figure 6.2: Raw response returned from the Cypher in listing **??**. Notice that the response contains both information about the applications themselves as well as textbook information on enterprise architecture management (right-most column).

7 DISCUSSION

This chapter is meant to evaluate the implementation. what went well. what didn't go so well. etc.

possible improvements:

- The schema call before every prompt is costly for large schemas (i assume as of 02.01.26). This source from wan [25] confirms this - giving the complete schema information can overload the context.
- Does the APOC XML parser create noise in the database? you know how the XML files have view-data in it on how to display it in archi? we're not leaving that out. does that get saved into the database?
- The LLM often doesn't know which node type to look in. E.g. if in archi a landscape was created with information in a "BusinessFunction" node and this is queried in Masuta, then the LLM often doesn't know that the information it should look for is in the business function node type. it searches elsewhere, doesn't find anything, and returns 0 results. This not knowing what node type to look in without supplying the context in the prompt is a necessary improvement.

8 CONCLUSION AND FUTURE WORK

Todo

8.1 CONCLUSION

main learnings include

- Pre-querying the database for its schema so that cyphers can be generated dynamically

8.2 FUTURE WORK

Some ideas for future work: how to improve the challenges mentioned. what other areas this could find utility in. how could access management be handled? e.g. if the database contains customer-information that not every user should be able to see, how can that be differentiated? real-world scenarios that could use my prototype and try out a pilot phase?

BIBLIOGRAPHY

[1] Shoaib Ahmed, Nazim Taskin, David J Pauleen, Jane Parker, et al. "Motivating information technology professionals: The case of New Zealand." In: *Australasian Journal of Information Systems* 21 (2017).

[2] Anthropic PBC. *Introducing the Model Context Protocol*. 2025. URL: https://www.anthropic.com/news/model-context-protocol (visited on 01/02/2026).

[3] Richard L Baskerville. "Investigating information systems with action research." In: *Communications of the association for information systems* 2.1 (1999), p. 19.

[4] Sandra Castro and Jürgen Jung. "Towards Measuring Success of Enterprise Architecture Decisions: Survey among Practitioners and Outline of a Framework." In: *Enterprise Architecture Professional Journal* (2021), pp. 1–22.

[5] Flora Cornish, Nancy Breton, Ulises Moreno-Tabarez, Jenna Delgado, Mohi Rua, Ama de Graft Aikins, and Darrin Hodgetts. "Participatory action research." In: *Nature Reviews Methods Primers* 3.1 (2023), p. 34.

[6] Yang Deng, Lizi Liao, Liang Chen, Hongru Wang, Wenqiang Lei, and Tat-Seng Chua. *Prompting and Evaluating Large Language Models for Proactive Dialogues: Clarification, Target-guided, and Non-collaboration*. 2023. arXiv: 2305.13626 [cs.CL]. URL: https://arxiv.org/abs/2305.13626.

[7] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. "From Local to Global: A Graph RAG Approach to Query-Focused Summarization." 2024. URL: https://www.microsoft.com/en-us/research/publication/from-local-to-global-a-graph-rag-approach-to-query-focused-summarization/.

[8] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. *From Local to Global: A Graph RAG Approach to Query-Focused Summarization*. 2025. arXiv: 2404.16130 [cs.CL]. URL: https://arxiv.org/abs/2404.16130.

[9] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. "A survey on llm-as-a-judge." In: *arXiv preprint arXiv:2411.15594* (2024).

[10] Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. "Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects." In: *Authorea preprints* 1.3 (2023), pp. 1–26.

[11] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. "Model context protocol (mcp): Landscape, security threats, and future research directions." In: *arXiv preprint arXiv:2503.23278* (2025).

[12] Jürgen Jung. "Purpose of enterprise architecture management: investigating tangible benefits in the german logistics industry." In: *2019 IEEE 23rd International Enterprise Distributed Object Computing Workshop (EDOCW)*. IEEE. 2019, pp. 25–31.

[13] Jürgen Jung and Bardo Fraunholz. *Masterclass Enterprise Architecture Management*. Springer, 2021.

[14] Dong Li, Yichen Niu, Ying Ai, Xiang Zou, Biqing Qi, and Jianxing Liu. *T-GRAG: A Dynamic GraphRAG Framework for Resolving Temporal Conflicts and Redundancy in Knowledge Retrieval*. 2025. arXiv: `2508.01680 [cs.AI]`. URL: `https://arxiv.org/abs/2508.01680`.

[15] Zhigen Li et al. *ChatSOP: An SOP-Guided MCTS Planning Framework for Controllable LLM Dialogue Agents*. 2025. arXiv: `2407.03884 [cs.CL]`. URL: `https://arxiv.org/abs/2407.03884`.

[16] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. "Agentbench: Evaluating llms as agents." In: *arXiv preprint arXiv:2308.03688* (2023).

[17] Nandana Mihindukulasooriya, Sanju Tiwari, Carlos F Enguix, and Kusum Lata. "Text2kgbench: A benchmark for ontology-driven knowledge graph generation from text." In: *International semantic web conference*. Springer. 2023, pp. 247–265.

[18] Neo4j, Inc. *Indexes*. Cypher manual documentation on index definition and usage in Neo4j. 2025. URL: `https://neo4j.com/docs/cypher-manual/current/indexes/` (visited on 01/01/2026).

[19] Neo4j, Inc. *Neo4j MCP Server*. GitHub repository. 2025. URL: `https://github.com/neo4j/mcp` (visited on 01/02/2026).

[20] Neo4j, Inc. *Procedures*. Neo4j Operations Manual documentation including the `db.schema.visualization` procedure. 2025. URL: `https://neo4j.com/docs/operations-manual/current/procedures/#procedure_db_schema_visualization` (visited on 01/01/2026).

[21] Neo4j, Inc. *apoc.meta.schema*. Technical documentation for the APOC library in Neo4j. 2025. URL: `https://neo4j.com/labs/apoc/4.1/overview/apoc.meta/apoc.meta.schema/` (visited on 01/01/2026).

[22] OpenAI. *OpenAI API Documentation*. Technical documentation for accessing large language models via the OpenAI API. 2025. URL: `https://platform.openai.com/docs` (visited on 01/01/2026).

[23] Christoph Rühlemann and Matt Gee. "Conversation Analysis and the XML method." In: *Gesprächsforschung* (2017).

[24] Karan Singhal, Shekoofeh Azizi, Tao Tu, S Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, et al. "Large language models encode clinical knowledge." In: *Nature* 620.7972 (2023), pp. 172–180.

[25] Yuwei Wan, Zheyuan Chen, Ying Liu, Chong Chen, and Michael Packianather. "Prompting large language models based on semantic schema for text-to-Cypher transformation towards domain Q&A." In: *Decision Support Systems* (2025), p. 114553.

[26] Anna Wolters, Arnold Arz von Straussenburg, and Dennis M. Riehle. "Evaluation Framework for Large Language Model-based Conversational Agents." In: July 2024.

[27] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. "An explanation of in-context learning as implicit bayesian inference." In: *arXiv preprint arXiv:2111.02080* (2021).

[28] Stefano Zeppieri. "MMAG: Mixed Memory-Augmented Generation for Large Language Models Applications." In: *arXiv preprint arXiv:2512.01710* (2025).

[29] Wenshuo Zhai, Jinzhi Liao, Ziyang Chen, Bolun Su, and Xiang Zhao. "A Survey of Task Planning with Large Language Models." In: *Intelligent Computing* (2025). DOI: 10.34133/icomputing.0124.

Part II

APPENDIX

A
ACTION RESEARCH PROTOCOL

This chapter contains the full Action Research protocol.

A.1 PRECONDITIONS

Test: This reference should have a lowercase, small caps A if the option

A.2 CYCLE 1

Start: October 17th, 2025. End: October 31st, 2025

Diagnosis: The initial problem statement lacked concrete technical formulation and the feasibility of the natural language interaction with enterprise architecture data was unclear. The co-advisor described in detail what his daily work looks like and what tools are used at his company to document and interact with enterprise architecture data. He also showcased the existing solution Dragon1 (described in more detail in section todo).

The co-advisor had the idea of creating an AI, acting as a black-box system, that ingests the specific enterprise architecture data and creates its own internal knowledge graph. All as a black-box. The feasibility as well as academic applicability of this was questioned intensely because of the difficulty of testing such a black-box system when nothing is known about the innern mechanisms.

The academic supervisor recommended an approach using a knowledge graph and supplementing a language model with the domain specific information. He also recommended tools such as Microsoft's Copilot Studio. The academic supervisor also allowed the use of his textbook "Masterclass Enterprise Architecture Management" [13] to feed the knowledge graph with information specific to the role of an enterprise architect.

The difference between the black-box and white-box (knowledge graph) versions is that the black-box system would build its inner mechanisms itself in order to generate domain specific answers. In the white-box version the technical architecture would have to be built manually.

These steps were required in order to scope the potential solution and to think about what possible data sources could be.

First rough ideas were a chatbot with which an enterprise architect could make changes to the application landscape. This included being able to complete unfinished or incomplete application landscapes. For example, being able to interact with the chatbot to identify inconsistencies and weak spots in the landscape.

Action Planning: The goal of the first phase was clear: create a proof of concept. This meant setting up a basic LLM, feeding it with general enterprise architecture data and context so that it is trained to see the interaction through the eyes of an enterprise architect. On top of this a small UI to be able to interact with the proof of concept chatbot. The end-goal of this proof of concept chatbot was to be able to make recommendations on how to edit an application landscape while still covering all business capabilities. The goal until the next cycle would be to be able to ask a chatbot general questions about enterprise architecture management and receive answers based off of the textbook information fed into a knowledge graph.

It was agreed upon, that a single-agent architecture will be used, as multi-agent architectures come with more complexity and are not ideal for this use case.

At this stage, it was not clear that Action Research will be applied during development. At this stage it was assumed that the finished prototype will be passed through an expert interview.

Action Taken: Setup a neo4j database, created a graph representation of the first chapter of "Masterclass EAM". Created a React.js frontend which was directly connected to the graph database. Setup the first prompt context to query the database. Confirmed that the supplemented answers used the knowledge graph by comparing the textbook to the generated answers. Improved embeddings into the graph database (better nodes and relationships). Transforming the textbook into graph representations was done using a python parser which was created with the help of ChatGPT. Improvements on parsing the textbook into a graph representation. Read in the rest of the chapters from the textbook. Created a capability support matrix in excel and parsed it into the database and adapted the queries for this.

Evaluation: Both advisors were ecstatic to see the progress so far after giving a demo of the current state. The co-advisor agreed that building a knowledge graph is the correct path forward after having seen the proof of concept in action.

Learning:

A.3 CYCLE 2

Start: October 31st, 2025. End: November 28th, 2025 **Diagnosis**: a next idea would be to feed in more textbooks. idea would be to compare my results to an existing copilot such as the Microsoft Copilot Studio. An idea came up that it might make sense to have 1 database for the textbook information and 1 database for the enterprise architecture data. we agreed to use one database in hopes of achieving better results as nodes between the textbook and the enterprise data may be connected together in the single database, achieving better results. This is also the phase, where we started discussing potential data sets. the problem here being that the co-advisor is not allowed to simply export his company data and give it to me. we also started discussing how to evaluate my system. we discussed potential experiments and evaluation methods - including testing the knowledge graph directly. we agreed that i would use the data from the assignment of our uni's course "Enterprise

Architecture Management" which i completed in the previous semester. this dataset contains an application landscape and a business capability map for a fictitious company named "SpeedParcel".

Action Planning: we agreed that in this cycle, i would continue implementing more data from speedparcel - new data includes a business object model and a cross-application data-flow diagram. the goal until the end of the cycle would be to improve the querying and have more data in the knowledge graph to query. the "real world data" will come at a later point and only speedparcel data will be used in the next phase.

Action Taken: development went like this: read in the capability support matrix for speedparcel and fit the queries to work with this. i was faced with the challenge that it was impossible to query the data correctly because the schema was getting complex and i needed a better solution than having hard-coded cyphers. after discussing this with a friend, he recommended an MCP server to me. i implemented this and also created a custom node.js backend. UI changes were also done. with the MCP server, responses were quicker and more precise because the cyphers were now being built via LLM (text-to-cypher method) instead of being hardcoded.

Evaluation: Both advisors were very satisfied with the MCP server and the improvements made. both were glowing with ideas on how to continue and how this can be applied in the real world. the co-advisor confirmed that being able to interact with a chatbot is a great way to gain information about the architectural landscape of an enterprise, especially as the data gets more and more complex.

Learning: we identified that there are three categories of questions that the chatbot may be faced with. category 1 being questions specifically in the realm of textbook data, e.g. "how does an enterprise architect do x, y, or z?". Category 2 being simple questions directed towards the architecture, e.g. "how many capabilities does application x support?". Category 3 being a cross section between 1 and 2, e.g. "what do we have to look out for when replacing application x?"

possible use cases for my application came up during discussions about the daily work of an enterprise architect. for example: if an application gets removed and replaced, then there isn't just the technical overhead, but also the application owners who are being replaced. meaning, enterprise architecture also has a lot to do with people management. a seemingly simple change can lead to a complex situation because of stakeholders.

my co-advisor has a large-scale vision for my prototype (this goes beyond my thesis) that it can help during project planning. for example, when planning to remove or add an application, which interfaces need to be touched, which stakeholders will be affected, etc.

A.4 CYCLE 3

Start: November 28th, 2025. End: December 17th, 2025

A.5 CYCLE 4

Start: December 17th, 2025.End: January 23rd, 2026

A.6 FINAL MEETING

Meeting date: January 23rd, 2026. Location: Campus of the Frankfurt University of Applied Sciences.

B INSTRUCTION MANUAL FOR MASUTĀ

Masutā is the name of the prototype enterprise architecture chatbot. Follow these instructions to download Masutā, set it up, and learn how to use it.

When executing the commands from the terminal, always execute them from the root folder of the project. The best results are achieved by executing each command individually (i.e. copy-paste the commands individually line-for-line instead of the entire block).

B.1 DOWNLOADING MASUTĀ

In order to get Masutā up and running, a few prerequisites will have to be fulfilled.

The first is to download the application code for Masutā from GitHub: https://github.com/HendrikDA/masuta-ea-chatbot-prototype. Download the entire folder as a .zip file (under Code→Download ZIP) and unpack it locally.

The second prerequisite is to download Docker and Docker Compose. Instructions on how to set this up on various operating systems can be found under this link: https://docs.docker.com/compose/install/.

An optional prerequisite is to download neo4j Desktop. This allows the user to inspect the graph database within an interactive desktop application. Neo4j Desktop can be found here: https://neo4j.com/download/. However, this is not mandatory and the databases may also be inspected via neo4j's web-application.

B.2 FIRST TIME SETUP

B.2.1 *Setting Environment Variables*

Before being able to run any containers, a few environment variables must be set. This requires creating and editing `.env` files throughout the project. It may be required to use a code editor to execute this step. All `.env` files are prepared via the adjacent `.env.example` file of the individual folder. The following steps explain how to create the required `.env` files per sub-folder:

- **Project Root Folder**: Within the root folder of the project, simply duplicate the `.env.example` file and rename the duplicated file to `.env`. Ensure it is in the root folder of the project. No values in the `.env` file need to be changed for the prototype to work.

- **Frontend**: Within the frontend folder, simply duplicate the `.env.example` file and rename the duplicated file to `.env`. Ensure it is in the root folder of the ./frontend/ folder. No values in the `.env` file need to be changed for the prototype to work.

- **MCP Backend**: Within the mcp-backend folder of the project, simply duplicate the `.env.example` file and rename the duplicated file to `.env`. Ensure it is in the root folder of the ./mcp-backend/ folder. Within the `.env` file, the value for **OPENAI_API_KEY** must be set. It may require that an API key is generated within OpenAI's API platform. All other values may remain as they are for the prototype.

Once these environment variables are set, the databases may be set up.

B.2.2 *Preparing the Databases*

Before being able to set up the databases with the example data, the backup files need to be downloaded and placed into the correct folders. They can be found under the following links:

- Textbook Data: neo4j-2025-11-16T12-53-54-fde218db.backup
- SpeedParcel Data: neo4j-2025-12-10T21-44-58-fde218db.backup

If the above files are not accessible, please reach out to the author or advisors of this thesis.

Once both files are downloaded they are ready to be placed within the project's folders. The Textbok Data backup file must be placed within the folder `./textbook-data/`. The SpeedParcel Data backup file must be placed under `./speedparcel-data/`. This ensures that the backup files are present and in the correct folders when running the Docker commands to set up the databases in the next step.

B.2.3 *Setting Up the Databases*

If Masutā is being run for the first time, then two containers need to be executed once so that they populate both databases with the default data. From within the root folder of the project, execute the following commands

```
docker compose --profile speedparcel-restore run --rm speedparcel-restore
docker compose --profile textbook-restore  run --rm textbook-restore
```

Windows: If the executing system runs on Windows, this step may require the use of WSL (Windows Subsystem for Linux) and to set the HOME variable via SET HOME=\C:\Users\<username>. It may be required to execute these commands from a normal terminal instead of the Windows PowerShell.

Once these commands have run through successfully, both databases are populated with the default data and are ready to be used.

B.2.4 *Hard-Resetting Everything*

If at any time a hard-reset for the entire application is required, run the following commands:

```
docker compose down
rm -rf $HOME/neo4j/data
rm -rf $HOME/neo4j_empty/data
```

After resetting the application, restart with the first time setup.

B.3 RUNNING THE APPLICATION

Running Masutā is as simple as running the following command:

```
docker compose up
```

When running this command for the first time, it may take a while as it will have to download dependencies.

With this command, four containers are started.

- The SpeedParcel neo4j database filled with example data and the textbook
- The playground neo4j database filled with only the textbook
- The MCP-Backend which is the backend node.js which by default runs under http://localhost:4000
- The frontend which by default runs under http://localhost:3000/

If Docker does not automatically open the frontend in the browser, navigate to it under http://localhost:3000/. Everything is now ready and Masutā may be used.

B.4 FEATURE OVERVIEW

This section details the features within Masutā.

B.4.1 *Input Field and Chat History*

At the bottom of the application is an input field where the user may prompt Masutā. Pressing enter sends the prompt. The user's input prompt then appears as a chat bubble bound to the right of the chat-area. The system then starts working on answering the prompt and displays a chat bubble bound to the left of the chat-area displaying "Thinking...". As soon as the application is done thinking, the response is displayed in the same chat bubble.

Masutā's response has a button appended to it which allows the user to copy the cypher which was used to generate the answer. The user may take this copied cypher and paste it into neo4j desktop to inspect the raw response that was used to generate the answer.

B.4.2 *Toggling the Database*

At the top right of the application is a toggle switch. This allows the user to toggle between the SpeedParcel database and the playground database.

B.4.3 *Resetting the Playground Database*

Under the menu at the top left, the user is presented with the option to reset the database. This option is only available if the playground database is selected via the toggle switch. After confirming the reset in the dialog that pops up, the playground is reset to its default state with only the textbook information. The SpeedParcel database cannot be reset via the UI.

B.4.4 *Importing Custom Data*

Under the menu at the top left, the user is presented with the option to import their own data. This option is only available if the playground database is selected via the toggle switch. Within the dialog that pops up, the user can add up to 10 files either by selecting them via the file system or by dragging-and-dropping them into the dialog. Only .xml files are accepted.

After clicking the import button, the files are uploaded. If the upload was successful, an alert is shown notifying the user of this. The uploaded data remains within the realms of the Docker containers and is not uploaded to any third-party sources. The data can then be queried if the database toggle is set to the playground database.

B.4.5 *Inspect Database*

Under the menu at the top left, the user is presented with the option to view the graph data in neo4j browser their own data. This option is available for both the SpeedParcel and playground database. Clicking this opens another tab under https://console-preview.neo4j.io/tools/query. Here, the user can connect to the selected database. The connection string is displayed in a dialog within Masutā. Here, the user can inspect the graph structure and paste the cyphers used for the generated responses.

In order to access the neo4j console, it may be required that the user has a neo4j account.

B.4.6 *Chat Context and Building on Previous Answers*

Masutā saves the chat history so that the user is able to build upon previous prompts. The chat history is set so that the single previous user prompt and corresponding response is saved. This means that a further question can be asked about the previous prompt or its response. The chat history's context only goes back one prompt and response.

B.5 EXAMPLE QUESTIONS

Questions can be broken down into three categories:

1. Category 1: Textbook questions about enterprise architecture management
2. Category 2: Basic information about the enterprise architecture data
3. Category 3: Questions that span both category 1 and 2 and are more complex than questions in category 2. These require more in-depth information about the data as well as how to deal with it as an enterprise architect.

The following list of questions serve as examples that Masutā can handle. These simply serve as ideas to test the system - the user may input their own thought up questions.

1. Questions in Category 1
   - What are the schools of enterprise architecture management?
   - How does enterprise architecture relate to town planning?
2. Questions in Category 2
   - How many capabilities are supported?
   - How many applications are in use?
   - Which application supports the most capabilities?
   - If I remove application x, which capabilities will be affected?
3. Questions in Category 3
   - We plan on removing application x. What do we have to look out for when doing so?
   - We wish to implement a new application x which covers the capabilities x, y, and z. Which existing applications may become redundant?
   - Which capabilities currently rely on single-point-of-failure applications? How should we deal with this?
   - Which capabilities are over-supported by multiple applications?
   - What interfaces does application x have?
   - How big of a task is it to migrate from application x to application y?

This concludes the instruction manual for Masutā. Beyond the example questions, users are encouraged to prompt Masutā to explore the example data provided in SpeedParcel or to analyze their own proprietary data after importing it, in order to gain insights into their enterprise architecture.

C MASUTĀ CODE SNIPPETS

This chapter showcases code snippets which are of special interest as they show how key functionalities of Masutā work. The code snippets do not showcase the full application and may not complete.

C.1 SYSTEM PROMPTS

The following code snippets showcase what system prompts are passed to the LLM in order to generate the cyphers via the natural language text input and how the cypher results are transformed back into natural language.

C.1.1 *System Prompt to Generate Cyphers from Natural Language Prompt*

```
async function nlToCypher(nlPrompt: string, schema: string) {
  const systemPrompt = "
    You are working with a Neo4j graph whose structure is described in the schema summary.

    Important rules:
    - Do not assume fixed labels like :Application or :Chunk unless they appear in the schema.
    - Infer meaning from label names (e.g. ApplicationComponent     application).
    - If no text-centric nodes exist, answer using structural relationships.

    Index usage based on the schema information passed to you:
    - If the schema summary lists a VECTOR index relevant to the task, start with:
      CALL db.index.vector.queryNodes(<indexName>, $embedding, <k>) YIELD node, score
    - If the schema summary lists a FULLTEXT index relevant to the task, start with:
      CALL db.index.fulltext.queryNodes(<indexName>, $query, {limit: <k>}) YIELD node, score
      (Do NOT pass a bare integer as the 3rd argument.)
    - After any CALL ... YIELD, you MUST finish with a RETURN clause.
      Example: CALL ... YIELD node, score RETURN node, score
    - Otherwise use MATCH with WHERE + indexed properties.

    Your task:
    - Write a SINGLE valid Cypher query.
    - Output ONLY Cypher. No explanations.

    Cypher syntax rules (important):
    - Do NOT use exists(node.property).
    - Neo4j 5+ requires property existence checks to use:
      node.property IS NOT NULL
```

```
    - Always use IS NOT NULL instead of exists(...)
    - NEVER use EXPLAIN or PROFILE. Always output an executable query that ends
        with RETURN.
    - You MAY start with CALL db.index.fulltext.queryNodes(...) or CALL db.index
        .vector.queryNodes(...).
  "
};
```

C.1.2 *System Prompt to Natural Language Response from Cypher Results*

todo

D

MASUTĀ EXAMPLE QUESTIONS AND ANSWERS

This chapter contains screenshots of user prompts and the system's responses. It serves as a showcase of what the application looks like and how it responds to questions.

D.1 QUESTION XYZ

todo