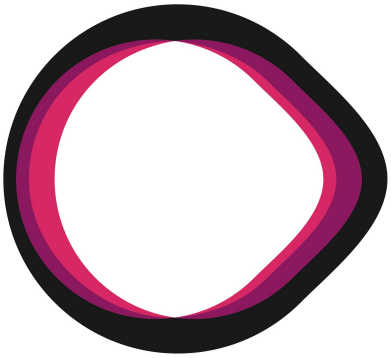
Studienarbeit
Documentation

Conversational Ai Client For Bank Client Onboarding

Semester: Fall 2025



OST
Eastern Switzerland
University of Applied Sciences

Date: 2025-12-19

Project Team: Arnel Veladzic
Fabio Gomes Silva

Project Advisor: Dr. Prof. Mitra Purandare

School of Computer Science
OST Eastern Switzerland University of Applied Sciences

Part I

Abstract

Abstract

Financial institutions and banks using Atfinity's solution, rely on their highly configurable, rule-driven onboarding system in which field visibility and required inputs depend on previously collected information. This dynamic structure poses a challenge for AI-based data extraction, as static mapping approaches cannot adapt to evolving field dependencies or bank-specific configurations. The objective of this project was to develop and evaluate an AI-supported prototype that extracts client information from natural-language input, maps it to Atfinity's conditional data model and interacts with the Case Management System through the Model Context Protocol.

The proposed solution introduces an LLM-driven mapping framework capable of adapting to the Case Management System, caching values for fields not yet visible and validating extracted data against Atfinity's schema. Three prototype variants were implemented: (1) an iterative LLM mapping approach that repeatedly queries the LLM and the Case Management System to adapt to conditional field exposure, (2) an iterative approach enhanced with a preemptive cache with the goal to increase performance and (3) a single-pass mapping architecture using a precomputed cache to minimise latency.

Validation across multiple scenarios demonstrates that the iterative mapping approach (1) provides the most robust handling of dynamic field dependencies, achieving up to 95.24% F1 score in simple cases and maintaining predictable behaviour in complex configurations. The preemptive cache variant in combination with the iterative LLM mapping (2) improves recall but introduces precision losses and longer execution times, while the single-pass solution (3) offers the highest performance (10–24 seconds) at the cost of reduced recall in complex cases.

Overall, the results demonstrate that LLM-based information extraction combined with incremental field mapping is a feasible and effective approach for Atfinities client onboarding platform. Future work should focus on extending the prototype with a multi-agent architecture and additional robustness measures in order to move the solution closer to production readiness.

Part II

Management Summary

Management Summary

Atfinity

Atfinity is a Swiss fintech based in Zurich offering an AI-supported, no-code automation platform for banks and financial institutions. The platform digitizes and streamlines key processes such as client onboarding, KYC/KYB/AML and client lifecycle management, while integrating with existing IT systems. Atfinity helps institutions improve operational efficiency, accelerate onboarding and ensure regulatory compliance. It is used by clients including Bitcoin Suisse, Kaleido and Kaiser Partner.

Situation

Client onboarding in the banking sector is a complex and highly regulated process. Although banks follow similar regulatory frameworks, the concrete rules, required documents and process flows vary significantly across institutions, banks and client types. Atfinity addresses this challenge through a dynamic onboarding platform that guides Relationship Managers step by step and adapts the onboarding flow to the current context. Fields are displayed only when they become relevant based on previously entered information.
While this conditional logic significantly reduces cognitive load and enforces regulatory consistency, large parts of the onboarding process still rely on manual entry of client data by Relationship Managers. This manual effort is time-consuming, prone to errors and costly. To address these limitations, Atfinity aims to extend its platform with AI-driven onboarding automation that allows users to provide client information in natural language. The system should automatically interpret, validate and map this information to the appropriate fields. Furthermore, client cases should be created and updated automatically within Atfinity's Case Management System through the Model Context Protocol, enabling seamless integration between conversational AI and Atfinity's backend system without manual intervention.

Problem

As briefly outlined in the previous section, Atfinity's onboarding system is built on a conditional field architecture in which the visibility and relevance of many fields depend on previously provided inputs. For example, fields such as *Total Wealth* or *Name* may only become available after the *Account Type* has been defined. This behavior reflects standard onboarding procedures in banks and financial institutions, where information is collected in a staged and conditional manner. Typically, a Relationship Manager begins the onboarding process by completing an initial form. Depending on the values entered, additional forms may be required; for instance, selecting a specific option may trigger the completion of Form C, while another option may lead to Form B. This decision-based progression continues until all required information has been collected and no further forms are necessary. Atfinity has digitized this traditional workflow by accurately modeling these conditional dependencies and dynamically guiding users through the onboarding process in a fully digital environment.
The underlying data model is represented by a large JSON structure containing field definitions, dependencies and other relevant data. Because banks can configure these structures freely, JSON payloads vary widely.
This flexibility introduces several key challenges for developing a prototype using the Model Context Protocol with AI-driven data extraction and mapping:

- **Dynamic visibility:** The LLM must map values to fields that may not yet be visible.

- **Configuration variance:** Bank-specific field sets prevent the use of static mapping. This means that every bank/configuration can have different field keys, processes, id's, etc.
- **Lack of patterns:** There are currently no established architectural best practices for bank client onboarding automation with AI-driven data extraction and dynamic field mappings using the Model Context Protocol.
- **Data mapping complexity:** Incoming humand-readable text gets mapped to a structued data schema (JSON).

These uncertainties required extensive analysis, experimentation and iterative refinement of architectural concepts.

Approach and Technology

The project developed three prototypes that enable natural-language intake for onboarding using a Large Language Model (LLM). The system processes free-text input, extracts information and maps it to Atfinity's internal JSON model.
The project followed a structured yet exploratory development approach:

Analysis of the Existing System Architecture

- Examination of Atfinity's onboarding rule-engine and Case Management System.
- Understanding the conditional field architecture.
- Analysis of the JSON model containing field definitions, dependencies and other relevant data.

Analysis of MCP and Integration Concepts

- Review of the MCP documentation and implementation of a minimal MCP server for experimentation.
- Exploration of different interaction models between the LLM and Atfinity via MCP tools.

Architectural Design and Iterative Refinement

- Initial development of a simple three-layer architecture.
- Versioned architecture approach: each iteration built on previous learnings and addressed new challenges.
- Evaluation of each architecture through predefined expected-vs-actual behaviour analysis, as well as through the automated testing framework.

Test Concept and Quality Measurement

- Development of an automated testing framework.
- Quality metrics included precision, recall, F1 score and execution time.

Results

The project successfully met its primary objective: developing a working prototype capable of extracting client information from natural-language input and mapping it into structured JSON suitable for Atfinity's Case Management System. Three prototypes were implemented, sharing the same core architecture but differing in communication flow:

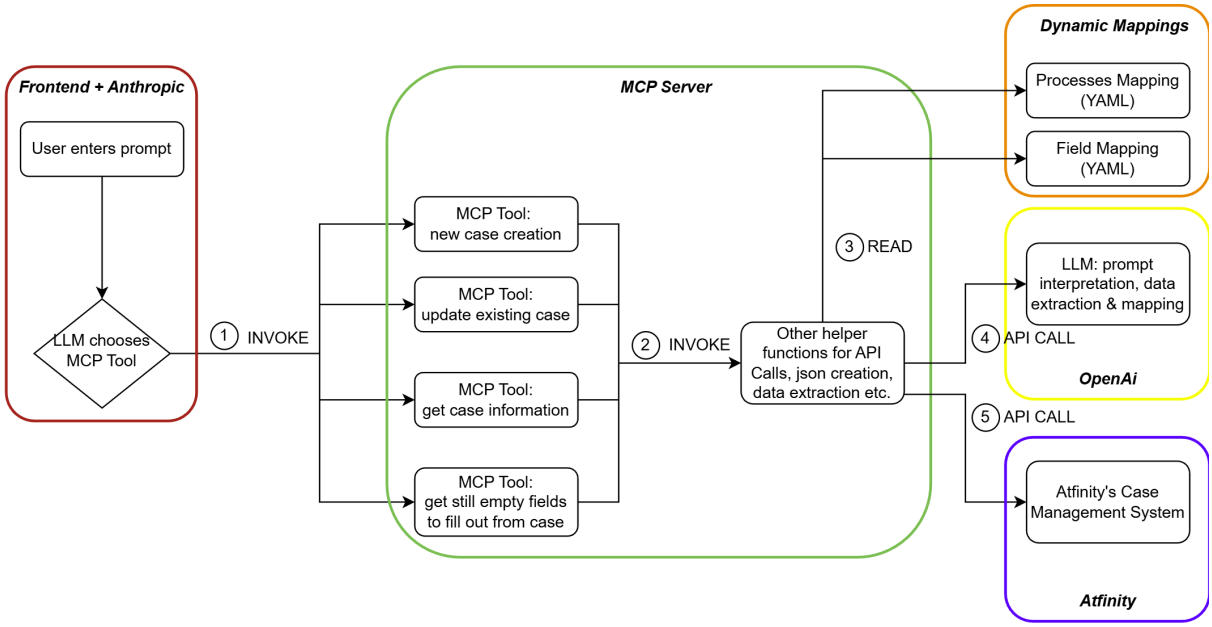


Figure 1: Simplified core cummunication flow

1. Prototype: Iterative LLM-Driven Mapping (v5)

This version adapts directly to Atfinity's incremental field-disclosure logic. After identifying the required onboarding process, the system creates a blank case, retrieves the current disclosed fields via MCP and sends both the Relationship Managers prompt and the available fields to the LLM for data extraction and mapping. The returned JSON is transformed and applied to the case. If new fields become visible, the cycle repeats until no further field disclosures occur.
It achieved high accuracy in simpler cases (Precision 90.91%, Recall 100%, F1 95.24%) with 23–25s latency. Performance dropped in corporate onboardings (F1 80%, Recall 66.67%).

2. Prototype: Iterative Mapping with Preemptive Cache (v6)

The second prototype introduces a full ontology and field (information) YAML mapping, describing all fields, constraints and permitted values the corresponding Case Management Environment. Before case creation, the LLM generates a comprehensive mapping by mapping the YAML file against the Relationship Managers prompt. During case updates, cache values are applied first, with LLM calls triggered only for unmapped fields.
It achieved 100% Recall but lower Precision (66.67–73.68%) and the slowest speed (32–44s).

3. Prototype: Preemptive Cache with Single-Pass Mapping (v7)

The final architecture retains the ontology and field (information) YAML mapping. Once the initial cache is computed, all values are retrieved exclusively from it, prioritizing speed and reducing communication complexity. No further LLM calls are made for unmapped fields.
It achieved the best speed (10–24s), however Recall decreased in complex cases (44.44%).
All prototypes were tested across four automated onboarding scenarios and manual defined scenarios of varying complexity.

	Architecture v5	Architecture v6	Architecture v7
Test Case 1: Individual Account Onboarding	Precision: 86.67% Recall: 92.86% F1 Score: 89.66% Duration: 25198ms	Precision: 73.68% Recall: 100.00% F1 Score: 84.85% Duration: 36907ms	Precision: 91.67% Recall: 78.57% F1 Score: 84.62% **Duration**: 19813ms
Test Case 2: Simple Onboarding	Precision: 90.91% Recall: 100.00% F1 Score: 95.24% Duration: 23377ms	Precision: 66.67% Recall: 100.00% F1 Score: 80.00% Duration: 34382ms	Precision: 100.00% Recall: 80.00% F1 Score: 88.89% **Duration**: 14991ms
Test Case 3: Corporate Entity	Precision: 100.00% Recall: 66.67% F1 Score: 80.00% Duration: 14606ms	Precision: 90.00% Recall: 100.00% F1 Score: 94.74% Duration: 31937ms	Precision: 100.00% Recall: 44.44% F1 Score: 61.54% **Duration**: 10650ms
Test Case 4: Complex Case with detailed data	Precision: 85.00% Recall: 100.00% F1 Score: 91.89% Duration: 39546ms	Precision: 85.00% Recall: 100.00% F1 Score: 91.89% Duration: 43602ms	Precision: 93.75% Recall: 88.24% F1 Score: 90.91% **Duration**: 23729ms

Figure 2: Quality Measure of final prototypes

Overall, the prototypes demonstrate that AI-based onboarding via MCP is feasible, with architecture v7 offering the best trade-off between speed and quality and architecture v5 performing most reliable in terms of accuracy.

Contents

Part III

Introduction

Chapter 1

Overview

1.1 Atfinity

Atfinity (Atfinity AG) is a Swiss financial technology company headquartered in Zurich that provides an AI-enhanced, no-code process automation platform for banks and financial institutions. Its software enables organizations to digitize and automate complex operational processes, such as client onboarding, Know-Your-Customer (KYC)/Know-Your-Business (KYB)/Anti-Money Laundering (AML) procedures, client lifecycle management and loan origination. The platform is designed to integrate with existing IT systems and supports rapid deployment, compliance enforcement and flexibility in workflow configuration without requiring extensive coding. Atfinity's solution aims to improve operational efficiency, accelerate onboarding times and ensure regulatory compliance, serving private banks, wealth managers, fintechs and other financial service providers. The company's tools have been adopted by institutions such as Bitcoin Suisse, Kaleido and Kaiser Partner. [3]

1.2 Background / Problem statement / State of the Art

Client onboarding in the banking sector is subject to strict regulations that require different information to be collected and validated depending on the client and the account type. This conditional onboarding process creates a scenario where a comprehensive set of potential data fields exists, but only a subset is required for any given client based on their profile and circumstances. For instance, enhanced due diligence requirements may be triggered by specific client attributes such as political exposure, resulting in additional documentation and compliance checks. In practice, this conditional logic creates a highly complex and repetitive workflow that consumes significant time and resources.
Atfinity addresses this challenge through a digital platform that encodes these conditional business rules. Rather than requiring users to navigate static forms, the system dynamically determines field visibility and relevance based on data progressively entered during the onboarding process. When a Relationship Manager initiates a new client onboarding case, an initial set of mandatory fields is presented. As these fields are being filled, the system evaluates the entered values and may reveal additional fields. For example, fields such as Total Wealth or specific account features become visible only after the Account Type has been specified. While this architecture reduces cognitive load and enforces consistency, the underlying data entry task remains manual and time-intensive.
To further improve efficiency, Atfinity envisions extending its platform with AI-capabilities. The goal is to enable Relationship Managers to provide client information through unstructured humand-readable free-text as a conversational prompt, which an AI system would then interpret, extract relevant data from and automatically create a new onboarding case and populate it into the appropriate case fields. This would substantially reduce manual effort and human error, automating what is currently an intensive process.
However, realizing this vision presents substantial technical obstacles. The first arises from Atfinity's incremental architecture. The platform's data model is a large JSON, which evolves with each bank's specific requirements. Banks can define their own fields, validation rules, permitted values and field dependencies tailored to their regulatory environment. Consequently, the underlying JSON structures representing onboarding cases vary significantly across banks and financial institutions, making static mapping approaches and hard-coded extraction logic infeasible. Each new bank or configuration change would require redesign and re-development of the prototype.

Finally, integrating large language models with business applications through emerging architectural patterns such as the Model Context Protocol remains an understudied problem domain. While MCP provides a standard for large language models to interact with other systems, established best practices for field extraction, validation workflows, error handling and interaction flows, especially in banking-specific environments, are scarce. Designing a robust solution requires substantial exploratory work to understand how to effectively combine MCP's architectural principles with the specific constraints of Atfinity's conditional architecture.

1.3 Goals

The overarching goal of this project is to design, analyze and develop a working prototype that addresses the challenges identified in section 1.2. Specifically, the project aims to demonstrate that AI-powered field extraction can be effectively integrated into Atfinity's case management system using the Model Context Protocol. The prototype should enable Relationship Managers to input client information through natural language prompts and automatically extract, map and populate relevant data into Atfinity's case management system.

1.3.1 Prototype Objectives

The prototype must fulfill the following core objectives:

- Accept user prompts containing client information in natural language
- Interact with an MCP server to enable extraction and mapping logic to large language models as well as automatic case creation and updating
- Accurately extract relevant client data from prompts regardless of how explicitly the information is presented
- Map extracted data to Atfinity's dynamic field structure, respecting conditional rules and dependencies
- Answer questions and provide valuable information to a certain case
- Implement other valuable MCP tools that might help an RM in his everyday work with client onboarding cases

1.3.2 Research Questions

During the project, five research questions emerged that guided the development and evaluation:

1. **RQ1 (Information Extraction Accuracy):** How effectively can a large language model extract and map client information from free-text prompts into Atfinity's structured fields, even when the prompt does not explicitly indicate the type of information provided?
    - This question evaluates the model's semantic understanding and field-mapping capability during case creation and updates. Success is measured using field extraction accuracy and precision metrics compared against reference solutions.
2. **RQ2 (Intent Detection):** How accurately can the system distinguish between user queries and data input requests and how does this classification affect overall correctness?
    - This question addresses the reliability of intent classification, particularly distinguishing between create, update and query operations. Misclassification can lead to incorrect API calls or unintentional data modifications.
3. **RQ3 (Client Suitability):** Is a standard MCP client/open source UI sufficient for solving the problem domain, or does the domain require a dedicated client interface?
    - This question examines the interaction layer between the user and the system. While standard MCP clients and open source UI's provide generic tool exposure, banking workflows may require domain-specific abstractions, user guidance or traceability features to improve adoption and reduce errors.

4. **RQ4 (Performance):** What are the end-to-end latencies for automated onboardings?

    - This question measures the overall case creation duration. How much time is needed from entered prompt to finished onboarding. Performance is critical for user acceptance and user experience.

5. **RQ5 (Architecture and Model Selection):** What architecture is best suited for this task? How do different large language models perform? Are locally-deployed models viable alternatives to cloud-based AIs?

    - This question compares different prototype configurations, model choices (such as Claude versus GPT-4o-mini), and deployment strategies. Evaluation criteria include extraction accuracy, throughput, infrastructure complexity and other metrics defined in section 5.4.

1.3.3 Research Value

An important aspect of this project is that the research questions are valuable independent of whether a fully functional prototype is achieved. Answers to each research question provide valuable insights for Atfinity and the broader community of practitioners working on MCP and AI-driven data extraction and mapping. Understanding the limitations of current LLM's in field extraction, the reliability of intent detection and the architectural requirements for MCP-based banking integrations contributes to the field even if not all implementation challenges are fully resolved. The findings serve as a foundation for future iterations and refinements.

1.4 Deliverables

The expected deliverables are:

- Protoype(s): Source code
- Comprehensive documentation of all aspects of the developed systems including answers to research questions
- Blog article for Atfinity's website about the conducted work and results

All deliverables except for the source code and blog article are part of this document. The complete source code and blog article is handed in separately.

1.5 Abbreviations

RM - Relationship Manager
CMS - Case Management System (of Atfinity)
MCP - Model Context Protocol
AI - Artificial Intelligence
LLM - Large Language Model
MVP - Minimum Viable Product
RUP - Rational Unified Process
UI - User Interface
Ontology - Atfinity specific implementation: Imagine it as a Java Class/Object
Information - In the context of Atfinity's CMS an Information is a Field e.g. total wealth

Part IV

Product Documentation

Chapter 2

Requirements

2.1 Use Case Overview

The use case diagram illustrates how the Relationship Manager, the Large Language Model and the MCP System interact to enable AI-driven, automated client onboarding. The workflow begins when an RM enters a free-text prompt containing client information or questions. The system detects the user's intent and executes one of three operations: creating a new case, updating an existing case, or retrieving case information/data.
The system involves three main actors:

- **Relationship Manager (RM):** Provides natural language input about client information or asks questions about cases.
- **Large Language Model (LLM):** Analyzes user intent to determine the correct MCP tool.
- **MCP System:** Exposes the core operations (create, update, retrieve) to Atfinity's backend. The MCP System in this diagram encapsulates the decoupled LLM component responsible for data extraction and mapping. (Detailed Core Architecture)

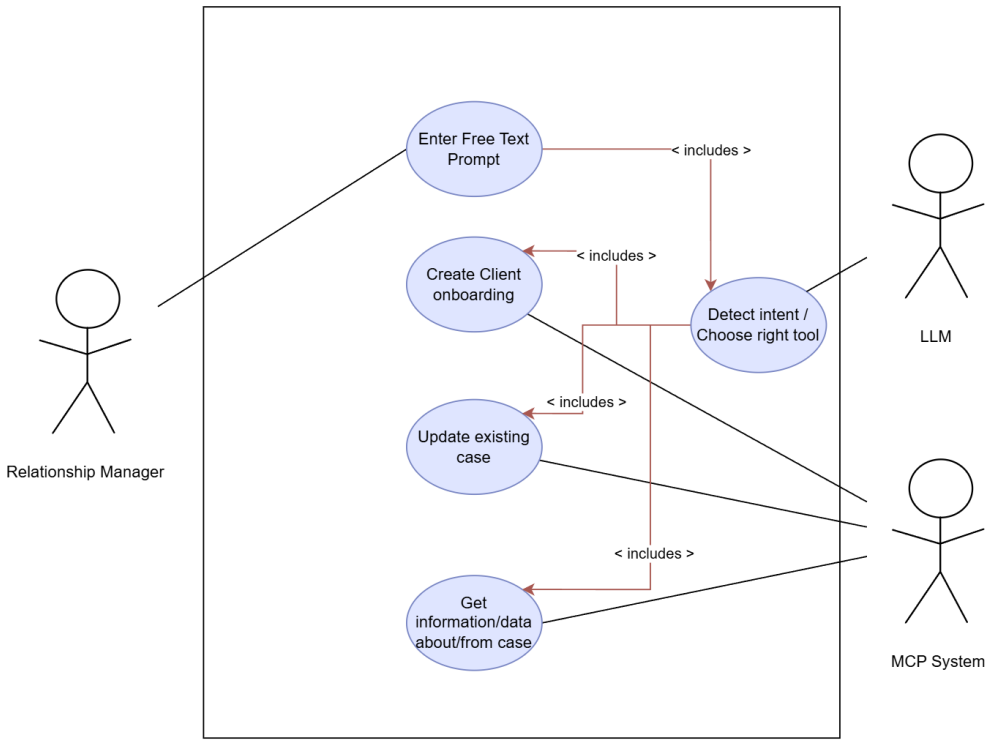


Figure 2.1: Use Case Diagram

2.2 User Stories

User stories capture the needs and workflows of Relationship Managers using the system:

2.2.1 User Story 1: Create automatic client onboarding

As a Relationship Manager, I want to conversationally input client information using natural language which will automatically populate and fill out the onboarding case, so that I can avoid manually filling out lengthy and repetitive forms.

2.2.2 User Story 2: Ask for still open fields to fill out

As a Relationship Manager, I want the system to understand which fields of an existing case are currently required and not filled out yet, so that I know what information to provide to the LLM.

2.2.3 User Story 3: Update existing Case

As a Relationship Manager, I want to update an existing case without manually searching for the field to update.

2.2.4 User Story 4: Get relevant information and ask questions about case

As a Relationship Manager, I want to ask questions about a certain case so that I can retrieve relevant information without manually navigating the system.

2.3 Functional Requirements

Functional requirements specify what the system must do to fulfill the user stories:

2.3.1 FR1: Retrieve Case information and other details

The system must fetch the current case from Atfinity's CMS, including all exposed fields, required information and other metadata. The LLM must represent a summary and answer the RM's questions via the frontend. (Supports User Stories 2 & 4)

2.3.2 FR2: Extract Structured Data from Free-Text Input

The system must process free-text input provided by an RM and automatically extract relevant data. The extraction must identify information such as names, addresses, nationalities and other provided information that match available fields to fill out. The system must handle variations in phrasing, different numerical formats and expressions of uncertainty (e.g., "approximately 100,000 CHF"). (Supports User Stories 1 & 3)

2.3.3 FR3: Work hand-in-hand with Atfinity's CMS

The system must adapt to and respect Atfinity's incremental architecture where field visibility depends on previously submitted information. (Supports User Story 1)

2.3.4 FR4: Detect User Intent

The system must classify incoming user messages as either questions or data input. Questions are requests for information about the case (e.g., "Whats the total wealth of Case 211?"), while data inputs are statements providing information to be extracted (e.g., "Change total wealth to 10 of case 211"). If a message is classified as a question, the system returns the requested information from the requested case. If classified as data input, the corresponding process and MCP Tool is being triggered. Furthermore, the system must distinguish the correct workflow to trigger within Atfinity's CMS based on the RM's prompt. (Supports all User Stories)

2.3.5 FR5: Expose Operations as MCP Tools

The system must provide all core operations as MCP tools accessible through a frontend. Each tool must return structured JSON responses according to the MCP specification. (Supports all User Stories)

2.4 Non-Functional Requirements

Non-Functional requirements specify what the system must fulfill in order to judge the operation of it:

2.4.1 NFR-1: Performance

The MCP server must process and map a typical onboarding text (approximately 150-200 words) to Atfinity's fields within 30 seconds. This includes LLM reasoning times, field mapping and any other necessary API calls. This requirement ensures acceptable responsiveness for interactive use by Relationship Managers.

- Acceptance: $\leq 30sec$

Verification: Benchmark the architecture under test conditions with realistic input samples. Measure end-to-end processing time for at least 3 representative test cases - each covering a different scenario.

2.4.2 NFR-2: Precision

Precision ensures that extracted values are correct and reduces the risk of incorrect data being submitted to Atfinity.
The ratio of correctly extracted fields among all extracted fields must meet the following target:

- Acceptance: ≥ 0.80 (80%)

Verification: Evaluate extracted fields against human-validated gold-standard reference data.

2.4.3 NFR-3: Recall

Recall ensures that relevant information is not missed or overlooked during extraction, preventing incomplete case submissions.
The ratio of correctly extracted fields among all fields that should have been extracted must meet the following target:

- Acceptance: ≥ 0.90 (90%)

Verification: Evaluate extracted fields against human-validated gold-standard reference data.

2.4.4 NFR-4: F1-Score

The F1-score balances the trade-off between correctness (precision) and completeness (recall), providing the primary aggregate measure of extraction quality. It ensures that the system neither sacrifices accuracy for coverage nor achieves high accuracy at the cost of missing information.
The F1-score, representing the harmonic mean of precision and recall, must achieve the following target:

- Acceptance: ≥ 0.85 (85%)

Verification: Evaluate extracted fields against human-validated gold-standard reference data.

2.4.5 NFR-5: Usability

The chatbot must handle users free-text prompts in natural language without requiring them to know or provide Atfinity's technical terms or field IDs.
Verification: A RM can enter a humand-readable free-text prompt containing client information which will be automatically processed without them knowing any further technical details of the architecture.

Chapter 3

Model Context Protocol (MCP)

3.1 Introduction and Definition

The Model Context Protocol is a standardized, open-source protocol that enables large language models to interact with external systems, data sources and tools. Rather than integrating tools directly into an AI application, MCP provides a uniform interface through which LLMs can discover, request and execute operations on external systems. It decouples the AI model from specific integrations, allowing the same LLM to work with multiple backends and enabling different clients to reuse the same integrations.

Think of MCP like a USB-C hub. On one side, you can plug in different LLMs (Claude, GPT-4, Llama, or any other model). On the other side, you can plug in different backend systems (banking platforms like Atfinity, CRM systems, databases, etc.). The hub allows all these LLMs and backends to communicate with each other seamlessly without needing to know anything about one another. The LLM doesn't need to understand the specifics of the backend and the backend doesn't need to be aware of which LLM is using it. They communicate through the standardized MCP protocol, just as any device with USB-C can communicate with a USB-C hub without custom drivers or special knowledge. [7]

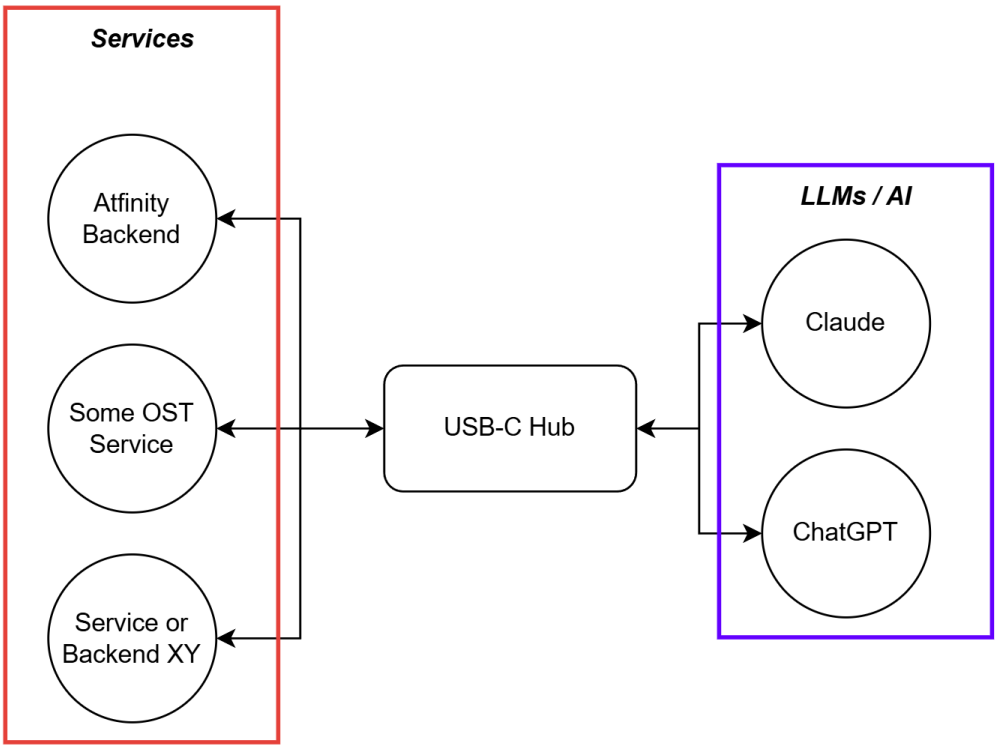


Figure 3.1: MCP - USB Hub Example

3.2 Why MCP Matters and What It Enables

Integrating large language models with external systems has traditionally required custom, tightly coupled solutions. Such integrations are often designed for a specific model, application or backend system, which leads to significant limitations in scalability, maintainability and reuse. As a result, organizations frequently duplicate integration logic across projects. This fragmentation increases development effort and introduces inconsistencies that complicate system evolution. Moreover, conventional integrations tend to be inflexible. Changes to the underlying backend system, the introduction of new tools or the replacement of the language model typically require modifications to application code. In the absence of a shared standard, large language models have no consistent mechanism to discover available capabilities or understand how to interact with them. This lack of standardization further increases cognitive load for developers and limits interoperability across systems.

The Model Context Protocol addresses these challenges by defining a standardized interface through which language models can interact with external tools, resources and services. By decoupling model reasoning from system-specific implementations, MCP allows integrations to be implemented once and reused across different models, applications and environments. This significantly reduces development effort while improving flexibility and long-term maintainability.
Beyond technical simplification, MCP enables a wide range of advanced AI-driven use cases. AI agents can access and combine information from multiple systems, such as calendars, document repositories or enterprise databases, to support more personalized and context-aware assistance.
From a stakeholder perspective, MCP provides benefits at multiple levels. Developers benefit from reduced integration complexity and faster iteration cycles. AI applications gain access to a growing ecosystem of standardized tools and data sources, enabling richer functionality. End users ultimately experience more capable and reliable AI systems that can access relevant information and perform meaningful actions on their behalf. In this sense, MCP serves as a foundational enabler for scalable, interoperable and production-ready AI-assisted systems. [7]

3.3 How MCP Works

MCP operates on a client-server architecture. The protocol defines two main roles:

- **MCP Server:** A server exposes tools, resources and capabilities through the MCP protocol. The server implements specific business logic, such as fetching data from a banking system, executing field extraction or validating information. The server listens for requests from MCP clients, processes them and returns structured JSON responses.
- **MCP Client:** A client (usually an AI application like Claude Desktop) connects to MCP servers and discovers what tools and resources are available. The client can then request execution of tools or retrieval of resources. The LLM using the client can see what capabilities are available and decide which to use in response to user prompts.

Communication between client and server follows a standardized message format, typically using JSON. Messages include requests for tool discovery, tool execution, resource retrieval and prompts for the LLM to reason about available capabilities. [7]

3.4 How LLMs Interact with MCP

An LLM interacts with MCP servers through its client application. The workflow is as follows:

1. **Discovery:** When initialized, the MCP client connects to one or more MCP servers and retrieves their available tools and resources. Each tool is described with its name, parameters and expected behavior.
2. **Awareness:** The LLM's context window includes descriptions of available MCP tools. The LLM can reason about which tools are relevant to the user's request.
3. **Tool Selection:** When the LLM determines that a tool would be useful, it generates a tool request in a structured format (e.g., JSON) specifying the tool name and parameters.

4. **Execution:** The MCP client intercepts the tool request, validates it against the tool's schema, sends it to the appropriate MCP server and receives the result.

5. **Integration:** The LLM incorporates the tool's response into its reasoning and generates a response to the user based on the returned information.

This loop continues as long as the user needs additional information or operations. The LLM can call multiple tools sequentially, building up context and refining its understanding of the user's request.

3.5 Key Capabilities of MCP

MCP servers can expose three main types of capabilities:

- **Tools:** Executable operations that the LLM can invoke. Tools should have well-defined inputs (parameters), outputs (results) and descriptions. Examples include fetching current case state, extracting structured data, validating information or submitting data to a backend system. Tools are synchronous and return concrete results.

- **Resources:** Stable, referenceable data that the LLM can access. Resources might represent documents, knowledge bases, database records or API responses. Unlike tools, resources are often read-only and represent information that should be retrieved and incorporated into the LLM's reasoning.

- **Prompts:** Pre-defined prompting templates that can be used to initialize LLM behavior. A server might expose prompts such as "Extract client information from text" or "Generate a compliance questionnaire." These enable consistency and best-practice sharing across applications.

3.6 Relevance to This Project

For this project, MCP is the ideal architectural foundation. Rather than building a custom integration between a chatbot and Atfinity's API, we expose custom operations as MCP tools that interact with Atfinity's CMS. This enables:

- Any MCP-compatible client (Claude Desktop, custom applications, etc.) to interact with the onboarding system without modification

- Clear separation between the AI application and Atfinity's CMS

- Extensibility: new tools can be added to the MCP server as capabilities evolve

By adopting MCP, the project ensures that the solution is not tightly coupled to a specific LLM or client, making it more maintainable and adaptable to future changes in tooling and requirements. Furthermore, different banking institutions can independently choose which LLM to use for their MCP server communication. Whether Claude, GPT-4, open-source models or locally-deployed solutions, without requiring any changes to the MCP server itself. This flexibility is critical in the market where different banks and companies have varying requirements, compliance constraints and strategic technology partnerships.

Chapter 4

Architecture

4.1 Preamble

For documenting our architecture diagrams we chose to use the C4 model. The C4 model is a framework for visualizing the architecture of software systems. It provides a hierarchical way to describe the system at different levels of detail, from high-level context diagrams to low-level component diagrams. Every part of the C4 model will be covered in the following sections, with comments about design choices where necessary - except the Code and Component Diagram. [4]

4.2 System Context Diagram

Our system context diagram provides a high-level overview of the system and its interactions. The component "Conversational AI System" encapsulates the MCP Server, LLMs and the Frontend.

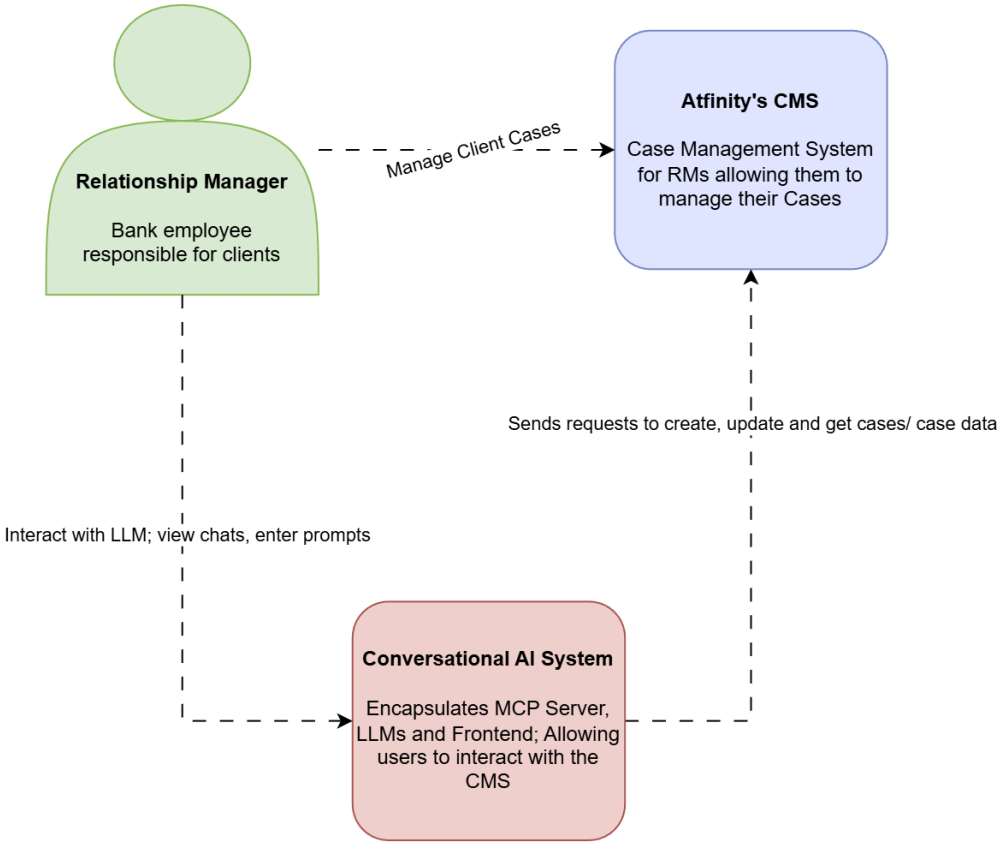


Figure 4.1: System Context Diagram

4.3 Overview

The software developed in this project is built around the Model Context Protocol. Because established architectural patterns for banking-specific implementations do not exist yet, traditional architecture approaches (such as strict layered architecture, microservice architecture or event-driven architecture) were not directly applicable.
Instead, a research-oriented and iterative development approach was adopted. The work started with a minimal viable architecture, which was progressively refined across successive iterations. Each iteration incorporated findings from manual and automated testing, newly identified system constraints and emerging insights into the behavior and capabilities of the Model Context Protocol.
The following sections concentrate on the final three architectural versions (v5, v6 and v7), which constitute the most mature and refined designs developed during the project. Nevertheless, the overall development process comprised several earlier iterations. These preliminary architectures are briefly described in Section 4.4 to provide contextual insight into the exploratory steps and incremental refinements that led to the final solutions.

4.4 Brief Descriptions of Architectures v1–v4

This section provides an overview of the first four architecture iterations, documenting the evolution of the system, key findings and the rationale for advancing to subsequent versions. Each version addressed specific challenges and yielded insights that informed the design of the final three architectures (v5, v6, v7) described in detail in subsequent sections.

4.4.1 Architecture v1: Foundational Exploration

Architecture v1 represented the simplest viable design for the project and served as an exploratory baseline. The system consisted of three layers: Atfinity's CMS, a locally-run MCP server and Claude Desktop as the UI/AI application.
The MCP server initially exposed a single tool for retrieving case information. This minimal implementation allowed the team to become familiar with the MCP protocol, understand how Claude Desktop interacts with MCP servers and explore the behavior of MCP. Development followed the official MCP specification, providing a solid foundation for understanding the protocol's capabilities and constraints. While functionally limited, v1 successfully demonstrated that MCP could effectively bridge an AI application and Atfinity's CMS.

4.4.2 Architecture v2: Tool Expansion and Architecture Migration

Architecture v2 expanded the MCP server with additional tools for case creation, case updates and case retrieval, broadening the system's capabilities. All tools relied on static field mappings and were therefore limited to a small, predefined subset of possible fields. A significant architectural change involved containerizing the entire system using Docker and replacing Claude Desktop with OpenWeb UI, a locally-deployable LLM interface. To enable integration with OpenWeb UI, the system transitioned from Anthropic's Claude API to OpenAI's GPT-4o-mini model, as OpenWeb UI does not provide native support for Anthropic models. [9]
However, this iteration yielded an unexpected and critical finding: Claude demonstrates substantially superior performance in MCP tool invocation and reasoning compared to OpenAI's models. The images below illustrate this disparity:

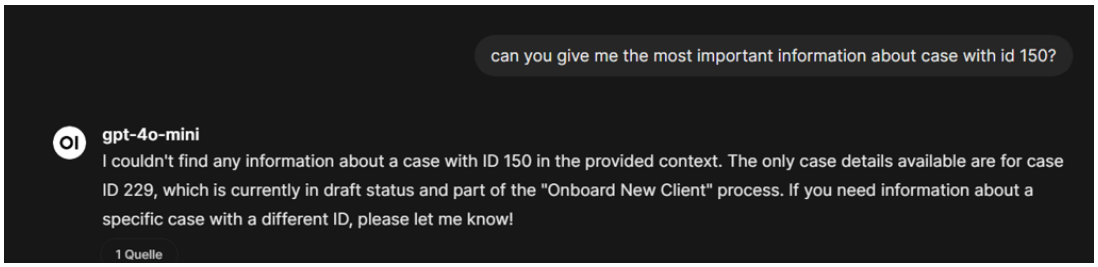


Figure 4.2: GPT-4o-mini fails to retrieve case information, reporting case 150 does not exist despite available case data

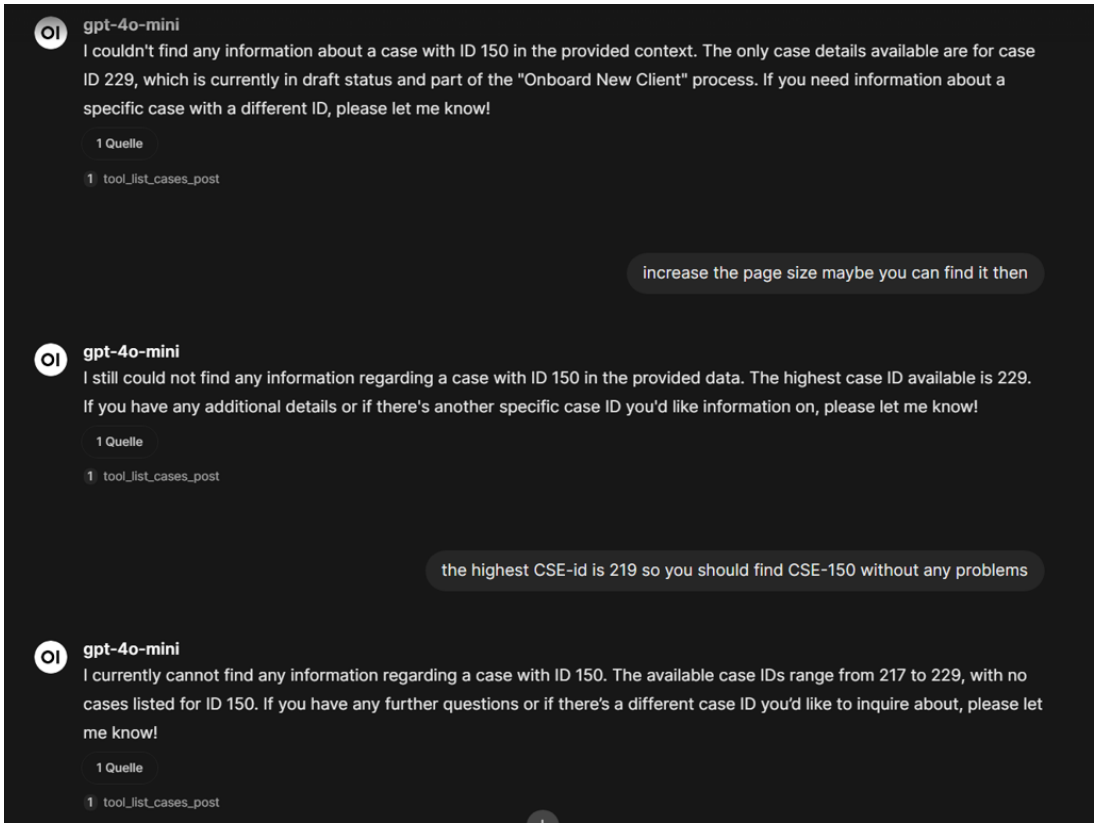


Figure 4.3: GPT-4o-mini continues to fail case retrieval even after user guidance, contrasting with available case IDs (217-229)

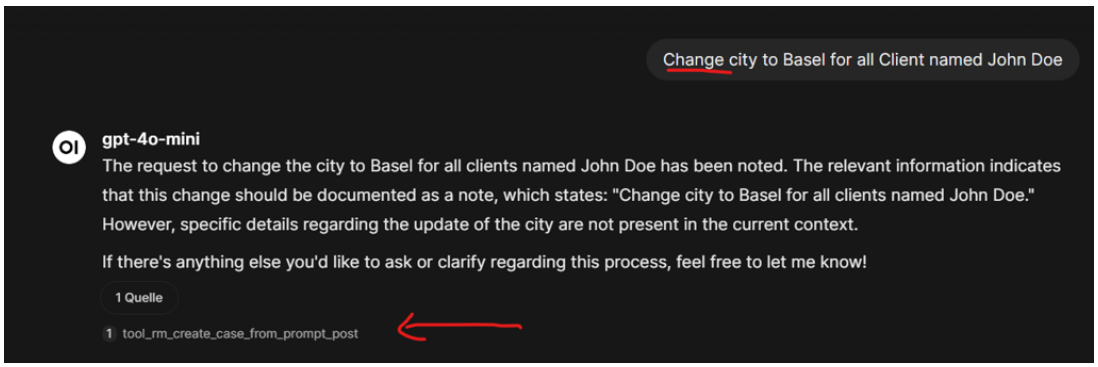


Figure 4.4: GPT-4o-mini attempts to invoke incorrect tools and fails to properly reason about available operations

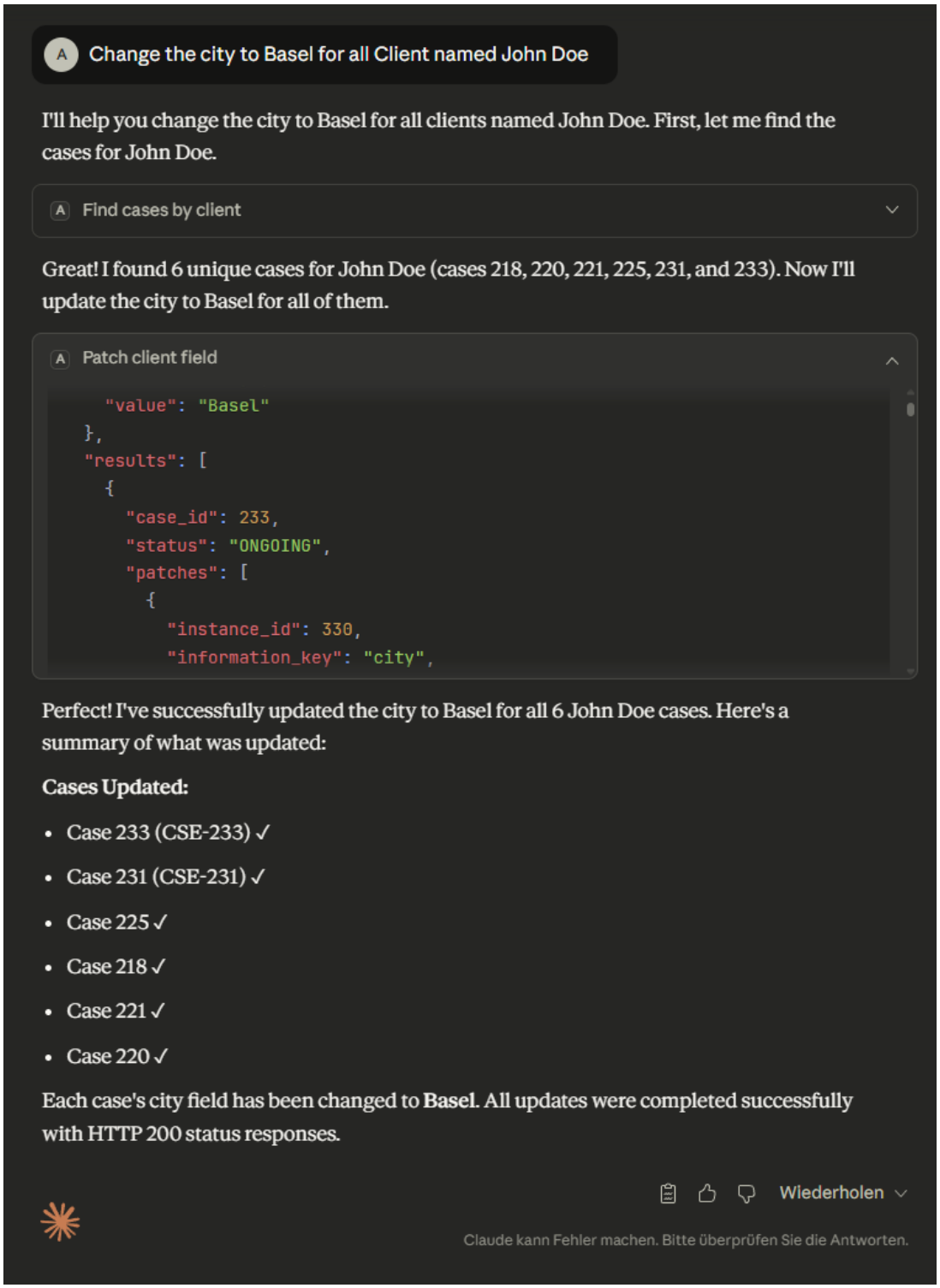


Figure 4.5: Claude successfully executes complex, multi-step operations: identifying cases by client name and atomically updating multiple cases with proper tool sequencing

These observations demonstrate that OpenAI's models frequently misunderstand tool schemas, invoke incorrect tools and fail at basic information retrieval tasks that Claude handles reliably. Claude exhibits superior semantic understanding of tool descriptions and makes deliberate, context-aware tool selections. This finding was instrumental in shaping subsequent architectural decisions.

4.4.3 Architecture v3: Return to Claude and Dynamic Field Mapping

Based on the findings from v2, architecture v3 abandoned containerization and returned to Claude Desktop as the primary interface. This decision was driven by the additional complexity introduced when integrating dockerized MCP servers with Claude Desktop. Establishing this connection requires multiple non-trivial steps, including enabling beta Docker features and publishing the MCP server image to the Docker catalog [5], which was considered disproportionate to the benefits at this stage of the project.

With a clearer understanding of MCP's behavior and the project's technical challenges, v3 focused on enabling dynamic field mapping between user prompts and Atfinity's CMS.
Two substantial technical challenges emerged:

1. **Process and Intent Resolution:** Atfinity's CMS organizes onboarding workflows as distinct processes, each with a unique process ID. The system must first determine which process the user intends to trigger before executing operations. To address this, a separate Python program was developed to retrieve available processes from the CMS and generate a YAML file. An LLM analyzed this configuration to create structured descriptions of each process and associated user intents, enabling accurate intent classification from natural language prompts.

2. **Field Mapping and Instance Resolution:** Atfinity's CMS employs an instance-based data model where information keys must be paired with specific instance identifiers to update values. Using Java as an example, properties of a class can only be assigned once an instance of the class has been instantiated. Critically, instances can only be read from cases already in draft status or created cases - they are not available beforehand. This constraint meant static field mapping was insufficient; the system required dynamic mapping, adapting to Aftinitiy's CMS behavior.

These constraints prevented simple prompt-to-JSON mapping and necessitated a more sophisticated approach in subsequent iterations.

4.4.4 Architecture v4: Incremental Processing with LLM-Based Extraction

Architecture v4 introduced two key innovations to address v3's limitations:

1. **Incremental MCP Server Behavior:** Rather than attempting complete case processing in a single step with predefined YAML mappings, v4 adopted an incremental architecture, adapted to Atfinity's CMS behaviour, where the MCP server processes case population and field filling in incrementations.

2. **LLM-Based Field Extraction and Mapping:** A dedicated LLM component was introduced specifically for field extraction and data mapping. This component gets sent available instances and fields, enabling accurate mapping of extracted user information to the appropriate case structure.

This architectural approach proved surprisingly effective. The system reliably extracted information, performed intelligent field mappings and maintained schema validity throughout processing. The success of v4 validated the core approach and shifted the project's focus toward optimizing quality metrics (documented here) and exploring architectural refinements.
Architecture v4 provided the technical foundation for the three final architectures (v5, v6, v7) discussed in the following sections. These final versions explore different communication flows, LLM integration patterns and performance optimization while maintaining the core incremental processing model.

4.5 Final Core Architecture – Shared Across All Three Final Prototypes

A critical objective for the final architecture was ensuring all components run within Docker containers to avoid local dependency management issues that plagued earlier iterations. LibreChat was introduced newly in the architecture - a self-hosted web UI that supports Anthropic's API while remaining fully containerizable. This design choice enabled the deployment of a containerized system without sacrificing the superior MCP performance demonstrated by Claude in earlier prototypes.
All three final architecture versions (v5, v6, v7) share the same foundational containerized infrastructure. Differences between versions lie in internal MCP server design and communication flow, discussed separately in subsequent sections. The container diagram below illustrates the shared system architecture.

4.5.1 Container Diagram

The C4 container diagram provides a detailed view of the system's components and their interactions. Rather than presenting only the application layer, we include the complete infrastructure stack to clearly communicate deployment requirements and runtime dependencies. This level of detail is critical for understanding how components communicate across processes and machine boundaries.

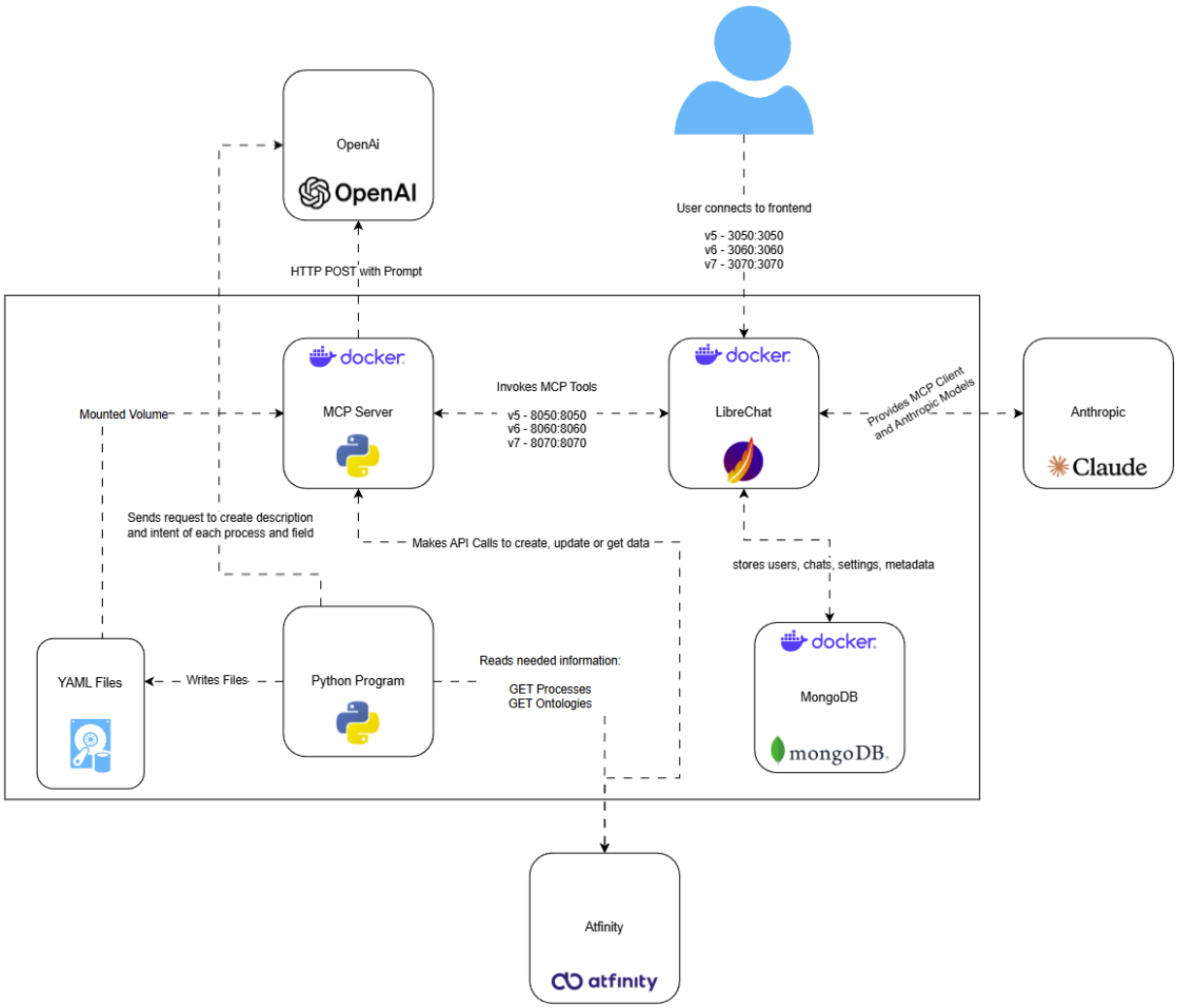


Figure 4.6: C4 Container Diagram - Final core architecture showing containerized components and infrastructure

Frontend – LibreChat

LibreChat is a self-hosted web UI for large language models that provides a ChatGPT-like user experience while maintaining full control over backend connectivity and data storage. In our architecture, LibreChat serves three critical functions:

1. **API Gateway:** LibreChat provides native support for Anthropic's Claude API, eliminating the need for custom middleware. Users interact with Claude through LibreChat's web interface, which handles API requests, streaming responses and conversation history.
2. **MCP Client Integration:** LibreChat natively integrates with MCP servers, automatically discovering and exposing available tools to Claude. This enables Claude to invoke MCP operations transparently during conversations.
3. **Data Persistence:** LibreChat maintains conversation history, user accounts, settings and other metadata in MongoDB.

Compared to OpenWeb UI (used in earlier prototypes), LibreChat offers Anthropic API support, more flexible customization and better containerization. OpenWeb UI requires workarounds to use Anthropic and demonstrated inferior integration with MCP client functionality. LibreChat's native Anthropic support and MCP-aware design made it the preferred choice for the final architecture. Librechat is also mentioned on a graphic in the official MCP documentation. [7]

MCP Server

The MCP server is the core component exposing Atfinity specific operations as MCP tools accessible to Claude. It implements all business logic for Atfinity's CMS integration through four primary MCP tools and 4 Classes.
The MCP server runs in a Docker container and exposes these tools on designated ports (v5: 8050/8050, v6: 8060/8060, v7: 8070/8070) for connection to LibreChat. All tools return structured JSON responses including status indicators, operation results and human-readable messages, conforming to the MCP specification and enabling user feedback.

A detailed implementation of the MCP server is documented in the section 5.2

Decoupled AI Component

Although Claude (Anthropic) serves as the primary reasoning model within the MCP server context, the system deliberately adopts a decoupled multi-model architecture. Specialized AI tasks are delegated to distinct components to improve modularity, flexibility and future extensibility.
In particular, an HTTP-based integration with the OpenAI API is used for tasks that require field mapping and data extraction logic. OpenAI models are responsible for:

- Mapping human-readable user prompts to structured field representations
- Determining the correct Atfinity process to trigger within the CMS
- Creating comprehensive user intents and descriptions for the helper YAML files (described in this section).

This separation ensures that AI reasoning, extraction and intent detection are not tightly coupled to a single model or provider. The MCP server orchestrates these components transparently, allowing individual models to be replaced or compared without modifying core system logic.

Due to project time constraints, a comprehensive quantitative comparison between Claude, GPT-models and other models was not conducted.

Helper Python Program

A python program that generates YAML files describing available Atfinity processes, ontologies and fields within a banks CMS. This program retrieves process and field definitions from Atfinity via API, enriches them with semantic descriptions and stores them as structured YAML. The MCP server reads these files

and provides them as a guide/resource for OpenAi to map fields correctly.

A detailed implementation is described in this section: 5.1

Persistent Storage – MongoDB

MongoDB serves as the data layer for LibreChat, storing conversation histories, user credentials and system metadata.

Backend Integration – Atfinity's CMS

Atfinity's case management system remains the authoritative source of truth for client onboardings. The MCP server communicates with Atfinity via its REST API to retrieve case data, submit updates and create new cases.

4.5.2 First final Architecture v5 – Iterative LLM Mapping

Architecture v5 implements a fully iterative and adopted LLM-based field mapping strategy to Atfinity's CMS. It does not rely on pre-generated field mappings.

The workflow operates as follows:
When a user enters a prompt requesting case creation, Claude analyzes the request and invokes the `trigger_case_creation` MCP tool. The MCP server reads the process YAML file (generated beforehand by the Python helper program) and sends both, the available processes and the user prompt to OpenAI for intent classification. OpenAI identifies the correct process ID and returns it to the MCP server. The server then creates a blank case in Atfinity's CMS with this process ID, receiving a JSON payload containing all exposed fields for the current case.
The server identifies fields that are unfilled or contain only default values and sends these available fields including system constraints like permitted values along with the original user prompt to OpenAI for semantic mapping. OpenAI extracts relevant information from the user prompt and maps it to the available fields, returning a structured JSON mapping. The MCP server processes and applies this mapping to the case via PATCH requests to Atfinity's API.
Critically, after each field update, Atfinity's conditional logic may expose new fields based on the values just submitted. The MCP server detects these newly available fields and repeats the mapping and update cycle. This iteration continues until no new fields emerge, at which point the operation completes and results are presented to the user.

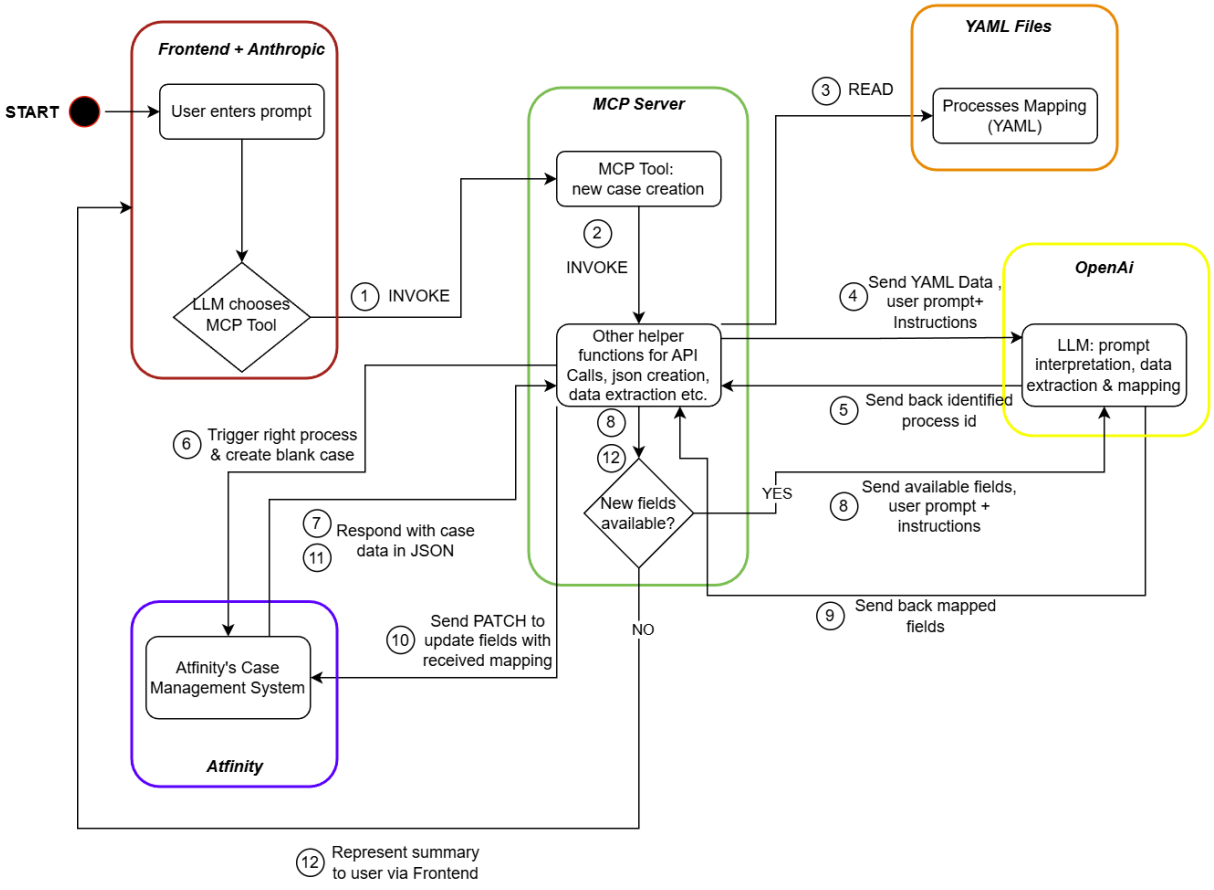


Figure 4.7: Visualized Flowchart of Architecture v5 (Scenario: Create Case)

4.5.3 Second final Architecture v6 – Iterative Mapping with Preemptive Mapping

Architecture v6 combines the iterative approach of v5 with an optimization strategy: preemptive LLM mapping. The system generates a comprehensive field-to-value mapping upfront, then uses iteration as a fallback mechanism only when the cache content is not hit.
The workflow begins identically to v5: user prompt → Claude invokes tool → read process YAML. However, v6 additionally reads an ontology mapping YAML file containing descriptions and constraints for all fields and ontologies in the corresponding CMS. The MCP server sends both the user prompt and the available fields from the YAML to OpenAI, which generates a preemptive mapping.
The server creates the blank case and identifies available fields. For each available field, it first checks the preemptive cache. If the field exists in the cache, the cached value is used directly without additional LLM invocation. If the field is not in the cache, the system falls back to the iterative approach: it sends the unmapped field and user prompt to OpenAI for on-demand extraction. The server applies all available mappings (cached and newly extracted) to the case.
After each update, the server checks for newly exposed fields and repeats the process: attempt to satisfy from cache first, fall back to OpenAI if needed. This continues until no new fields appear.

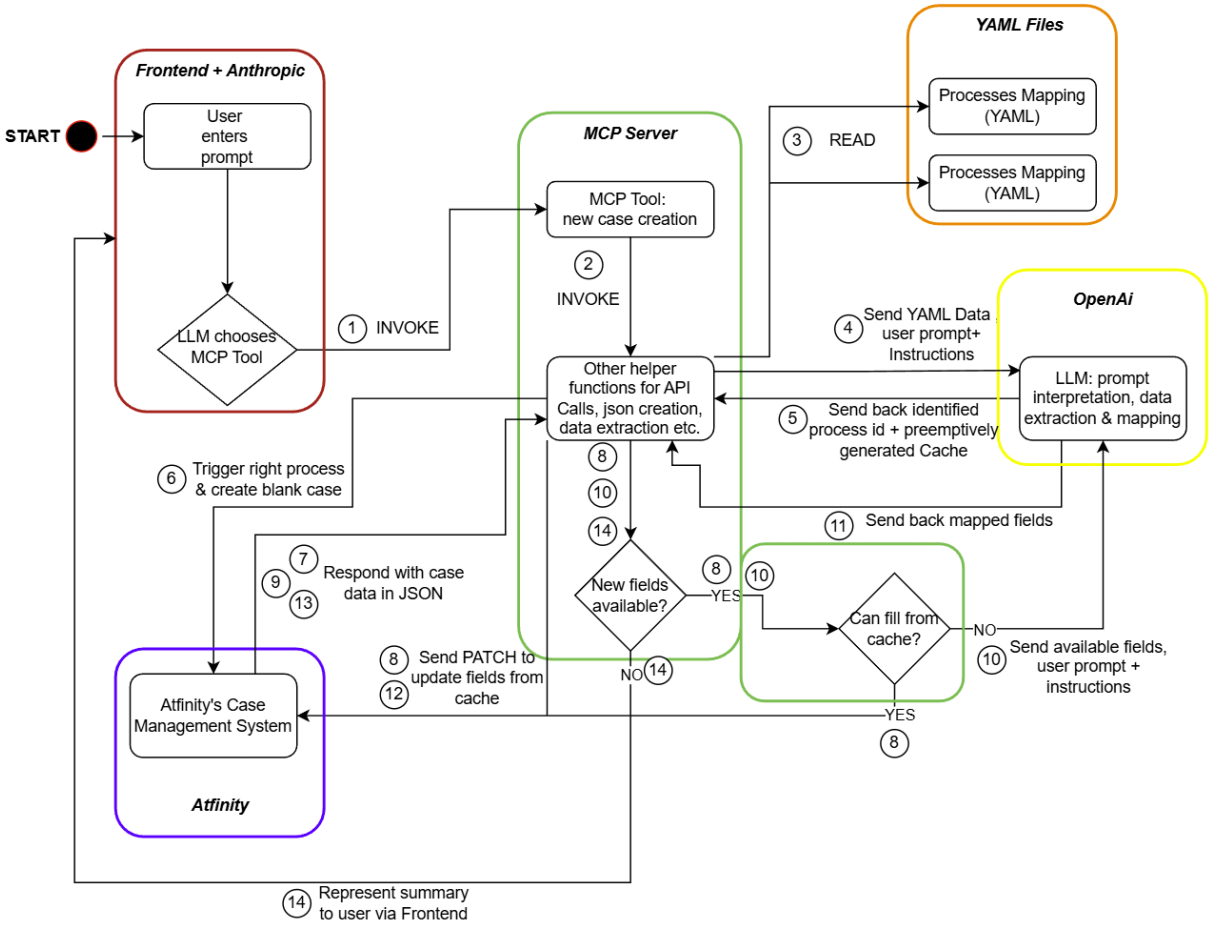


Figure 4.8: Visualized Flowchart of Architecture v6 (Scenario: Create Case)

4.5.4 Architecture v7 – Preemptive Caching Without Iteration

Architecture v7 pursues maximum optimization by eliminating the iterative mechanism entirely. All field extraction occurs in a single preemptive LLM call, after which the case is populated exclusively from the generated cache.

The workflow initializes similarly to v6: read process YAML and ontology YAML, send user prompt and fields to OpenAI and generate preemptive mapping. OpenAI performs semantic analysis of the user prompt and extracts all mappable values based on the ontology YAML, returning a comprehensive JSON mapping.

The server creates the blank case in Atfinity's CMS and receives the initial set of available fields. The server identifies available fields and attempts to populate them exclusively from the preemptive cache. If a field exists in the cache, its value is applied. If a field does not exist in the cache, it is skipped resulting in no additional LLM calls to extract relevant data.

After no more values can be applied from the cache the process ends and a summary is presented to the RM.

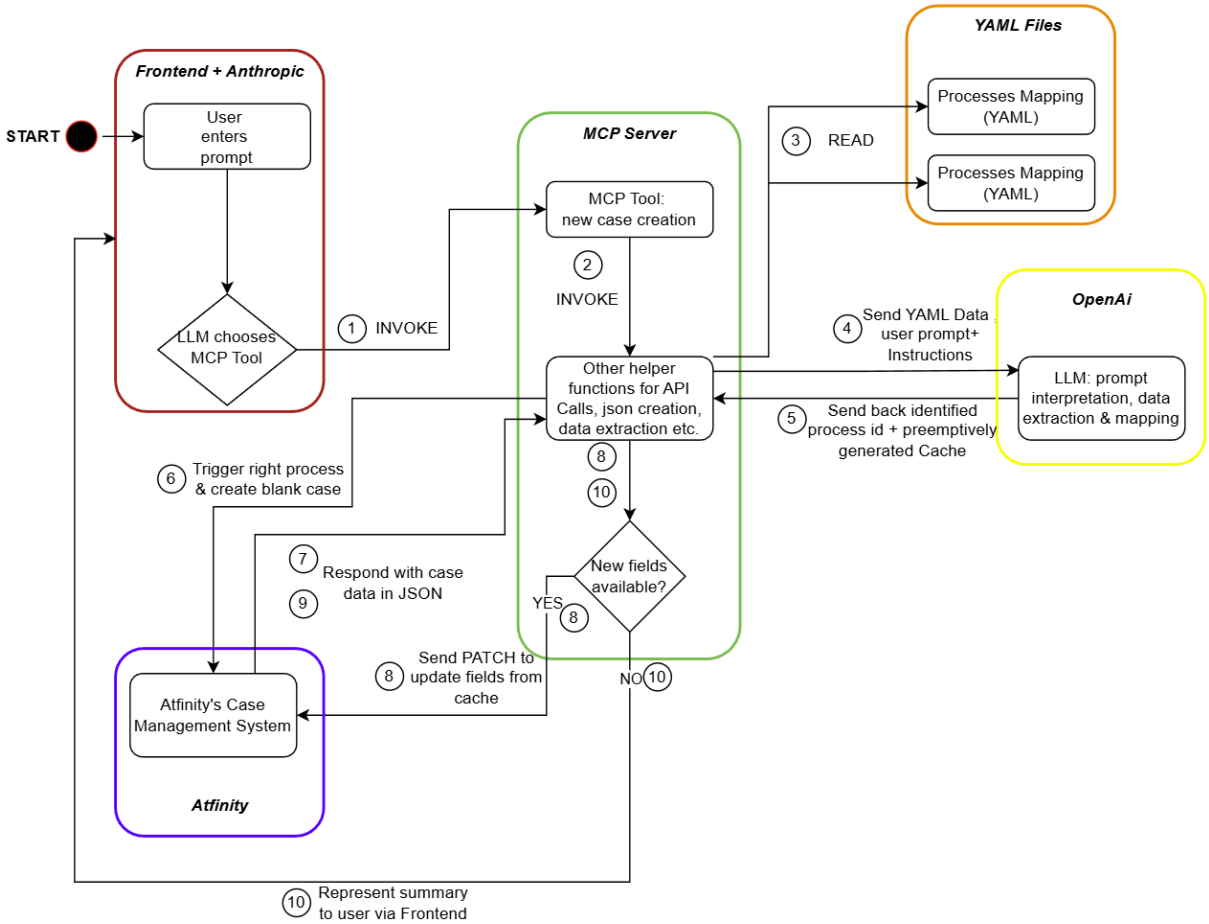


Figure 4.9: Visualized Flowchart of Architecture v7 (Scenario: Create Case)

4.6 Architectural Decision Records (ADRs)

Architectural decisions throughout this project were formed by manual and automated testing with predefined user scenarios. Rather than committing to a fixed design upfront, we adopted an iterative refinement process where each architecture version was evaluated against expected behavior. Findings from testing were analyzed and insights directly shaped the next iteration.

4.6.1 Test-Driven Architecture Evolution

Predefined user scenarios provided concrete test cases representing typical Relationship Manager workflows. These scenarios included:

- Creating new client cases from natural language descriptions
- Updating existing cases with additional information
- Querying case information and retrieving specific data

For each scenario, test prompts and corresponding expected behaviors were defined. These prompts were then executed, either manually or automated, against the current architecture version and the system's actual behavior was observed and evaluated based on the outcomes.

The detailed test concept including test prompts and scenarios is documented in this section: 5.3

4.6.2 ADR01: Exploratory and Iterative Architecture Approach

ADR: ADR01
Author: Arnel Veladzic
State: Accepted
Related: Architecture v1

Decision Record

At the beginning of the project, the problem domain and technical constraints of integrating large language models with Atfinity's CMS were not yet fully understood. In addition, no established state-of-the-art reference architectures existed for this type of LLM-driven, MCP-based integration. Conventional software engineering approaches based on upfront design proved infeasible due to the system's highly dynamic and non-deterministic behavior. Rather than committing to a complex architecture upfront, the decision was made to adopt an exploratory and iterative approach.
As a result, the project intentionally started with a minimal playground, consisting of three layers: Atfinity's CMS, a locally executed MCP server and Claude Desktop serving as the MCP client.

Comments

This simple three-layer architecture enabled rapid experimentation and reduced technical overhead during the early stages of the project. It provided a controlled environment to explore MCP communication patterns, tool invocation behavior and CMS integration without introducing premature complexity.

- Early validation of core assumptions
- Reduced risk of premature architectural commitment
- Clear baseline for later iterations

4.6.3 ADR02: Deployment Tools

ADR: ADR02
Author: Arnel Veladzic
State: Accepted
Related: Architecture v1, v2

Decision Record

During the transition from architecture v1 to v2, the system was containerized to address portability and runtime inconsistencies observed across different desktop environments. Variations in local setups, dependencies and operating systems led to unstable behavior and increased setup effort. To ensure a reproducible and portable deployment, the decision was made to dockerize the environment.
Docker Compose was selected as the deployment tool to orchestrate the containerized components. More complex orchestration frameworks were intentionally excluded in order to maintain a simple and low-overhead deployment suitable for rapid prototyping.

Comments

As part of the containerization effort in architecture v2, the frontend was replaced with OpenWeb UI, a locally deployable LLM interface. This change required the use of a different large language model, as OpenWeb UI does not provide native support for Anthropic's Claude models. Consequently, the system was temporarily migrated to OpenAI's GPT-4o-mini to maintain compatibility with the selected frontend.

4.6.4 ADR03: LLM Model Selection

ADR: ADR03
Author: Arnel Veladzic
State: Accepted
Related: Architecture v2, v3, ADR02

Decision Record

In the context of selecting a large language model for MCP tool invocation and semantic reasoning, the project initially evaluated OpenAI's GPT-4o-mini and Anthropic's Claude models. Manual testing revealed systematic failures in tool invocation, reasoning accuracy and case retrieval when using GPT-4o-mini. As a result, the decision was made to adopt Claude as the primary LLM for all subsequent architectures.

Comments

While GPT-4o-mini provided easier integration with containerized UI solutions, its limitations in MCP tool understanding posed unacceptable risks for correctness and reliability. Claude demonstrated superior semantic understanding, correct tool selection and robust multi-step reasoning.
The decision to favor Claude led to architectural adjustments, including abandoning containerization strategies that depended on OpenAI compatibility and adopting LibreChat to preserve Claude support.

- Higher tool invocation accuracy
- Superior semantic reasoning
- Reliable multi-step workflow execution
- Reduced risk of incorrect API calls

4.6.5 ADR04: Temporary Abandonment of Containerization

ADR: ADR04
Author: Arnel Veladzic
State: Accepted
Related: Architecture v3, ADR02

Decision Record

Based on the findings from architecture v2, the decision was made to abandon containerization in architecture v3 and return to Claude Desktop as the primary interface. While containerization improved portability, integrating dockerized MCP servers with Claude Desktop would have introduced significant complexity.
Establishing a connection between a containerized MCP server and Claude Desktop required multiple non-trivial steps, including enabling beta Docker features and publishing the MCP server image to the Docker catalog, which was considered disproportionate to the benefits at this stage of the project.[5]

Comments

This decision prioritized development velocity during an exploratory phase of the project. By reverting to a non-containerized setup, the focus could remain on resolving core architectural challenges related to intent resolution, field mapping and incremental processing.

- Faster iteration during exploratory development
- Deferred containerization to later project phases

4.6.6 ADR05: Incremental Field Processing

ADR: ADR05
Author: Arnel Veladzic
State: Accepted
Related: Architecture v3, v4

Decision Record

During early iterations, static field mapping approaches failed due to Atfinity's fields that are referenced by instances and where field visibility depends on previously submitted information. Testing, as well as formal clarification with Atfinity's revealed fields and the corresponding instances only become available after case creation.
To address this constraint, the system adopted an incremental processing model in which case fields are retrieved and populated iteratively after each update.

Comments

This decision aligned the system's behavior with Atfinity's CMS architecture. Incremental processing allowed the system to react dynamically to newly exposed fields and ensured schema validity throughout the workflow and provided the fundamental architecture for the final prototypes.

- Compatibility with Atfinity's architecture
- Enable dynamic mapping
- Foundation for later performance optimizations

4.6.7 ADR06: Intent Resolution and Process Mapping

ADR: ADR06
Author: Arnel Veladzic
State: Accepted
Related: Architecture v3, v4

Decision Record

Atfinity's CMS organizes onboarding workflows into distinct processes, each identified by a unique process ID. Early designs relied on hard coded processes, which proved unreliable. To ensure correct routing of requests, intent resolution and process mapping were separated into a dedicated preprocessing step.
A helper Python program was introduced to retrieve available processes from the CMS and generate structured YAML descriptions of available processes for LLM-based intent classification.

Comments

This decision significantly improved intent classification accuracy and ensured that the correct workflows were triggered based on natural language input.

- Improved intent classification reliability
- Explicit mapping between prompts and processes
- Reduced risk of incorrect workflow execution
- Clear separation of responsibilities

4.6.8 ADR07: LLM-Based Dynamic Field Extraction

ADR: ADR07
Author: Arnel Veladzic
State: Accepted
Related: Architecture v4, ADR05

Decision Record

To overcome the limitations of static YAML-based field mappings, a dedicated, decoupled LLM component was introduced for field extraction and mapping. This component receives the currently available fields and instances from the CMS and maps extracted user information dynamically.

Comments

This approach enabled accurate handling of free-text input, variations in phrasing and partial information. It proved essential for maintaining schema validity while supporting flexible natural language interaction.

- Dynamic adaptation to available CMS fields
- Enabled incremental processing architecture

4.6.9 ADR08: Containerized UI with LibreChat

ADR: ADR08
Author: Arnel Veladzic
State: Accepted
Related: Core Architecture, ADR02

Decision Record

During the development of architecture v5, the decision was made to reintroduce containerization while preserving compatibility with Anthropic's Claude models. Earlier containerization attempts in architecture v2 required a migration to OpenAI models due to frontend limitations.
To reconcile containerization with Claude support, the frontend was switched to LibreChat, a container-friendly LLM interface that offers native support for Anthropic models. This enabled the system to combine the portability benefits of Docker with the previously identified advantages of using Claude.

Comments

LibreChat allowed the architecture to remain fully containerized while avoiding the model compatibility constraints encountered with OpenWeb UI. This change enabled consistent runtime environments across development machines and restored the use of the preferred LLM without reintroducing unnecessary deployment complexity.
The decision balances portability and reproducibility with model quality, accepting the additional configuration effort required to integrate LibreChat in exchange for improved extraction accuracy and reasoning reliability.

- Restored compatibility with Claude in a containerized setup
- Improved portability and reproducibility of the system
- Avoided limitations imposed by earlier frontend choices
- Supported further architectural experimentation in v6 and v7

4.6.10 ADR09: Preemptive Caching Strategy

ADR: ADR09
Author: Arnel Veladzic
State: Accepted
Related: Architecture v6, Architecture v7

Decision Record

Testing of architecture v5 revealed that repeated LLM invocations during incremental processing significantly increased latency for complex onboarding scenarios. To mitigate this effect, a preemptive caching strategy was introduced to generate and reuse field mappings where possible.

Comments

Preemptive caching reduced the number of required LLM calls and improved overall response times, while fallback mechanisms preserved correctness when cached data was insufficient.

- Reduced latency for complex cases in v7
- Basis for architecture variants v6, v7

4.6.11 ADR10: Combined Preemptive and Iterative Mapping Strategy

ADR: ADR10
Author: Arnel Veladzic
State: Accepted
Related: Architecture v6, ADR09

Decision Record

During the development of architecture v6, a combined strategy was adopted that integrates preemptive caching with iterative LLM-based field mapping. The MCP server first attempts to populate fields using precomputed mappings whenever a response from Atfinity's CMS is received. For all fields that cannot be filled from the cache, the system falls back to the iterative LLM-based extraction and mapping process. This hybrid approach was chosen with the goal to gain a performance boost in terms of latency.

Comments

Testing revealed that the fallback mechanism introduces additional latency in scenarios where cached mappings are incomplete. Each fallback requires an additional LLM invocation and synchronous waiting for the result before continuing the workflow. As a result, the overall end-to-end processing times increase, especially for complex onboarding cases with many fields that cannot be resolved from the cache alone. Despite the increased latency, the decision was accepted in favor of improved completeness and robustness. The architecture prioritizes correctness and field coverage over minimal response time when preemptive mappings are insufficient.

- Improved completeness through fallback to iterative mapping
- Increased latency due to additional LLM invocations
- Deterministic handling of cache misses
- Clear separation between fast-path (cache) and slow-path (LLM) processing

4.6.12 ADR11: Elimination of Iterative Fallback for Performance Optimization

ADR: ADR11
Author: Arnel Veladzic
State: Accepted
Related: Architecture v7, ADR10

Decision Record

The introduction of a combined preemptive and iterative mapping strategy in architecture v6 aimed to reduce overall processing latency while maintaining high field completeness. However, measurements showed that the expected performance improvements were not achieved.
As a result, architecture v7 eliminated the iterative fallback mechanism entirely. In this design, the system relies exclusively on preemptively computed field mappings. No additional LLM calls are performed during case processing. Communication is limited to interactions between the MCP server and Atfinity's CMS, where all fields that can be populated from the precomputed mapping are filled and all remaining fields are intentionally skipped.

Comments

This decision prioritizes the latency of the overall process. By removing synchronous LLM calls from the critical path, architecture v7 achieves lower and more stable end-to-end latencies. The trade-off is an increased likelihood of unpopulated fields.
The decision was accepted in favor of improved latencies, while still maintaining completeness and robustness.

- Elimination of runtime LLM dependencies
- Reduced processing latency
- Simplified control flow

Chapter 5

Implementation

5.1 Helper Python Program

The helper Python program consists of three separate Python modules. The core module, `mapping.py`, serves as a shared library and provides essential functionality such as authentication and API request handling. Two additional modules, `ontology_mapping.py` and `processes_mapping.py`, are built on top of this core library.
Both additional modules generate YAML-based helper resources that are used as contextual input for the large language model (LLM). YAML was chosen because of humand readability which is essential for such helper resources. The process mapping module primarily supports intent detection. Since multiple processes can be triggered within Atfinity's CMS, each identified by a unique ID, this mapping ensures that the LLM consistently selects the correct process based on the user's intent.
The ontology mapping module provides structured information about the objects and fields available within the CMS. An ontology can be compared to a Java class, where the fields represent the attributes of that class. Multiple objects may share the same set of fields. This information is exported in YAML format and includes details such as field identifiers and allowed values. The mapping is a central element of the preemptive cache generation mechanism, as it enables the LLM to map user prompts in advance to all existing and valid fields within the system.

The process mapping workflow is implemented as follows:

1. A login request is simulated against Atfinity's CMS, as the required API endpoints are not publicly accessible.
2. The relevant endpoint (`/processes`) is queried using a GET request.
3. Atfinity's CMS responds with a JSON payload containing the required process data.
4. The program extracts relevant information such as process keys, names and IDs and forwards this data to the decoupled AI component (OpenAI) together with a structured prompt.
5. The AI component generates descriptive text for each process and produces a list of possible user intent variations associated with each process.

```yaml
processes:
- key: migrate_existing_client
  id: 5
  display_name: Migrate Existing Client
  description: The 'Migrate Existing Client' process facilitates the seamless transition
    of client data and case details from one system or platform to another, ensuring
    continuity of service and accurate record-keeping.
  intents:
  - Migrate a client to the new system
  - Transfer existing client data
  - Update client information in the system
  - Move client records to the new platform
  - Initiate client migration process
```

Figure 5.1: Content of the processes YAML file

The ontology mapping workflow is implemented as follows:

1. A login request is simulated against Atfinity's CMS because the required API endpoints are not publicly accessible.
2. The relevant endpoints (`/ontologies` and `/informations`) are queried using HTTP GET requests.
3. Atfinity's CMS returns a JSON response containing the required ontology and information data.
4. The program extracts and combines the ontology and information data. The processed data is then forwarded to the decoupled AI component (OpenAI) together with a structured prompt.
5. The AI component generates descriptive text for each ontology, which serves as contextual information to help the LLM better interpret and map user information.

```yaml
ontologies:
- key: Contract
  id: 14
  display_name: Contract
  description: The "Contract" ontology represents the formal agreements between parties
    within the banking and financial context, encompassing essential details such
    as terms, conditions, and obligations that govern the relationship and transactions
    between clients and financial institutions. Its purpose is to facilitate effective
    management, tracking, and compliance of contractual obligations throughout the
    case management process.
  icon: ledger
  roles: []
  number_of_information: 10
  allow_linking: true
  informations:
  - key: account_type
    id: 164
    display_name: Account Type
    information_type: enum
    optionality: not_optional
    is_calculated: false
    possible_values:
    - key: individual_account
      label: Individual Account
    - key: entity_account
      label: Entity Account
```

Figure 5.2: Content of the ontology YAML file

5.2 MCP Server

The MCP Server acts as the central orchestration layer between the LLM and Atfinity's CMS. Its primary responsibility is to enable communication between the LLM and Atfinity's CMS. The server is composed of four core classes that implement the system logic and four MCP tools that serve as controlled entry points for the LLM.

5.2.1 Core Classes

The internal logic of the MCP Server is implemented in four main classes. Each class has a clearly defined responsibility and encapsulates a specific concern within the overall workflow.

- **ProcessesManager**: This class is responsible for loading, validating and managing the process mappings defined in a YAML configuration file. It provides access to all available processes and supports lookup operations by process key or process identifier. This class ensures that process selection remains consistent and independent from data extraction and field mapping logic.
- **CaseAPI**: This class encapsulates all communication with Atfinity's CMS REST API. It provides low-level methods for creating new cases, retrieving case details and updating individual case fields. All HTTP requests, authentication headers, payload construction and error handling are implemented within this class. This abstraction isolates future API changes.
- **LLMFieldExtractor**: This class is responsible for generating structured prompts and invoking the LLM to extract relevant field values from natural language input. It enforces strict extraction rules by embedding dynamically field data, allowed values and type constraints directly into the prompts. The class supports both case creation and case update scenarios and validates the LLM responses by parsing and normalizing the returned JSON structures.
- **IncrementalCaseFiller**: This class coordinates the complete workflow of process determination, case creation, field extraction and iterative field population. It repeatedly queries the CMS for newly available fields and triggers further extraction rounds until no additional fields can be populated. This iterative design aligns with Atfinity's dynamic and conditional CMS behavior.

5.2.2 MCP Tools

The MCP Server exposes four MCP tools, which represent the only interfaces directly accessible to the LLM. These tools are intentionally kept minimal and contain no business logic. Instead, each tool delegates all processing to the underlying classes and methods.
A key aspect of the MCP tools is their detailed textual descriptions. These descriptions serve as reasoning guidance for the LLM, enabling it to select the correct tool based on user intent. The quality and precision of these descriptions are critical for correct tool selection.

```python
@mcp.tool()
def update_case_fields(case_id: int, user_prompt: str) -> str:
    """
    Update existing case fields based on user's update request.

    Use this tool when user wants to change, modify, or update specific fields in an existing case.
    Examples:
    - "Change surname to Doe for case 13"
    - "Update email and phone number for case 25"
    - "Set wealth to 100000 for case 42"

    Args:
        case_id: The case ID to update
        user_prompt: User's request describing which fields to update and their new values

    Returns:
        JSON string with update results including successfully updated fields and any failures
    """
```

Figure 5.3: MCP tool description

- `trigger_case_creation`: Creates a new onboarding case and intelligently populates fields from user-provided text. The tool extracts relevant information from the user's prompt and maps it to Atfinity's case structure automatically filling applicable fields. This enables rapid case creation without requiring manual field-by-field data entry. Internally, this tool invokes the incremental case filling workflow, including process detection, case initialization and iterative field population.
- `get_case_information`: Fetches comprehensive information about a specific onboarding case from Atfinity. This tool returns all available data for a case, enabling Claude to answer user questions and provide case summaries. Users can request specific information (e.g., "What is the client's name for case 12?") or ask for full case overviews.
- `get_case_still_open_available_fields`: Retrieves the set of fields in a case that have not yet been filled, providing the Relationship Manager with a clear overview of incomplete data without requiring a context switch to the Atfinity CMS. The RM can then directly request Claude to extract and populate the remaining fields based on information the client provides. This tool supports follow-up interactions where additional user input is required to complete a case.
- `update_case_fields`: Updates specific fields in an existing case based on user-provided modification requests. Users can specify field changes (e.g., "Change surname to Doe for case 13") and the tool intelligently maps user intent to the appropriate Atfinity API calls. The tool extracts only explicitly requested changes and applies them via controlled PATCH requests.

5.2.3 Design Guidelines

To ensure consistency and maintainability, the MCP Server follows strict coding conventions. Class names use *PascalCase*, while function and method names follow *snake_case*. This convention improves readability and aligns with common Python best practices.

5.3 Test Concept

The testing strategy for the MCP-based client onboarding system is designed to address the challenges of validating AI-driven field extraction, where deterministic unit testing is insufficient due to the stochastic nature of Large Language Models.

5.3.1 Testing Scope and Limitations

Traditional software testing methodologies, such as unit tests with fixed inputs and expected outputs, are inadequate for LLM-based systems, where the same prompt may yield semantically equivalent but syntactically different responses across multiple executions. Furthermore, the distributed nature of the architecture (MCP server, Atfinity API, LLM client) introduces additional variability in response times and output formatting.
The test concept therefore focuses on:

- **End-to-end functional validation**: verifying that natural language prompts result in correct API calls and field assignments
- **Field extraction accuracy**: measuring how reliably the system maps unstructured input to structured JSON fields
- **Performance benchmarking**: quantifying latencies and cache efficiency to assess operational viability

Unit-level testing of individual MCP tools is performed manually during development but is not part of the automated test suite due to the dependency on external LLM services and the non-deterministic nature of their outputs.

5.3.2 Automated Test Case Design

Test cases are structured as JSON documents containing:

- **Test ID and metadata**: unique identifier, difficulty classification and expected execution duration

- **Tool invocation**: the MCP tool to be called (e.g., `trigger_case_creation`)
- **User prompt**: natural language input simulating a Relationship Manager's instruction
- **Expected output**: a list of fields with their anticipated values and match strategies

Four primary test scenarios were defined to cover the spectrum of client onboarding complexity:

Test Case 1: Simple EUR Individual Account A minimal individual account with basic personal information (name, citizenship, expected AUM in EUR). This test validates baseline extraction capability for straightforward prompts with limited ambiguity. Expected fields: 13.

Test Case 2: Simple CHF Individual Account Similar to Test Case 1 but denominated in Swiss Francs, testing currency handling and basic personal data extraction. Expected fields: 14.

Test Case 3: Entity Account with Corporate Details A corporate account for TechCorp AG with company-specific fields such as `name`, `founding_date` and `source_of_wealth`. This test assesses the system's ability to distinguish between individual and entity account structures. Expected fields: 20.

Test Case 4: Complex HNWI with Political Affiliation A high-net-worth individual (Marcus Hoffmann) with detailed wealth breakdown, multiple currencies, political party membership and extensive KYC requirements. This test represents the upper boundary of extraction complexity and evaluates the system's handling of multi-faceted, information-dense prompts. Expected fields: 16.
Each test case includes provisions for flexible matching where exact string equivalence is not semantically required (see Quality Measures below).

5.3.3 Automated Test Execution

The test framework is implemented as a Python script (`test_mcp_server.py`) that:

1. Connects to the MCP server via the FastMCP client library
2. Loads test cases from a JSON configuration file
3. Invokes the specified MCP tool with the provided user prompt
4. Parses the response, which contains fields extracted from both cache and LLM inference
5. Compares actual field values against expected values
6. Calculates precision, recall and F1 scores (detailed below)
7. Outputs a comprehensive JSON report with per-field comparison results

The script supports command-line arguments for specifying test case files and output destinations:

```
python test_mcp_server.py --test-cases cases.json --output results.json
```

5.3.4 Response Format and Field Extraction

The MCP server returns responses in a structured format containing:

- **Process metadata**: case ID, confidence score, reasoning
- **Preemptive extraction**: fields cached from initial ontology-based parsing (architectures v6 and v7)
- **Rounds**: iterative extraction cycles, each containing:
  - `fields_filled_from_cache`: values retrieved from preemptive extraction (v6, v7)
  - `fields_filled_from_llm`: values requiring LLM inference
  - `cache_misses`: fields that could not be satisfied by cache alone (v6, v7)

The test framework merges all fields across rounds, unwraps nested `{label, value}` structures returned by the API and filters out empty values (empty strings, empty arrays, `null`) before comparison. This normalization ensures that only semantically meaningful fields are evaluated.

5.3.5 Manual Test Case Design

Test cases are structured as RM scenarios definid an objective, prompt and expected behavior of the system:

Test Case 1 – Case Creation from Unstructured Input

Objective: Verify that the LLM can correctly extract client data from a single unstructured prompt and that the MCP successfully creates a new case and applies the extracted values using Atfinity's API.

Input (Prompt Examples):

- *"Create a new onboarding case for John Doe. He is single, lives at Bahnhofstrasse 1, 8001 Zürich, earns about 120 000 CHF per year and his source of wealth is a regular income. Assign it to admin@atfinity.ch"*
- *"Create a case for John Doe. Single, roles AccountHolder and BeneficialOwner. Total wealth 2,502,050 CHF, source of wealth company sale and other 'Found it on the street'. Account type individual. Owner email admin@atfinity.ch. Name it 'Test for Mikkel'."*

Expected Behavior:

- The LLM classifies the prompt as a case creation intent.
- The MCP sends a `POST /api/1/cases` request with a valid `process_id`.
- Field values such as `first_name`, `last_name`, `address` and `income` are correctly populated in Atfinity.
- A summary of what happened is represented via the UI to the RM.

Test Case RM 2 – Query About Existing Case

Objective: Assess whether the LLM can distinguish an informational query from data input and correctly retrieve the requested case using the appropriate API endpoint.

Input (Prompt Example):

- *"Can you tell me the total wealth and current process stage of John Doe's case?"*

Expected Behavior:

- The LLM classifies the intent as a query and invokes the right MCP tool.
- The MCP server retrieves the corresponding case ID and its data.
- The response is summarized and returned to the relationship manager in natural language (e.g., "John Doe's case is currently in Stage 3: Compliance Review, with a total wealth of 2.5 million CHF.").

Test Case 3 – Mixed Intent Prompt

Objective: Verify that the system can correctly detect and handle multiple intents within a single prompt by executing the required update and query actions in the correct order.

Input (Prompt Example): *"John Doe moved to Basel. Please update his address and also tell me what his current wealth is."*
Expected Behavior:

- The LLM identifies both an update intent and a query intent within the same prompt.
- The MCP first invokes the right MCP tool to update the client's address.
- The MCP subsequently invokes the right MCP tool to retrieve the wealth.
- The LLM returns a response confirming that the address was updated and reporting the current wealth.

Test Case 4 – Ambiguous Input

Objective: Assess the system's ability to interpret ambiguous natural language input and either apply the correct field update or request clarification when the intent cannot be determined with sufficient confidence.

Input (Prompt Example): *"John Doe is now married."*
Expected Behavior:

- The LLM interprets the statement as a potential update to the `marital_status` field.
- If the mapping is unambiguous, the MCP applies the update using the appropriate MCP tool.
- If ambiguity remains, the LLM requests clarification from the user before executing any update.

Test Case 5 – Invalid or Incomplete Input

Objective: Verify that the system detects insufficient input data and prevents case creation until the minimum required information is provided.

Input (Prompt Example): *"Please create a case for a new client."*
Expected Behavior:

- The LLM recognizes that the provided information is insufficient to create a case.
- The system responds with a clarification request (e.g., "Could you provide the client's name and address?").
- No API call to Atfinity is executed until the required data completeness criteria are met.

5.3.6 Manual Test Execution

Manual tests were executed ad hoc by simulating the actions of a relationship manager.

5.4 Quality Measures

Quality assessment for the MCP-based extraction system is based on information retrieval metrics adapted for structured field extraction. The evaluation does not rely on traditional software correctness criteria (e.g., code coverage, cyclomatic complexity) but instead quantifies the alignment between expected and actual field assignments.

5.4.1 Evaluation Metrics

The primary metric is the **F1 score**, which balances precision and recall. These metrics are derived from comparing the set of expected fields against the set of extracted fields.

Precision measures the accuracy of extracted fields, what percentage of the fields that the system extracted are actually correct. A field is considered correct if:

- It was expected in the test case definition
- Its value matches the expected value according to the comparison strategy (exact or partial)

High precision indicates that the system avoids extracting spurious or incorrect fields. For example, if the system extracts 10 fields and 9 of them are correct, precision is 90%.

Recall measures the completeness of extraction, what percentage of the required fields did the system successfully extract and assign correct values.
High recall indicates that the system does not miss important fields. For example, if 10 fields are expected and the system correctly extracts 8 of them, recall is 80%.

F1 Score is the harmonic mean of precision and recall, providing a single metric that balances both concerns. Unlike a simple average, the harmonic mean penalizes extreme imbalances, a system with 100% precision but 50% recall (or vice versa) receives a lower F1 score than a system with 75% in both metrics. A test case is considered to **pass** if its F1 score is ≥ 0.9 (90%), indicating high accuracy in both completeness (recall) and correctness (precision).

Example Calculation: Consider a test case expecting 10 fields. The system extracts 12 fields, of which 9 match the expected values:

- Precision $= \frac{9}{12} = 0.75$ (75%)
- Recall $= \frac{9}{10} = 0.90$ (90%)
- F1 Score $= 2 \cdot \frac{0.75 \cdot 0.90}{0.75 + 0.90} = 0.818$ (81.8%)

This scenario would fail the 90% F1 threshold despite high recall, because the system extracted 3 unexpected or incorrect fields, reducing precision.

5.4.2 Field Comparison Strategies

Field values are compared using one of two strategies, specified in the test case definition:

Exact Matching (default): String values must match exactly after normalization (case-insensitive, whitespace trimmed). This strategy applies to structured fields such as:

- Enumerated values (`currency`, `account_type`)
- Email addresses, postal codes, dates
- Numeric identifiers

Partial Matching (`match_strategy: "contains"`): The expected value must be a substring of the actual value. This strategy is used for descriptive text fields where the LLM may add contextual information or formatting:

- `wealth_description`: e.g., expected *"Swiss technology company"* matches actual *"TechCorp AG, a Swiss technology company"*
- `profession`: e.g., expected *"Technology Consultant"* matches actual *"Senior Technology Consultant and Software Entrepreneur"*

This design prevents penalizing the system for semantically correct but more verbose outputs, which are common in generative models.

5.4.3 Special Handling for Complex Data Types

Unit Fields (Monetary Values): Fields of type `unit` (e.g., `expected_aum`, `total_wealth`) contain both a numeric value and a currency unit:

```
{
  "decimal_number": 8500000.0,
  "unit": "chf"
}
```

Comparison requires both the numeric value (with tolerance < 0.01) and the unit to match exactly. The test framework handles both nested (`decimal_number: {source, parsedValue}`) and flat numeric representations.

Enum Fields: Fields that reference predefined ontology values (e.g., `account_type: "individual_account"`) may be returned as either:

- A simple string (`"individual_account"`)
- A structured object (`{key: "individual_account", label: "Individual Account"}`)

The test framework normalizes both representations to compare only the `key` value.

5.4.4 Performance and Cache Efficiency Metrics

In addition to extraction accuracy, the test framework logs:

- **Execution duration**: total time from prompt submission to result validation
- **Cache hit rate**: percentage of fields satisfied by preemptive extraction without LLM calls (applicable to architectures v6 and v7):

$$\text{Cache Hit Rate} = \frac{\text{Fields from Cache}}{\text{Fields from Cache} + \text{Fields from LLM}} \times 100\%$$

Higher cache hit rates indicate more effective ontology-based field mapping, reducing both latency and API costs.

5.5 Operational Guide

5.5.1 Prerequisites

To set up the development environment for the MCP-based client onboarding system, the following tools and dependencies are required:

- **Python 3.10+** — Core runtime for MCP server and testing framework
- **Node.js 18+** and **npm** — Required for LibreChat frontend dependencies
- **Docker** and **Docker Compose** — Container orchestration for deployment
- **Git** — Version control
- **FastMCP** — Python library for MCP server implementation (`pip install fastmcp`)

The recommended development IDE is **Visual Studio Code** with the following extensions:

- Python (Microsoft)
- Docker (Microsoft)
- YAML (Red Hat)

5.5.2 Project Structure

The prototype repository is organized as follows:

```
Architecture_v5/
|-- src/
|   |-- server.py              # FastMCP server implementation
|   |-- tools/                 # MCP tool definitions
|   `-- utils/                 # Helper functions
|-- quality-measures/
|   |-- test_mcp_server.py     # Automated testing framework
|   |-- test-cases.json        # Test case definitions
|   `-- results.json           # Test execution results
|-- logs/                      # Application logs
|-- processes_mapping.python   # Atfinity process mapping creation script
|-- processes_mapping.yaml     # Atfinity process mapping
|-- ontology_mapping.python    # Atfinity ontology mapping creation script
|-- ontology_mapping.yaml      # Atfinity ontology mapping
|-- docker-compose.yml         # Container orchestration
|-- Dockerfile                 # MCP server container image
|-- .env                       # Environment variables
|-- librechat.yaml             # LibreChat configuration
```

Each architecture variant (v5, v6, v7) follows an identical structure with implementation-specific differences in `server.py` and tool definitions.

5.5.3 Python Dependencies

The MCP server requires the following Python packages, which can be installed via `pip`:

```
fastmcp
requests
pyyaml
openai
python-dotenv
aiohttp
anthropic
```

For the testing framework, additional dependencies are required:

```
asyncio
```

A complete `requirements.txt` file is provided in each architecture directory for reproducible environment setup:

```
pip install -r requirements.txt
```

5.5.4 Local Development Setup

To run the MCP server locally without Docker:

1. Clone the repository and navigate to the desired architecture directory
2. Create and activate a Python virtual environment:

   ```
   python -m venv venv
   source venv/bin/activate  # On Windows: venv\Scripts\activate
   ```

3. Install dependencies: `pip install -r requirements.txt`
4. Configure environment variables in `.env` (see Section 5.6.1)
5. Start the MCP server: `python src/server.py`

The server will be available at `http://localhost:8050/mcp` by default.

5.5.5 Testing Environment Setup

The automated testing framework can be executed locally:

```
cd quality-measures
python test_mcp_server.py --test-cases test-cases.json --output results.json
```

Test execution logs are written to `stdout` and include detailed field-by-field comparison results. The JSON output file contains structured metrics suitable for automated analysis or visualization.

5.6 Deployment & Usage Guide

5.6.1 Configuration

The prototype uses Docker Compose for containerized deployment. A template configuration file (`docker-compose.yml`) is provided in each architecture directory. The deployment consists of three primary services:

mcp-server The FastMCP-based backend service that exposes the MCP protocol endpoint and communicates with the Atfinity API. It is accessible on port 8050 by default.

mongodb Provides data persistence for LibreChat, storing user authentication data, conversation history and session state. Uses MongoDB 6.0+ with authentication enabled.

librechat The web-based frontend that provides a chat interface for interacting with the MCP server via Claude or other LLM clients. Accessible on port 3050 by default. LibreChat is configured to use Anthropic's Claude models (Opus, Sonnet, Haiku) with MCP protocol support enabled.

5.6.2 Environment Variables

The following environment variables must be configured in the `.env` file located in the architecture root directory:

Atfinity API Configuration:

- `ATFINITY_BASE_URL` — Base URL of the Atfinity API (e.g., `https://ost.atfinity.ch`)
- `ATFINITY_API_KEY` — Authentication token for Atfinity API access
- `ATFINITY_API_AUTH_HEADER_NAME` — HTTP header name for authentication (`Authorization`)
- `ATFINITY_API_AUTH_SCHEME` — Authentication scheme (`Api-Key`)
- `ATFINITY_EP_CASES` — API endpoint path for case operations (`/api/1/cases`)

OpenAI Configuration (for LLM field extraction):

- `OPENAI_API_KEY` — API key for OpenAI or compatible LLM service
- `OPENAI_BASE_URL` — Base URL for LLM API (e.g., `https://api.openai.com/v1`)
- `OPENAI_MODEL` — Model identifier (default: `gpt-4o-mini`)

MongoDB Configuration:

- `MONGO_ROOT_USER` — MongoDB administrator username (default: `admin`)
- `MONGO_ROOT_PASSWORD` — MongoDB administrator password

Anthropic Claude Configuration (LibreChat):

- `ANTHROPIC_API_KEY` — API key for Anthropic Claude models

LibreChat Security:

- `JWT_SECRET` — Secret key for JWT token signing
- `JWT_REFRESH_SECRET` — Secret key for refresh token signing

5.6.3 Starting the Application

The recommended deployment method uses Docker Compose. To start all services:

```
docker-compose up --build -d
```

This command performs the following actions:

1. Builds the MCP server Docker image from the provided `Dockerfile`
2. Pulls the MongoDB and LibreChat images from their respective registries
3. Creates a bridged network (`librechat-network`) for inter-service communication
4. Starts all containers with persistent volumes for data retention

The `--build` flag ensures that code changes are incorporated into the container image. The `-d` flag runs containers in detached mode (background).
To verify that all services are running correctly:

```
docker-compose ps
```

Expected output:

```
NAME                  STATUS              PORTS
mcp-server-v5         Up                  0.0.0.0:8050->8050/tcp
librechat-mongodb-v5 Up                   27017/tcp
librechat-v5          Up                  0.0.0.0:3050->3050/tcp
```

5.6.4 Accessing the Application

Once deployed, the application components can be accessed at the following URLs (assuming default configuration):

- **LibreChat Web Interface:** `http://localhost:3050`
- **MCP Server Endpoint:** `http://localhost:8050/mcp`

Initial Setup:

1. Navigate to `http://localhost:3050` in a web browser
2. Register a new user account (if `ALLOW_REGISTRATION` is enabled)
3. Log in with your credentials
4. Select a Claude model from the model dropdown (e.g., Claude Sonnet 4.5)
5. The MCP server tools will be automatically discovered and available in the chat interface

5.6.5 Architecture-Specific Considerations

Each architecture variant (v5, v6, v7) has identical deployment procedures but differs in internal implementation.
To switch between architectures, navigate to the corresponding directory and run `docker-compose up` with the appropriate `.env` configuration.

Chapter 6

Results

6.1 Achievement of Project Goals

The primary goal of this project was to design and implement a conversational AI-based system using MCP, capable of dynamically mapping human-readable free-text input to structured JSON data compatible with Atfinity's CMS. This goal was achieved through the implementation of a modular architecture consisting of a helper Python program, an MCP server, a frontend and a decoupled LLM-component for data extraction and mapping.
Several architectures were explored, of which three were selected and implemented, all sharing the same core architecture but differing in communication flow. The prototypes demonstrate that cases in Atfinity's CMS can be created and updated automatically based solely on natural language input. The system supports process selection, case creation, iterative field filling, data retrieval and case updates without relying on predefined static mappings. These results confirm the technical feasibility of the proposed approach within the defined project scope.

6.2 User Story Coverage

The defined user stories were implemented and evaluated based on observed system behavior during the defined test scenarios. The results below summarize how each user story was supported by the implemented system.

- **User Story 1 – Create Automatic Client Onboarding:** Relationship Managers were able to enter client information conversationally using natural language. The system successfully extracted relevant data and populated onboarding cases automatically, eliminating the need for manual form-based data entry.
- **User Story 2 – Ask for Still Open Fields to Fill Out:** The system correctly identified required but unfilled fields in existing cases. It was able to prompt for missing information for a certain case, enabling Relationship Managers to provide only the missing data.
- **User Story 3 – Update Existing Case:** Existing cases could be updated using natural language instructions without manually specificing fields. The system identified editable fields and applied updates accordingly.
- **User Story 4 – Get Relevant Information and Ask Questions About a Case:** Relationship Managers were able to ask informational questions about cases in natural language. The system retrieved the requested data and returned concise, human-readable responses without requiring manual navigation of the CMS.

6.3 Functional Requirements

The functional requirements were implemented and evaluated based on observed system behavior during the defined test scenarios. The results are summarized below:

- **FR1 – Retrieve Case Information and Other Details:** The system successfully retrieved the data of a case from Atfinity's CMS, including exposed fields, required information and other relevant data. The retrieved data was summarized by the LLM and returned via the frontend, allowing Relationship Managers to ask questions and receive human-readable responses.
- **FR2 – Extract Structured Data from Free-Text Input:** Free-text input provided by Relationship Managers was processed and relevant information was automatically extracted. The system correctly identified relevant information such as names, addresses, nationalities and financial values, including variations in phrasing, numerical formats and expressions of uncertainty (e.g., approximate values).
- **FR3 – Work Hand-in-Hand with Atfinity's CMS:** The system adapted to Atfinity's incremental case architecture by adopting its communication flow to the CMS. Fields were populated only when exposed by the CMS and additional fields were processed as they became available after prior submissions.
- **FR4 – Detect User Intent:** Incoming user messages were correctly classified as either informational queries or data input. Queries resulted in information retrieval from the corresponding case, while data input triggered the appropriate process and MCP tool. The system also distinguished the correct workflow to execute within Atfinity's CMS based on the prompt content.
- **FR5 – Expose Operations as MCP Tools:** All core operations were exposed as MCP tools and accessed through user - LLM interaction. Each tool returned structured JSON responses in accordance with the MCP specification, enabling consistent interaction between the frontend, MCP server and the CMS.

6.4 Non-Functional Requirements

The non-functional requirements were assessed and verified based on implemented quality measures and defined test scenarios:

- **NFR1 – Performance:** The end-to-end processing time for typical onboarding prompts (approximately 150–200 words) was measured across representative test cases. All evaluated architecture variants processed the input within the defined acceptance threshold of 30 seconds, considering the word count and including LLM reasoning, field mapping and required API calls.
- **NFR2 – Precision:** The precision of extracted field values was evaluated against human-validated reference data. Across the tested scenarios, the ratio of correctly extracted fields to all extracted fields met or exceeded the defined acceptance threshold of 80%.
- **NFR3 – Recall:** Recall was measured by comparing extracted fields against all fields that should have been extracted according to the reference data. The evaluated architectures achieved recall values at or above the target threshold of 90% in the majority of test scenarios.
- **NFR4 – F1-Score:** The F1-score, calculated as the harmonic mean of precision and recall, was used as the primary aggregate quality metric. The measured F1-scores met or exceeded the required acceptance value of 85% for the evaluated onboarding scenarios.
- **NFR5 – Usability:** Relationship Managers were able to interact with the system using human-readable free-text prompts without knowledge of Atfinity-specific technical terms or field identifiers. During testing, client information entered in natural language was automatically processed and applied by the system without requiring additional technical input.

6.5 Implementation Status of Requirements

The following tables summarize the implementation status of user stories and requirements based on the results presented above.

User Story	**Implemented**
User Story 1: Create automatic client onboarding	Yes
User Story 2: Ask for still open fields to fill out	Yes
User Story 3: Get relevant information and ask questions about case	Yes

Table 6.1: User Stories and Implementation

Functional Requirements	**Implemented**
FR1: Retrieve Current Case State	Yes
FR2: Extract Structured Data from Free-Text Input	Yes
FR3: Maintain a Working Draft	Yes
FR4: Detect User Intent	Yes
FR5: Expose Operations as MCP Tools	Yes

Table 6.2: Functional Requirements and Implementation

Non-Functional Requirements	**Verified**
NFR-1: Performance	Yes
NFR-2: Precision	Yes
NFR-3: Recall	Yes
NFR-4: F1-Score	Yes
NFR-5: Usability	Yes

Table 6.3: Non-Functional Requirements and Verification

6.6 Research Question Results

During the project, five research questions guided the development and evaluation of the proposed system. This section summarizes the results for each research question based on the conducted tests and measured quality metrics.

6.6.1 RQ1 – Information Extraction Accuracy

Research Question: How effectively can a large language model extract and map client information from free-text prompts into Atfinity's structured fields, even when the prompt does not explicitly indicate the type of information provided?
Result: The evaluated architectures demonstrated that large language models can effectively extract and map client information from unstructured free-text input to Atfinity's structured fields. Extraction accuracy was quantified using precision, recall and F1-score metrics against human-validated reference data. Across the tested scenarios, precision exceeded the defined acceptance threshold of 80%, recall met or exceeded 90% for most cases and F1-scores reached values of 85% or higher. These results indicate that semantic understanding and field-mapping capabilities were sufficient to correctly interpret implicit information during both case creation and case updates.

6.6.2 RQ2 – Intent Detection

Research Question: How accurately can the system distinguish between user queries and data input requests and how does this classification affect overall correctness?
Result: The system successfully classified incoming prompts as either informational queries or data input instructions. During testing, correct intent classification enabled the appropriate invocation of the right MCP tool and triggered the right process within Atfinity's CMS. No incorrect case modifications were observed as a result of misclassified prompts in the evaluated test cases. The results show that reliable intent detection and process evalution is essential for correct MCP tool invocation and directly contributes to overall system correctness.

6.6.3 RQ3 – Client Suitability

Research Question: Is a standard MCP client or open source UI sufficient for the problem domain, or does the domain require a dedicated client interface?
Result: The use of a standard MCP client and an open source user interface was sufficient to demonstrate and validate the proposed system within the scope of the project. All defined user stories, including case creation, updates and informational queries were successfully executed without requiring a domain-specific UI or client.

6.6.4 RQ4 – Performance

Research Question: What are the end-to-end latencies for automated onboardings?
Result: End-to-end onboarding durations were measured from prompt submission to completed case creation. Across all evaluated architectures, processing times remained mostly within the defined performance threshold of 30 seconds for typical onboarding prompts of 150–200 words. Execution times varied by architecture. These measurements confirm that automated onboarding can be performed within acceptable time limits for interactive use by Relationship Managers.

6.6.5 RQ5 – Architecture and Model Selection

Research Question: What architecture is best suited for this task, how do different large language models perform and are locally deployed models viable alternatives to cloud-based solutions?
Result: The comparative evaluation of architecture variants showed measurable trade-offs between extraction quality and performance. Iterative architectures achieved higher recall and more complete field coverage, while single-pass architectures reduced processing time at the cost of lower recall for complex cases. All experiments were conducted using a cloud-based LLM (GPT-4o-mini & Claude). The results demonstrate that cloud-based models are viable and effective for the problem domain. A comparative evaluation with alternative models or locally deployed solutions was not conducted within the scope of this project and therefore cannot be assessed based on the available results.

6.7 Comparative Architecture Results

Four predefined test cases were executed against the final three architecture variants (v5, v6 and v7) to measure accuracy, completeness and performance. Test cases were grouped as follows:

- **Simple Cases**: Test Cases 1 and 2 (individual accounts in EUR and CHF)
- **Complex Case**: Test Case 4 (HNWI with political affiliation)
- **Corporate Case**: Test Case 3 (entity account)

6.7.1 Architecture v5: Iterative Mapping

Architecture v5 implements an iterative AI-driven extraction strategy, checking for newly exposed fields after each Atfinity response. [Architecture v5]

Metric	Simple Cases	Complex Case	Corporate Case
Precision	>90%	>85%	>90%
Recall	>90%	>90%	>67%
F1 Score	∼95%	>90%	>80%
Avg. Duration	∼25s	∼40s	∼14s

Table 6.4: Performance metrics for Architecture v5

The results show consistently high precision and recall for individual cases, with reduced recall for corporate cases due to entity-specific field gaps.

6.7.2 Architecture v6: Preemptive Caching with Fallback

Architecture v6 combines preemptive mapping with a fallback mechanism to the iterative AI-driven mapping when precomputed mappings are insufficient. [Architecture v6]

Metric	Simple Cases	Complex Case	Corporate Case
Precision	>70%	>85%	>90%
Recall	>90%	>90%	>90%
F1 Score	>82%	>92%	>95%
Avg. Duration	∼36s	∼44s	∼32s

Table 6.5: Performance metrics for Architecture v6

This architecture achieved high recall across all categories, including corporate cases - at the cost of longer execution times and reduced precision for simple cases, due to "hallucination". For example, assuming expected AUM equals total wealth when only one value is provided.

6.7.3 Architecture v7: Single-Pass Mapping

Architecture v7 uses a preemptive generated mapping and fills out all fields in a single-pass without iterative LLM calls. [Architecture v7]

Metric	Simple Cases	Complex Case	Corporate Case
Precision	>95%	>90%	>90%
Recall	>75%	>85%	>45%
F1 Score	>85%	>90%	>60%
Avg. Duration	∼17s	∼24s	∼11s

Table 6.6: Performance metrics for Architecture v7

Architecture v7 demonstrates strong precision across all case types, with optimal extraction quality for complex cases. The corporate case achieves the fastest execution time but at the cost of significantly reduced recall.

6.7.4 Observed Trade-offs

The comparative results highlight measurable trade-offs between execution time and field completeness across the evaluated architectures. Iterative approaches achieved higher recall, while single-pass extraction reduced processing time but missed a larger proportion of fields in complex scenarios.

6.8 Limitations

The conducted evaluation is subject to the following limitations:

- Only four test scenarios were defined, limiting coverage of edge cases and uncommon input patterns.
- The system was not evaluated against adversarial or intentionally malformed input.
- All tests were performed using a single LLM model. In other words: Multiple LLM models for the decoupled component were not tested.
- Test data was synthetic and may not reflect the full variability of real-world RM interactions.

6.9 Delivered Artifacts

The following artifacts were produced as part of the project:

- An MCP server with four core management classes and four MCP tools.
- A helper Python program for process and ontology mapping.
- YAML-based context resources for LLM prompting.
- Three working prototypes demonstrating dynamic case creation, updates and data retrieval in Atfinity's CMS.
- Technical documentation describing architecture and workflows.
- Blogpost

Part V

Project Documentation

Chapter 7

Project- and Time management

7.1 Project planning

The planning describes how the project is managed and planned.

7.1.1 Methodology

The project was initially conducted using the OST-developed Scrum+, a hybrid methodology combining RUP and Scrum. However, during the early stages of the project, this approach proved to be infeasible for the given context. As a result, the methodology was changed to a pure Scrum framework. Scrum supported development in exploratory and uncertain conditions while still maintaining a clear focus on long-term objectives.

7.1.2 Project Members

Role	**Name**	**Comment**
Client / Partner	Atfinity (Thorben Croisé)	Industrial Partner of OST and CTO of Atfinity
Supervisor	Prof. Dr. Mitra Purandare	
Developer, SA Author	Arnel Veladzic	
Developer, SA Author	Fabio Gomes Silva	

Table 7.1: Roles

7.1.3 Responsibilities

Each team member is not solely responsible for executing all tasks within their assigned area. Instead, they oversee their respective components, ensuring alignment and progress. For example, Arnel Veladzic is responsible for scheduling and managing meetings, ensuring the project remains on track. However, Fabio Gomes Silva also contributes to the planning process and actively commit to the scheduled milestones.

Arnel Veladzic:

- Project planning
- Prototype Architecture and Implementation
- Quality Measures: Implementation and Evaluation (Assistance)
- Documentation

Fabio Gomes Silva:

- Test Concept
- Quality Measures: Implementation and Evaluation
- Prototype Architecture and Implementation (Assistance)
- Documentation

7.1.4 Collaboration and Meetings

Since the project is working with Agile/Scrum most of the meetings will be related to it.

- Sprint Review/Retro: Every second Sunday 10:00 - 11:00
- Sprint Planning: Every second Sunday 11:00 - 12:00
- Weekly Sync, with Supervisor and Atfinity: Every Thursday 15:00 - 15:30
- Additional Meetings: Schedule when needed

7.1.5 Planning Tools

For effective tracking and management, the following tools are utilized:

- Jira: Task tracking and sprint backlog organization
- Confluence: Collaborative knowledge sharing and notes
- Miro: Visual project planning
- Clockify Plugin for Jira: Time tracking

7.1.6 Long-Term Planning

Following the transition to a purely Scrum-based agile approach, a detailed long-term plan, as typically required by methodologies such as Scrum+, was no longer maintained. Instead, the planning artifacts focus on outlining the overall project trajectory and the final project plan. The idea was to iteratively design and implement one architectural variant per sprint. This approach enabled structured exploratory work while allowing continuous evaluation and refinement of architectural decisions based on emerging insights and project requirements.

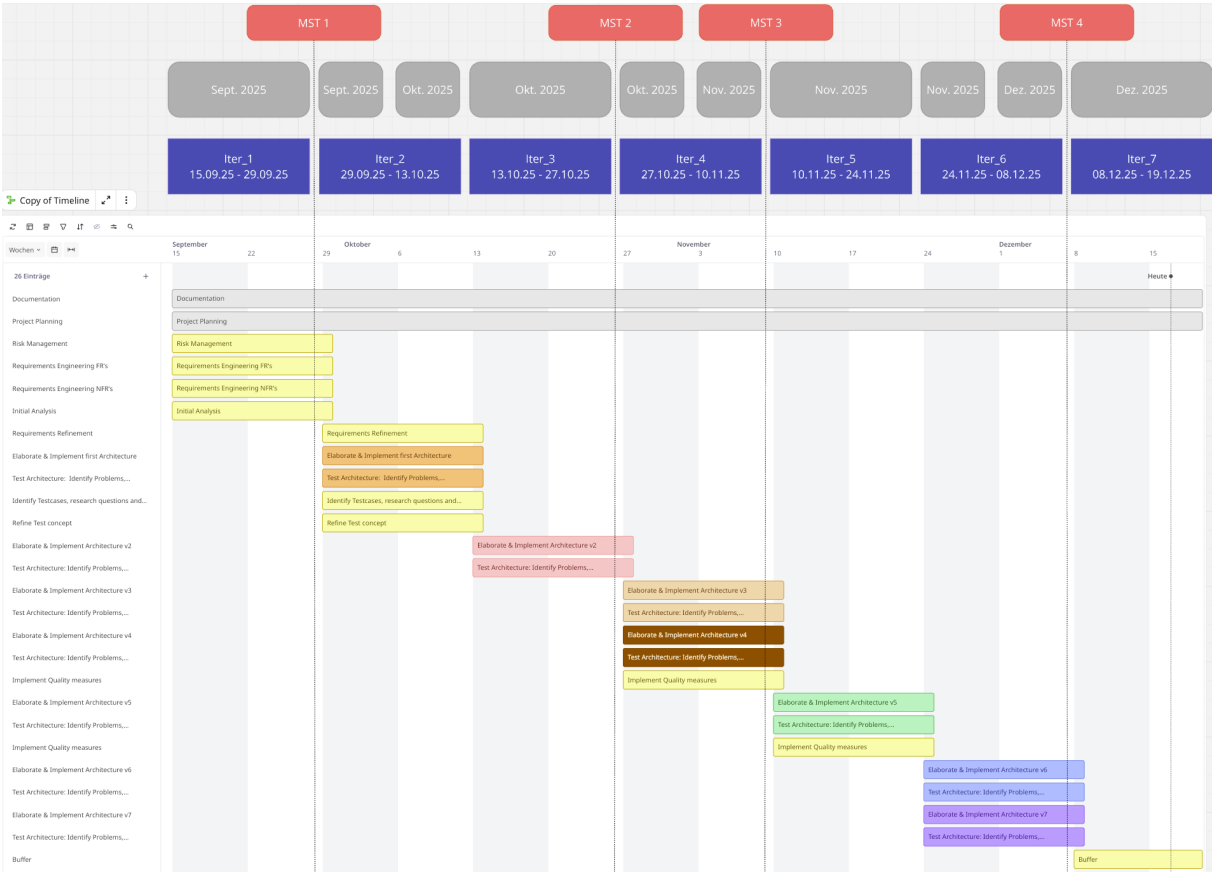


Figure 7.1: Final Project Plan

7.1.7 Milestones

The project is structured into clearly defined milestones to illustrate the progression of major activities and to indicate when key requirements and deliverables are achieved.

Milestone 1: Initial Analysis and Project Setup

Functional and non-functional requirements are identified, written down and reviewed. An initial analysis of the problem area is carried out and a basic project plan is created to guide the following sprints.

Artifacts:

- Documentation
- Functional Requirements
- Non-Functional Requirements
- Initial Project Plan
- Identify risks

Milestone 2: Playground, Comprehensive Analysis and plan for solving dynamic mapping problem

At least two versions of the system architecture is designed, implemented and tested to validate their feasibility. A comprehensive analysis of Atfinity's CMS is conducted. How does the platform behave? Which data is relevant to us? What do we need pay attention to in order to be able to solve the dynamic mapping problem? An initial architectural plan on how to solve the dynamic mapping problem is designed and reviewed.

Artifacts:

- Documentation
- Architecture v1
- Architecture v2
- Knowledge about CMS
- Design of Architecture v3 in order to solve the problem domain

Milestone 3: MVP

A working MVP is developed that addresses the problem of dynamically mapping human-readable free-text prompts to structured JSON data based on the previously designed architecture. This MVP provides the core architecture and essential functionality that serve as a foundation for subsequent architectural evolution and further extensions.

Artifacts:

- Documentation
- MVP

Milestone 4: Final Prototype(s), answered Research Questions and reviewed Quality Measures

The most suitable architecture is selected and one working prototype is developed. All predefined research questions are addressed and answered. Automated quality measures are implemented and the prototype is reviewed based on the defined metrics and their results.

Artifacts:

- Documentation
- Working Prototype(s)
- Answers to Research Questions
- Qualiy Measures

7.1.8 Short-Term Planning

Since the project is conducted using a fully agile methodology, work is organized into iterations with a duration of two weeks. A detailed short-term plan is not defined at the beginning of the project. Instead, planning is carried out at the start of each iteration in the form of sprint planning.

During sprint planning, upcoming work items are reviewed and prioritized, larger requirements are broken down into manageable tasks and new tasks are added when necessary. Effort estimates are discussed collaboratively and tasks are assigned to team members based on current priorities and capacities.

This iterative planning approach allows the project to remain flexible, continuously adapt to new insights and ensure steady progress throughout the development process.

7.2 Time Management

The working hours of the team are systematically tracked throughout the project using clockify. Time spent is recorded at the project level rather than being assigned to individual tasks or issues. Each team member is responsible for logging their working time accurately and consistently to ensure transparency and reliable effort tracking across the project.

7.2.1 Time Management Statistics

The following graphics illustrate the time tracking statistics. The time spent is compared to the time that should be achieved.
The increased effort observed in Sprints 3 and 4 is attributable to the development of the MVP, during which the primary focus was on establishing a core architecture capable of addressing the dynamic mapping problem. The elevated effort in Sprint 7 is primarily related to the finalization of the remaining documentation.

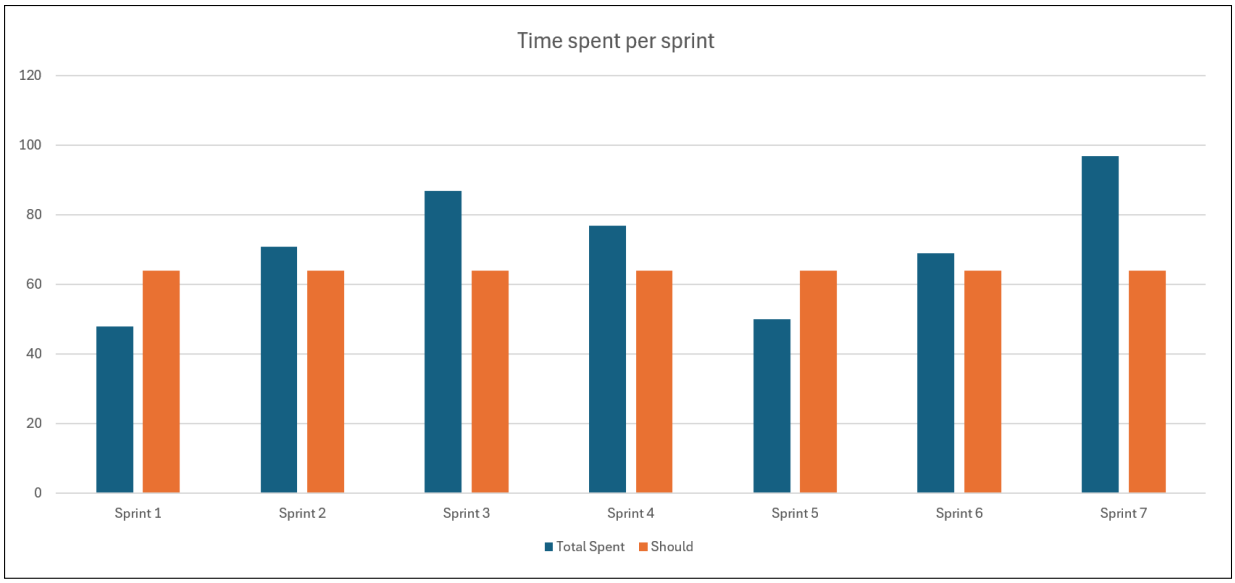


Figure 7.2: Time spent per sprint of whole team

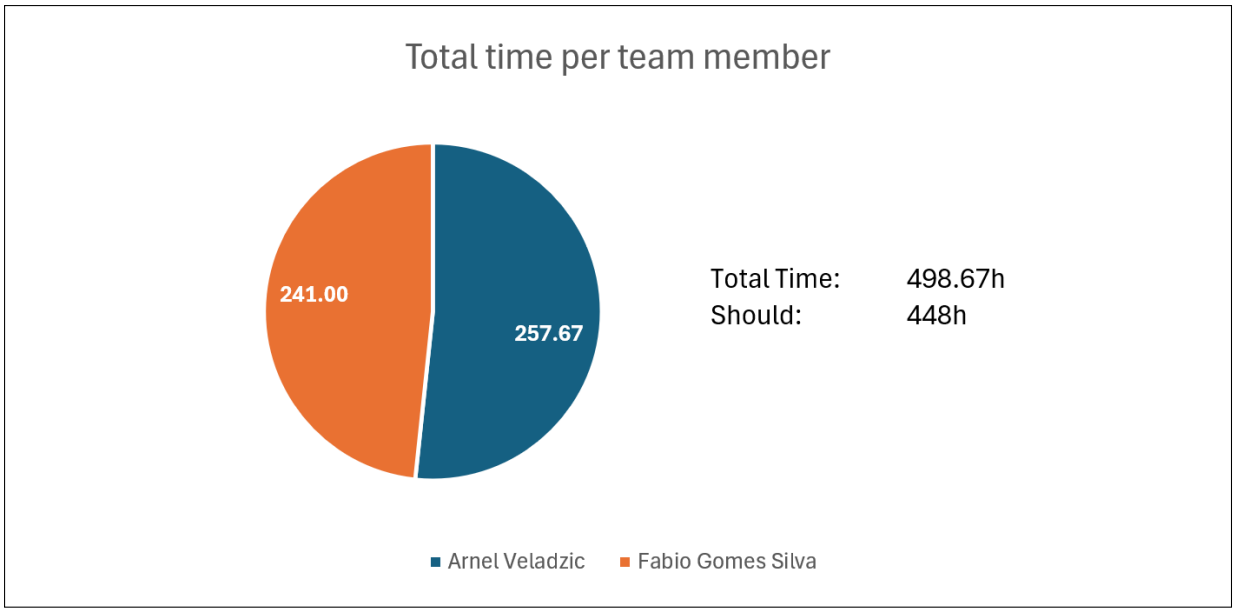


Figure 7.3: Time spent per team member for whole project

7.3 Risk Management

To manage project risks, a structured risk analysis is conducted to identify the key risks associated with the project. These risks are reviewed and updated on a regular basis to ensure adequate preparedness for potential issues.

7.3.1 Risks and Mitigations

Risk	**Description**	**Mitigation**	**Owner**
Not enough buffer planned	Insufficient time reserves for delays	Add time buffers to milestones	Arnel Veladzic and (Project Team)
Scope Creep	Uncontrolled expansion of requirements	Strict backlog prioritization	Arnel Veladzic and (Project Team)
Underestimation of Man-Hours	Effort estimates are too optimistic	Regular re-estimation per sprint	Arnel Veladzic and (Project Team)
Unclear/changing requirements	Requirements are incomplete or unstable	Continuous refinement and clarification	Project Team
Technology dependency	Reliance on external frameworks or tools	Evaluate alternatives early	Project Team
Knowledge gap	Missing expertise in specific areas	Targeted learning and documentation	Project Team
Bugs & Failures	Software defects affect functionality	Continuous testing and reviews	Project Team
Misconfig of Architecture	Incorrect system configuration	Architecture reviews and validation	Project Team
Misunderstood Atfinity's CMS	Incorrect usage of CMS features	Study documentation and prototypes	Project Team

Table 7.2: Risks and their mitigations (color matching to risk matrix)

7.3.2 Risk Matrix in Sprint 1

To support effective risk management, a structured risk analysis is conducted to identify the key risks associated with the project. These risks are reviewed and updated on a regular basis to ensure adequate preparedness for potential issues throughout the project lifecycle. Risk Cateogrization/Legend:

- Blue: Project Planning Risks
- Green: Technical Risks

Risks related to individual team members were deliberately excluded from the analysis, as the project is carried out by a team of only two developers. In this context, a detailed assessment of personnel-related risks was considered neither meaningful nor proportionate.

Figure 7.4: Sprint 1; Risk Matrix

Not Enough Buffer Planned

This risk is placed at moderate impact and rare probability, as initial milestones included basic time buffers. However, at the beginning of the project, uncertainties were still present, making insufficient buffering a potential but not yet likely issue.

Scope Creep

Scope creep is positioned at major impact and moderate probability. During Sprint 1, requirements were still being refined and changes were expected. Uncontrolled extensions of scope could significantly affect timelines and deliverables.

Underestimation of Man-Hours

This risk is located at major impact and likely probability. Due to limited prior experience with the problem domain and technology stack, effort estimation in early sprints was inherently uncertain.

Unclear or Changing Requirements

This risk is placed at major impact and moderate probability. In Sprint 1, requirements were not yet fully stabilized, which could lead to rework and architectural changes.

Technology Dependency

Technology dependency is positioned at major impact and unlikely probability. While the project relies on specific technologies, alternative solutions were known, reducing the likelihood of blocking issues early on.

Knowledge Gap

This risk is placed at major impact and unlikely probability. Although some technologies were new to the team, foundational knowledge was available and learning was planned as part of the early sprints.

Bugs and Failures

Bugs and failures are located at major impact and moderate probability. Early development phases typically involve instability, especially during architectural prototyping.

Misconfiguration of Architecture

This risk is positioned at extreme impact and likely probability. In Sprint 1, the architecture was still evolving, increasing the risk of incorrect design or configuration decisions.

Misunderstood Atfinity's CMS

This risk is placed at extreme impact and moderate probability. At the start of the project, the CMS was not yet fully understood, which could lead to incorrect assumptions affecting development.

7.3.3 Risk Matrix in Sprint 2

Risk Cateogrization/Legend:

- Blue: Project Planning Risks
- Green: Technical Risks

Figure 7.5: Sprint 2; Risk Matrix

Not Enough Buffer Planned

After Sprint Two, this risk remains in the low-probability and minor-impact area. Early sprint reviews and adaptive sprint planning helped validate the timeline assumptions, reducing the likelihood of schedule pressure. Continuous backlog refinement serves as an ongoing mitigation measure.

Scope Creep

Scope creep remains unlikely but retains a major potential impact. The introduction of clearer sprint goals and stricter backlog prioritization reduced the probability of uncontrolled scope growth. However, its placement reflects that uncontrolled changes could still significantly affect delivery.

Underestimation of Man-Hours

This risk moved slightly toward lower probability due to improved task breakdown and experience gained during Sprint One. Regular sprint retrospectives and effort reassessments act as mitigation strategies, though the impact remains moderate due to a small number of team memebers.

Unclear or Changing Requirements

The probability of this risk decreased as requirements were clarified through continuous stakeholder feedback and sprint reviews. Nevertheless, the impact remains major, since late requirement changes could still necessitate architectural adjustments.

Technology Dependency

Technology dependency remains unlikely but with a major impact. Early prototyping and architectural validation in Sprint Two reduced uncertainty, mitigating the probability. The impact remains significant due to reliance on external frameworks and services.

Knowledge Gap

This risk shifted toward lower probability as team members gained familiarity with the selected technologies and domain concepts. Knowledge sharing and joint development sessions served as effective mitigation measures, while the impact remains major in case critical expertise is missing.

Bugs and Failures

The risk of bugs and failures is assessed as moderately likely with major impact. Initial testing practices and early integration reduced uncertainty. Incremental development, testing and continuous integration act as mitigation measures.

Misconfiguration of Architecture

The probability of architectural misconfiguration decreased due to early architectural reviews and validation of core components. However, the impact remains extreme, as fundamental configuration errors could require extensive rework.

Misunderstood Atfinity Functionalities

This risk remains rare but with extreme impact. Initial clarification efforts and documentation review reduced the likelihood, but misunderstandings could still severely affect system integration. Continued communication and validation against requirements are used as mitigation strategies.

7.3.4 Risk Matrix in Sprint 3

Risk Cateogrization/Legend:

- Blue: Project Planning Risks
- Green: Technical Risks

Figure 7.6: Sprint 3; Risk Matrix

Not Enough Buffer Planned

This risk remains at low probability and minor impact. Buffer awareness improved through continuous sprint planning, but limited flexibility still exists due to fixed academic deadlines.

Scope Creep

Scope creep is assessed as unlikely but with major impact. Clear sprint goals and backlog refinement reduced its probability, while its potential effect on timelines and quality remains significant.

Underestimation of Man-Hours

The risk is placed at moderate probability and major impact. Although experience from earlier sprints improved estimation accuracy, exploratory development still introduces uncertainty in effort estimation.

Unclear or Changing Requirements

This risk is positioned at unlikely probability and major impact. Regular stakeholder alignment and iterative clarification reduced likelihood, but changes would still significantly affect architecture and implementation.

Technology Dependency

Technology dependency remains unlikely with major impact. Early architectural decisions and prototyping mitigated uncertainty, but reliance on external frameworks still poses a structural risk.

Knowledge Gap

The knowledge gap risk is classified as unlikely with major impact. Continuous learning and shared development tasks reduced probability, while insufficient expertise could still slow progress if encountered.

Bugs and Failures

This risk moved to likely probability with extreme impact. Increased implementation activity raised the likelihood of defects and failures at this stage could directly affect system stability and milestone achievement.

Misconfiguration of Architecture

The risk is assessed as moderate probability with extreme impact. As the architecture becomes more concrete, configuration errors are more likely and their consequences would be difficult to correct later.

Misunderstood Atfinity's Functionalities

This risk remains rare but with extreme impact. Incorrect assumptions about platform behavior would still have severe consequences if not detected early.

7.3.5 Risk Matrix in Sprint 4

Risk Cateogrization/Legend:

- Blue: Project Planning Risks
- Green: Technical Risks

Figure 7.7: Sprint 4; Risk Matrix

Not Enough Buffer Planned

This risk remains in the low-probability and minor-impact area. Buffer planning has been reviewed and continuously monitored, reducing the likelihood of unexpected schedule pressure. The remaining impact is limited due to early detection and mitigation.

Scope Creep

Scope creep has moved to a higher probability and major impact area. As implementation progressed, new feature ideas and refinements emerged. Regular backlog refinement and strict sprint goal enforcement are used to control this risk.

Underestimation of Man-Hours

This risk is positioned at moderate probability and major impact. Although estimation accuracy has improved based on previous sprints, complex implementation tasks still carry uncertainty. Continuous re-estimation during sprint planning mitigates the impact.

Unclear or Changing Requirements

The risk remains relevant with moderate probability and major impact. While requirements are more stable than in earlier sprints, refinements based on testing still occur. Frequent clarification and short feedback loops help reduce misunderstandings.

Technology Dependency

Technology dependency is placed at low probability but major impact. Core technologies have been validated, lowering likelihood; however, failures or limitations in external components would still have significant consequences. Early prototyping and fallback solutions mitigate this risk.

Knowledge Gap

This risk remains low in probability with major impact. Team knowledge has increased through hands-on development, but gaps may still arise in advanced or less familiar areas. Continuous learning and documentation reduce the likelihood further.

Bugs and Failures

Bugs and failures are positioned at high probability and extreme impact. Increased system complexity during implementation raises the likelihood of defects. Automated testing and incremental validation are applied to limit their impact.

Misconfiguration of Architecture

This risk is assessed as low probability but extreme impact. Architectural decisions are largely stabilized, reducing the chance of misconfiguration; however, any fundamental architectural error would significantly affect the system. Reviews and refactoring mitigate this risk.

Misunderstood Atfinity's Functionalities

The risk remains low in probability with extreme impact.

7.3.6 Risk Matrix in Sprint 5

Risk Cateogrization/Legend:

- Blue: Project Planning Risks
- Green: Technical Risks

Figure 7.8: Sprint 5; Risk Matrix

Not Enough Buffer Planned

This risk remains in the low-probability and minor-impact area.

Scope Creep

Scope creep is positioned with low probability but major impact. The core functionality and architecture was developed reducing the risk of a scope creep.

Underestimation of Man-Hours

This risk is placed at moderate probability and major impact. Although estimation accuracy improved through experience from previous sprints, complex implementation tasks still carried the risk of requiring more effort than initially planned.

Unclear or Changing Requirements

The risk is assessed as low probability with major impact. Regular communication and iterative clarification with stakeholders significantly reduced uncertainty, though late requirement changes could still have substantial consequences.

Technology Dependency

Technology dependency remains at low probability and major impact. Core technology decisions were stabilized, but reliance on specific frameworks or tools could still introduce risks if unexpected limitations arise.

Knowledge Gap

This risk is placed in the low-probability, major-impact area. Knowledge gaps were mitigated through shared development efforts and documentation, yet complex architectural decisions still required careful handling.

Bugs and Failures

Bugs and failures are positioned at moderate probability and extreme impact. Increased system complexity and integration activities raised the likelihood of defects, making thorough testing and validation essential.

Misconfiguration of Architecture

This risk remains at low probability but extreme impact. Architectural decisions were largely finalized; however, configuration errors at this stage could severely affect system stability and scalability.

Misunderstood Atfinity's Functionalities

The risk is assessed as low probability with extreme impact. Improved familiarity with the platform reduced misunderstandings, though incorrect assumptions about core functionalities could still lead to significant rework.

7.3.7 Risk Matrix in Sprint 6

Risk Cateogrization/Legend:

- Blue: Project Planning Risks
- Green: Technical Risks

Figure 7.9: Sprint 6; Risk Matrix

Not Enough Buffer Planned

This risk remains in the low-probability and low-impact area, as buffer considerations were continuously reviewed and adjusted throughout the project. Regular sprint reviews and adaptive planning reduced the likelihood of schedule overruns caused by missing buffer time.

Scope Creep

Scope creep is positioned with low probability but major impact. All requirements were mostly satisfied and implemented eliminating this risk almost entirely. However a change in the requirements could have direct impact on this risk.

Underestimation of Man-Hours

The risk is placed in the low-probability but major-impact area. Improved estimation accuracy was achieved through experience from previous sprints and better task breakdowns. However, underestimation could still have considerable impact on delivery timelines if it occurred.

Unclear or Changing Requirements

This risk is assessed as having low probability and major impact. Regular communication, clarification sessions and early validation of requirements reduced uncertainty, although requirement changes would still significantly affect development effort.

Technology Dependency

Technology dependency remains in the low-probability, major-impact quadrant. Key technologies were validated early through prototyping, reducing uncertainty, but dependencies on external tools or frameworks could still have notable consequences if issues arose.

Knowledge Gap

The knowledge gap risk is placed in the low-probability, major-impact area. Continuous learning, documentation and shared implementation efforts reduced the likelihood of skill-related issues, while unresolved gaps could still affect solution quality.

Bugs and Failures

This risk is positioned with low probability but extreme impact. Increased test coverage, stabilization of the core architecture and iterative bug fixing lowered the likelihood of critical failures, although severe defects would still have a major effect if they occurred late in the project.

Misconfiguration of Architecture

The risk is located in the low-probability and extreme-impact area. By Sprint 6, the architecture was validated and stable, reducing the chance of misconfiguration, while architectural errors would still have far-reaching consequences.

Misunderstood Affinity's Functionalities

This risk is assessed as low probability with extreme impact. Through hands-on usage, documentation review and iterative integration, understanding of the system improved significantly. Nevertheless, misunderstandings could still critically affect the overall outcome.

Chapter 8

Personal Reports

8.1 Arnel Veladzic

8.1.1 What Went Well

The exploratory and iterative approach worked very well for this project. At the beginning, many aspects were unclear to us, such as MCP behavior, LLM reasoning quality and Atfinity's conditional architecture. By building and testing several architecture versions, we were able to learn step by step and improve the solution over time. Early versions helped us understand the basics, while later ones focused more on quality, performance and overall improvements. Project coordination also worked well across multiple sprints. Regular planning, retrospectives and close communication helped us stay aligned and adjust both the technical direction and the way of working. Important technical insights, such as differences between LLMs and the need for incremental field processing instead of static mappings, strongly influenced the final architecture and were confirmed through testing. A clear architecture-first approach with separated components made it easier for us to iterate quickly and keep complexity manageable.

8.1.2 Areas for Improvement

One area for improvement is documentation and supervision. While working independently was efficient for me, earlier and more focused feedback on architecture decisions and documentation could have improved the final prototype. In future projects, I would benefit from more regular review sessions beyond simple status updates.

8.1.3 Personal Highlights

The overall highlight of this project for me was the opportunity to explore a very new and largely unexplored area. Instead of developing something purely theoretical or isolated, the work focused on solving a real problem and creating practical value for a real company. Seeing how experimental ideas around LLMs, MCP and automation could be applied to real onboarding workflows made the project especially motivating.

Furthermore, using measurements to guide architectural decisions was also very rewarding. Quality metrics and performance data directly influenced how the final prototypes evolved, making decisions more objective and based on real results. Key findings, such as clear differences in LLM behavior and the need for dynamic field mapping, strongly shaped the final solution. Dealing with approaches that did not work and changing direction when needed was challenging but valuable. Overall, contributing to a system that is both technically interesting and practically useful strengthened my skills in system design, technical leadership and applying new technologies in a real-world context.

8.2 Fabio Gomes Silva

8.2.1 What Went Well

We developed a automated testing framework that became the foundation for architectural decision-making for the final prototypes. Rather than relying on subjective assessments, we designed a rigorous quality measurement system with four representative scenarios, precision/recall/F1-score metrics and detailed field-by-field validation. This transformed how Arnel and I evaluated architectural trade-offs, enabling data-driven decisions about whether to optimize for performance or completeness.
Working closely with Arnel created a productive feedback loop. While he focused on architecture and implementation, I provided feedback on each variant's performance. The measurements and quality reports directly informed architectural refinements. This division of responsibilities, proved highly effective for exploratory development. My testing revealed crucial insights that shaped the project. The performance measurements across architectures v5-v7 provided concrete evidence of trade-offs between recall, precision and latency. This measurement-driven approach transformed architecture design from a theoretical exercise into an empirically-grounded process.

8.2.2 Areas for Improvement

We could have benefited from more feedback regarding testing methodology and quality frameworks. In future projects, I will seek guidance on measurement strategies earlier rather than finalizing them independently. Additionally, conducting comparative evaluations with alternative testing approaches or additional model variants could have provided deeper insights into the robustness of our measurement framework.

8.2.3 Personal Highlights

All non-functional requirements were met or exceeded through rigorous testing. The most satisfying moment was discovering that Claude significantly outperformed GPT-4o-mini at understanding and invoking MCP tools, a finding that revealed fundamental differences in model behavior affecting entire system design. Knowing that my measurements directly influenced the selection of v7 as the recommended production variant gave tangible meaning to the testing work.
I gained valuable hands-on experience with test framework design for non-deterministic systems, precision/recall/F1 score evaluation, field-comparison strategies for complex data types and measurement methodologies for AI-driven systems. Beyond the metrics, this project demonstrated that rigorous quality measurement is not merely validation but a design driver in uncertain domains. While Arnel's iterative architectural refinements provided the innovation, systematic measurement provided the evidence for sound decisions.

Bibliography

[1] Anthropic. Claude. Used as a support tool for drafting, language refinement and documentation assistance. No content was generated without human review. URL: https://claude.com/product/overview.

[2] Atfinity. Atfinity api documentation. Technical API reference, [last access: 2025-11-03]. URL: https://api-docs.atfinity.io/#intro.

[3] Atfinity AG. Atfinity – intelligent process automation for financial institutions. Official company website, [last access: 2025-12-16]. URL: https://www.atfinity.swiss.

[4] Simon Brown. The c4 model for visualizing software architecture. Software architecture visualization method, [last access: 2025-12-03]. URL: https://c4model.com/.

[5] Docker. Contributing.md for docker/mcp-registry. Project contribution guidelines, [last access: 2025-11-12]. URL: https://github.com/docker/mcp-registry/blob/main/CONTRIBUTING.md.

[6] LibreChat. Librechat documentation. Official software documentation, [last access: 2025-11-10]. URL: https://www.librechat.ai/docs.

[7] Model Context Protocol. Model context protocol: Getting started. Technical specification and documentation, [last access: 2025-11-27]. URL: https://modelcontextprotocol.io/docs/getting-started/intro.

[8] OpenAI. Chatgpt. Used as a support tool for drafting, language refinement and documentation assistance. No content was generated without human review. URL: https://chatgpt.com/.

[9] OpenWebUI. Quick start guide — openwebui documentation. Official documentation, [last access: 2025-09-30]. URL: https://docs.openwebui.com/getting-started/quick-start/.

List of Figures

List of Tables

Appendices

Appendix A

Examples of Working Client Onboardings

A.1 Example Scenarios

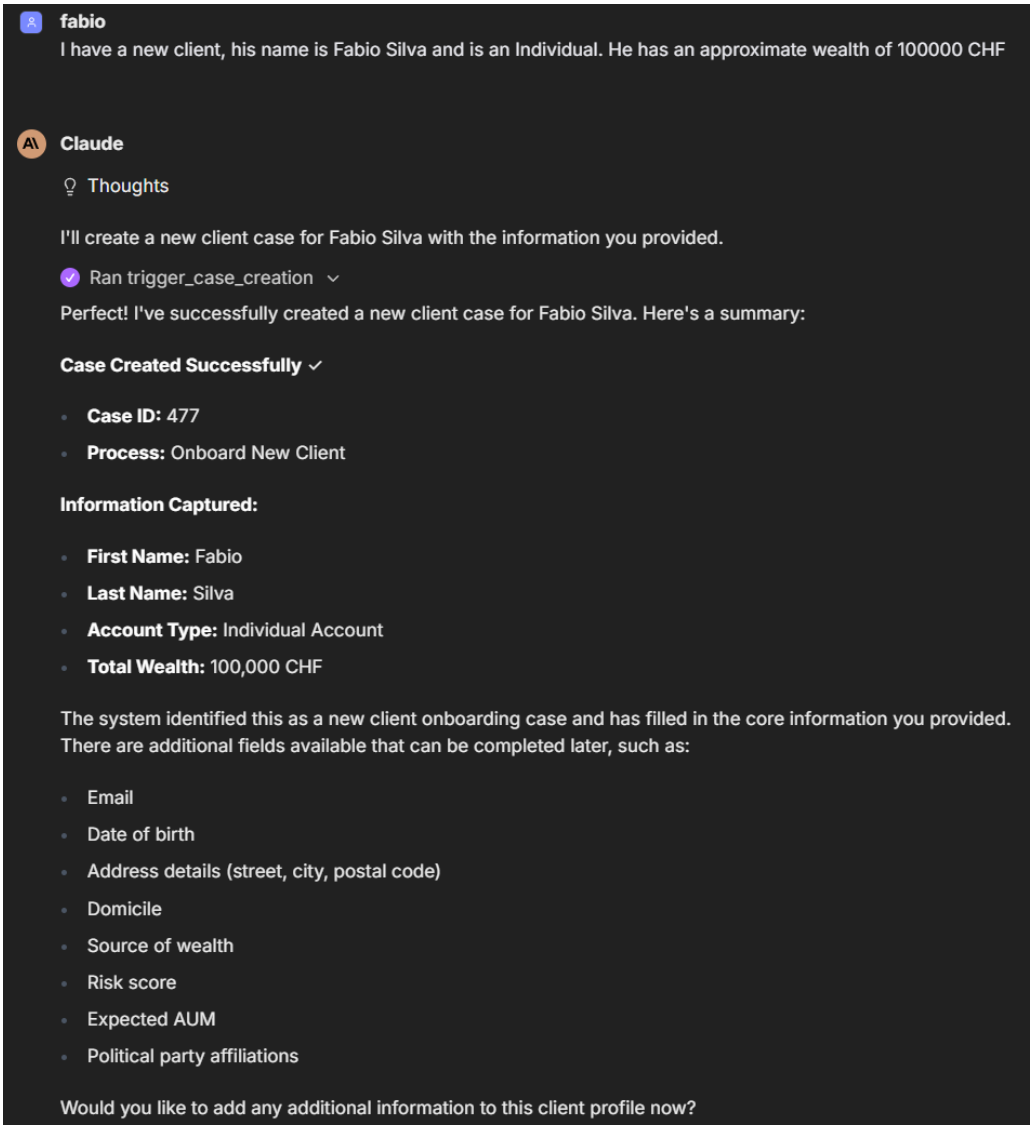


Figure A.1: Successful Case Creation

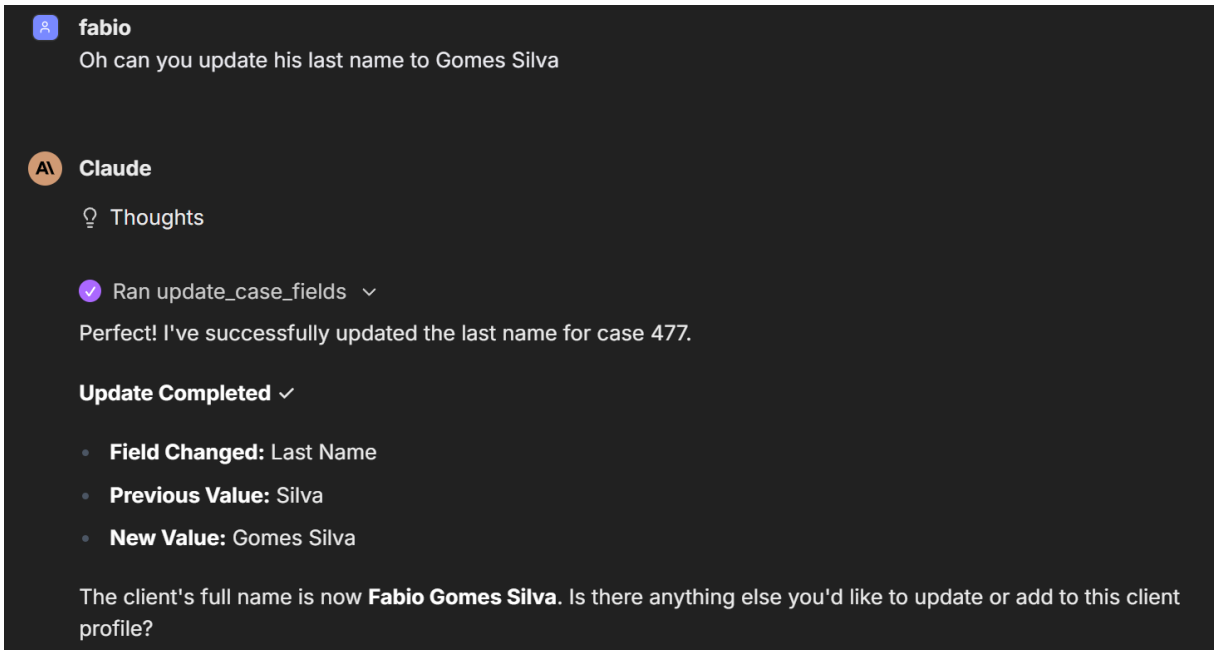


Figure A.2: Successful Field update

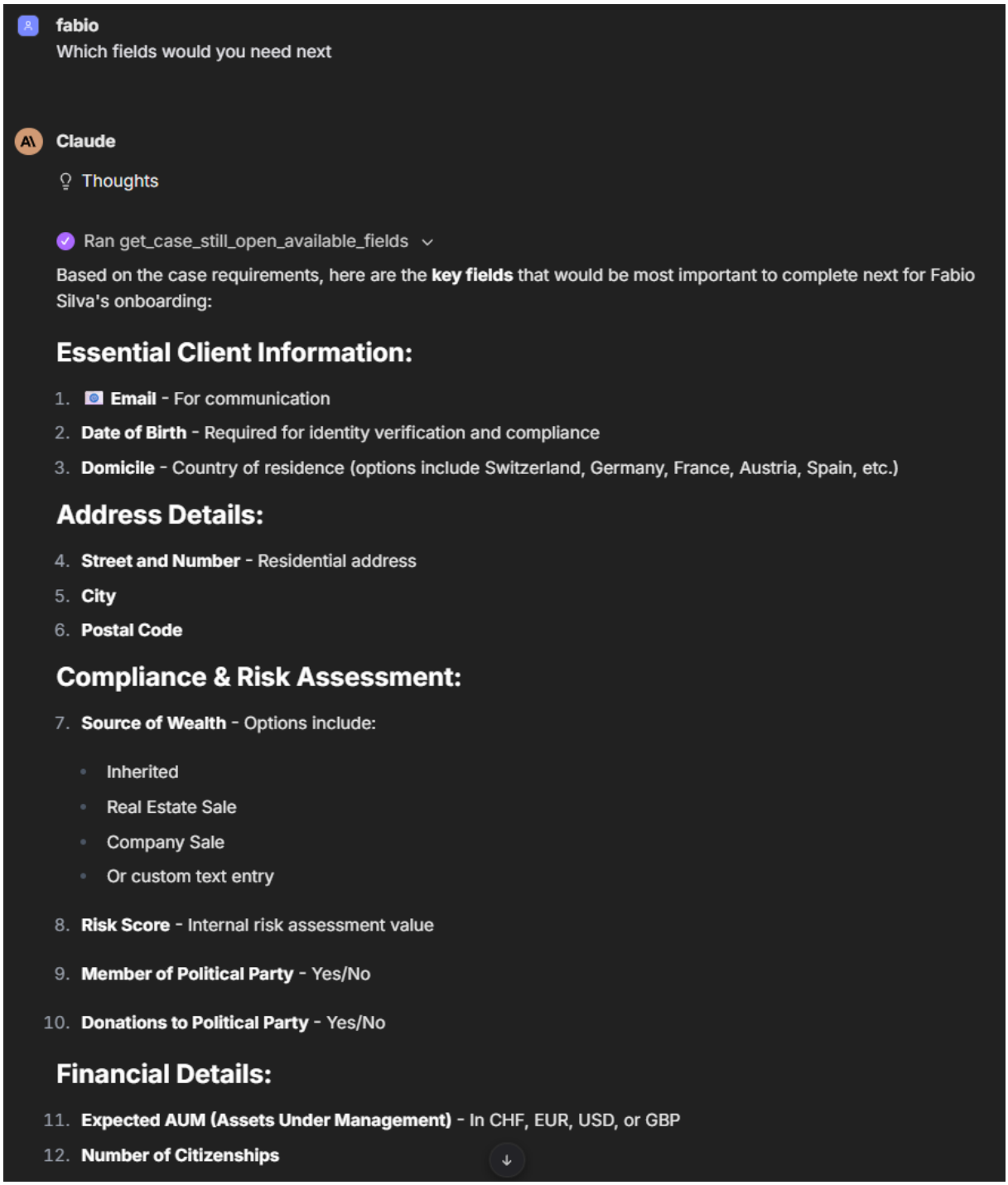


Figure A.3: Successful Retrieval of remaining, not yet filled, fields

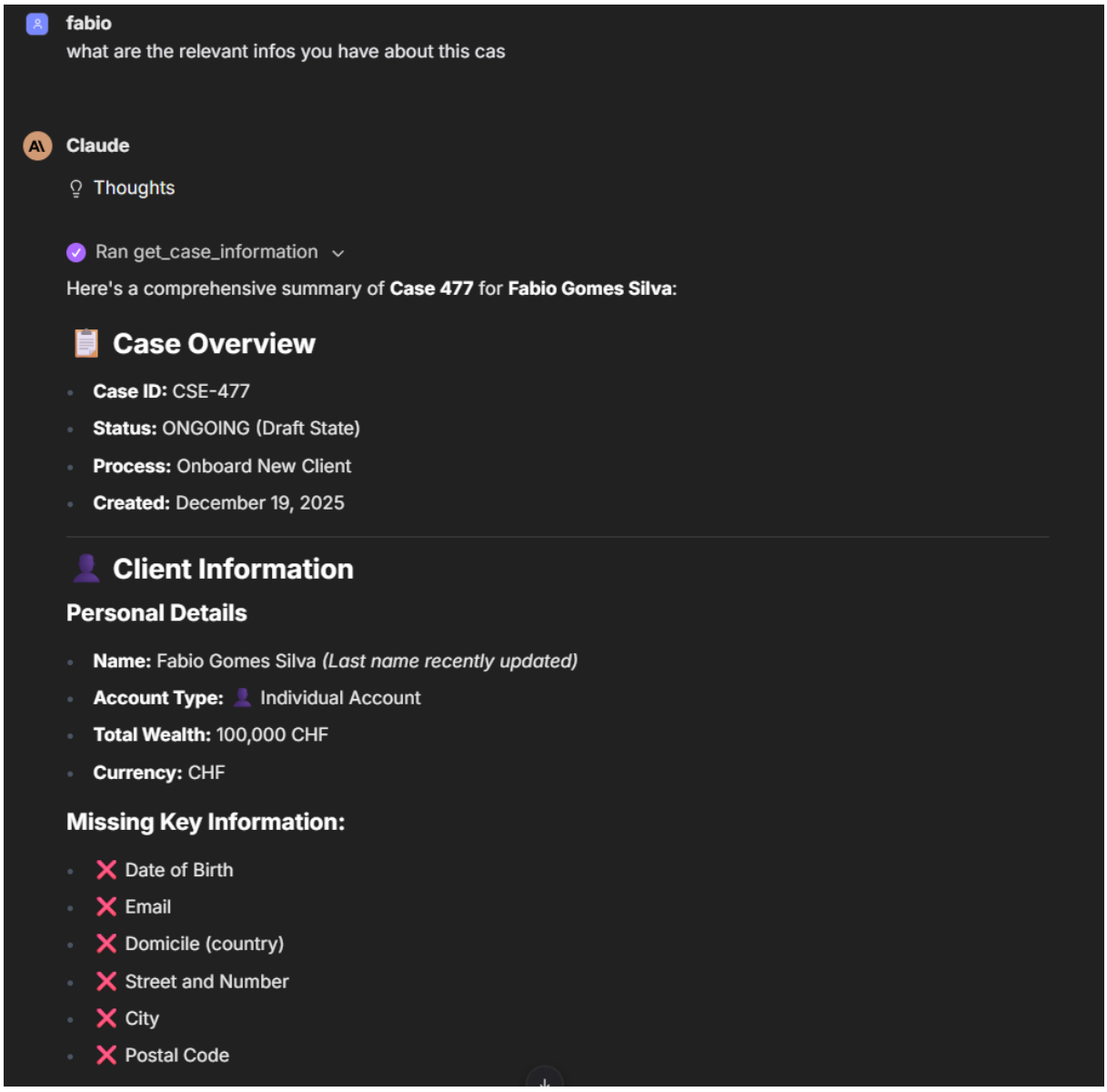


Figure A.4: Successful Retrieval of important information about the case (Part 1)

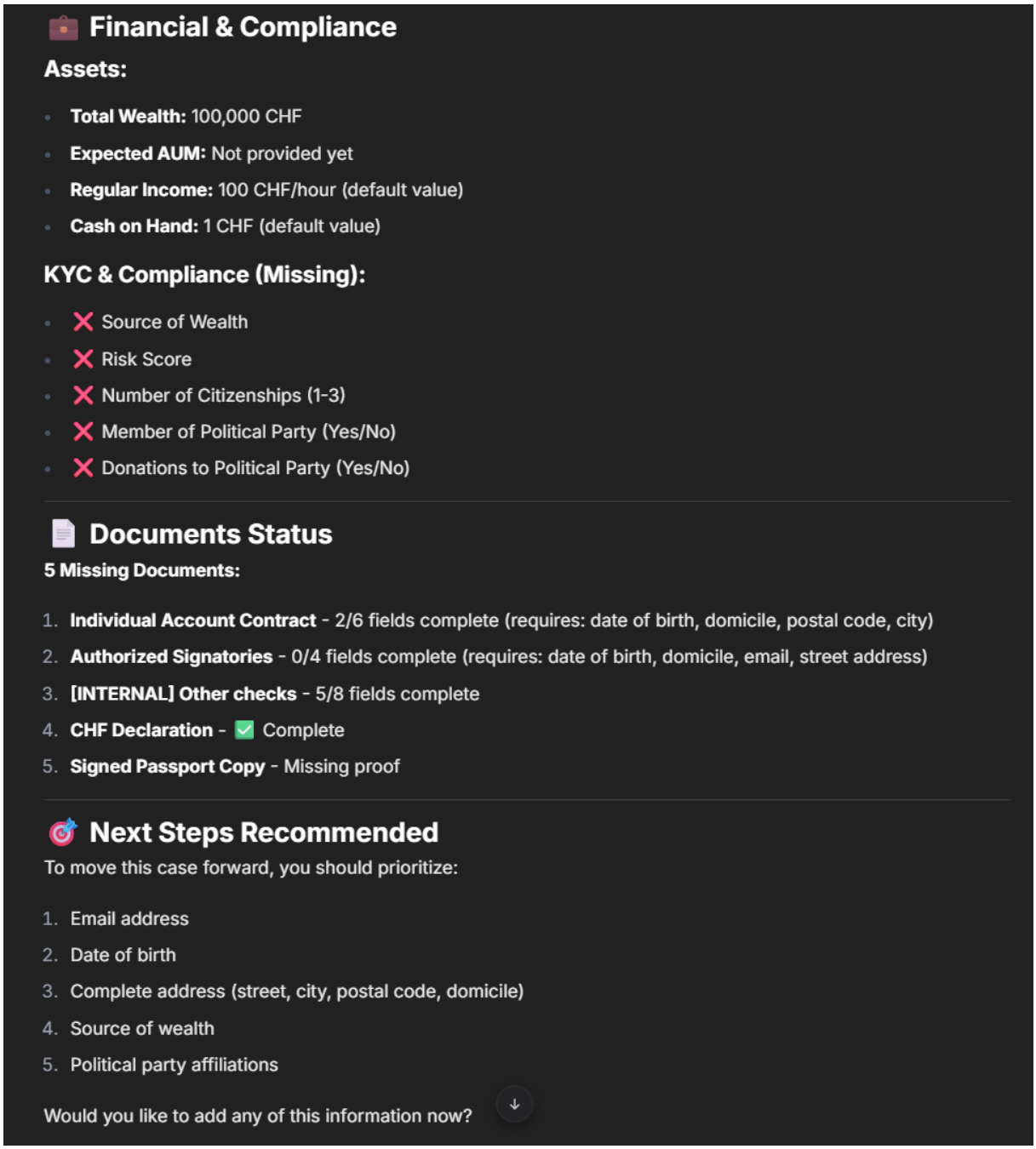


Figure A.5: Successful Retrieval of important information about the case (Part 2)