

Département Mathématiques et Informatique

Rapport du Stage Ingénieur

Filière :

« Ingénierie Informatique : Big Data et Cloud Computing »

II-BDCC

One Place Chat:

Interacting with APIs Through the Model
Context Protocol



Soutenu le 09 /12/2026

Réalisé par :

EL BADRY Mohammed

Encadré par :

LAANAIT Ismail

Année Universitaire : 2025-2026

Acknowledgments

At the completion of this work, I would like to express my sincere gratitude to the Director of École Normale Supérieure de l'Enseignement Technique de Mohammedia (ENSET-M), **M. Bouattane Omar**, for providing an environment conducive to learning and scientific development.

I also thank the Head of the Mathematics and Computer Science Department, **Pr. QBA-DOU Mohammed**, for his commitment to organizing and ensuring the smooth progress of internship projects.

My thanks also go to the Coordinator of the Computer Engineering : Big Data and Cloud Computing program, **Pr. EN-NAIMANI Zakariae**, for his rigorous follow-up and constant support throughout these academic years.

I would like to sincerely thank the jury members who agreed to evaluate this work. Their expertise, constructive remarks, and insightful advice constitute a valuable contribution to the completion of this project.

I express my special gratitude to my internship supervisor at Zenika, **M. Ismail Laanait**, for his guidance, availability, and valuable insights throughout this internship journey. His mentorship was instrumental in bringing this project to fruition.

I also extend my thanks to the entire teaching staff of the program for the excellence of the training provided and the transmission of knowledge that formed the foundation of this work.

Finally, I express my deepest gratitude to my family and loved ones for their unwavering support and encouragement, which have been an essential pillar throughout this academic journey.

Table des matières

Acknowledgments	1
Introduction	1
1 General Project Context	2
1.1 Introduction	2
1.2 The Host Organization : Zenika	2
1.2.1 Overview of the Company	2
1.2.2 Technical Fact Sheet	2
1.2.3 Geographic Presence and History	3
1.3 Organizational Structure and Culture	3
1.3.1 A Consultant-Centric Model	3
1.3.2 Corporate Vision : "Code the World"	3
1.3.2.1 Knowledge Sharing Initiatives	3
1.3.2.2 Continuous Learning	3
1.3.2.3 Community Support	4
1.4 Services and Strategic Partnerships	4
1.4.1 Service Offerings	4
1.4.2 Strategic Partnerships	4
2 Model Context Protocol (MCP)	5
2.1 Introduction to MCP	5
2.1.1 The Problem MCP Solves	5
2.2 MCP Architecture Overview	5
2.2.1 Client-Server Architecture	5
2.2.2 Key Participants	6
2.3 MCP Primitives	6
2.3.1 Server Primitives	6
2.3.2 Client Primitives	6
2.4 MCP Communication Flow	7
2.4.1 Understanding the MCP Workflow	7
2.4.1.1 The Discovery Phase	7
2.4.1.2 The Tool Selection Phase	7
2.4.1.3 Limitations and Design Implications	8
2.5 Conclusion	8
3 Problem and Proposed Solution	9
3.1 Introduction	9
3.2 Traditional API Integration Challenges	9
3.2.1 The Manual Integration Problem	9
3.2.1.1 Manual Tool Definition	9
3.2.2 Identified Limitations	9

3.2.3	Real-World Impact	10
3.3	Proposed Solution : One-Place-Chat	10
3.3.0.1	Core Innovation	10
3.3.1	Architecture of the Solution	10
3.3.1.1	Stage 1 : Automated Tool Generation	10
3.3.1.2	Stage 2 : Vector-Based Storage with ChromaDB	11
3.3.1.3	Stage 3 : Semantic Matching and Natural Language Processing	11
3.3.2	Implementation Strategy	11
3.3.3	Key Advantages	12
3.4	Conclusion	12
4	System Architecture and Technical Stack	13
4.1	Introduction	13
4.2	Overall System Architecture	13
4.2.1	High-Level Architecture	13
4.2.2	Component Overview	13
4.3	Database Architecture	14
4.3.1	Vector Database Strategy	14
4.3.2	ChromaDB Collections	14
4.3.2.1	Collection Descriptions	15
4.3.2.2	Collection Relationships	15
4.4	System Class Diagram	15
4.4.1	Class Responsibilities	16
4.5	Conclusion	16
5	Realisation	17
5.1	Introduction	17
5.2	Tools and Technologies Used	17
5.2.1	Frontend Technologies	17
5.2.2	Backend Technologies	17
5.2.3	Database and Storage	18
5.2.4	Development Tools	18
5.3	Frontend Architecture	18
5.3.1	Component Hierarchy	18
5.3.2	Key Components	18
5.4	Backend Architecture	19
5.4.1	Core Engine Workflow	19
5.5	Application Interfaces	19
5.5.1	Main Chat Interface	19
5.5.2	Chat Message Response	20
5.5.3	Tool Description Panel	21
5.5.4	Documentation Upload	22
5.6	Conclusion	22
	General Conclusion	23

Table des figures

2.1	MCP Client-Server Architecture	5
2.2	MCP Communication Workflow	7
3.1	MCP Communication Workflow	10
3.2	MCP Communication Workflow	11
3.3	MCP Communication Workflow	11
4.1	Three-Tier System Architecture	13
4.2	ChromaDB Vector Database Architecture	14
4.3	ChromaDB Collection Schema and Relationships	14
4.4	Backend System Class Diagram	15
5.1	Frontend Component Hierarchy	18
5.2	Backend Request Processing Flow	19
5.3	One Place Chat Main Interface	19
5.4	Chat Message Response Display	20
5.5	Tool Description Panel	21
5.6	OpenAPI Documentation Upload Interface	22

General Introduction

As part of our engineering studies at **École Normale Supérieure de l'Enseignement Technique de Mohammedia (ENSET-M)**, I undertook an internship at **Zenika**, a leading technology consulting company. This internship provided an exceptional opportunity to apply our academic knowledge in artificial intelligence and software development to real-world challenges while working alongside experienced professionals.

The modern software landscape is characterized by an ever-growing ecosystem of APIs that power everything from mobile applications to enterprise systems. However, interacting with these APIs traditionally requires technical expertise : understanding endpoint structures, authentication mechanisms, request parameters, and response formats. This technical barrier limits access to powerful tools and services for non-technical users and creates friction even for developers working with unfamiliar APIs.

It is within this context that **One Place Chat** was conceived—a conversational platform designed to democratize API interactions through natural language. Rather than requiring users to learn specific API syntaxes and structures, One Place Chat enables them to describe their intentions in plain English and automatically translates these requests into precise API calls.

The innovation of One Place Chat lies in its intelligent three-stage architecture : automatic tool generation from OpenAPI specifications, semantic tool discovery using vector embeddings, and natural language parameter extraction powered by large language models. This combination creates a seamless experience where complex technical operations become as simple as having a conversation.

This report details the complete design and development journey of One Place Chat, presenting the architectural decisions, technologies employed, and features implemented. It demonstrates our methodological approach and the practical application of AI technologies to solve real accessibility challenges in software interaction.

Chapter 1

General Project Context

1.1 Introduction

This chapter outlines the general framework of the end-of-studies project. It begins with a detailed presentation of the host organization, Zenika, including its history, structure, and areas of expertise. Following this, the chapter introduces the project context, related concepts, and the methodology adopted for the solution's realization, concluding with the project planning.

1.2 The Host Organization : Zenika

1.2.1 Overview of the Company



Zenika is a leading French technology consultancy and training firm founded in 2006. Specializing in digital transformation, organizational agility, and software architecture, Zenika acts as a bridge between the organic world and the digital world. The company supports its clients—ranging from startups to large enterprises—in their technological evolution by offering high-end services in consulting, training, and custom software development.

1.2.2 Technical Fact Sheet

To provide a concise overview of the organization, the following table summarizes Zenika's key identity and operational metrics.

Characteristic	Description
Company Name	Zenika
Founded	2006
Headquarters	Paris, France
Sector	IT Consulting, Software Development
Workforce	+600 employees (Consultants, Developers...)
Global Presence	France, Canada, Singapore, Morocco
Number of Agencies	14 (including Paris, Lyon, Casablanca, etc.)

TABLE 1.1 – Zenika Technical Fact Sheet

1.2.3 Geographic Presence and History

Since its inception, Zenika has pursued a strategy of proximity to its clients and consultants. This has led to a rapid geographic expansion.

- **France** : The network includes agencies in major tech hubs such as Paris, Rennes, Nantes, Bordeaux, Lille, Lyon, Grenoble, Clermont-Ferrand, Niort, and Brest.
- **International** : Zenika has expanded beyond French borders with established agencies in Montreal (Canada), Singapore, and Casablanca (Morocco).

This distributed structure allows Zenika to combine the agility of local teams with the strength and knowledge base of a global group.

1.3 Organizational Structure and Culture

1.3.1 A Consultant-Centric Model

Zenika's organizational structure is designed to foster autonomy and expertise. The workforce is primarily composed of highly qualified profiles :

- **Consultants** : Experts who integrate into client teams to provide guidance and technical leadership.
- **Trainers** : Professionals who deliver certified training courses to external clients.
- **Developers** : Full-stack engineers capable of building complex solutions from scratch.

1.3.2 Corporate Vision : "Code the World"

Zenika's vision extends beyond commercial success; it aims to impact the technological landscape positively. This vision is supported by several internal and external pillars :

1.3.2.1 Knowledge Sharing Initiatives

- **TechnoZaures** : Internal conference days dedicated to technical exchange among employees.
- **Lunch & Learn** : Informal lunchtime sessions where collaborators present on specific topics.
- **NightClazz** : Evening meetups open to the public, allowing deep dives into specific technologies.

1.3.2.2 Continuous Learning

- **Learning Expeditions** : Zenika organizes trips to major tech hubs (like Silicon Valley) and exchanges between agency offices to foster cross-pollination of ideas.
- **Conference Access** : Consultants are encouraged to attend and speak at national and international tech conferences, ensuring they remain at the cutting edge of industry trends.

1.3.2.3 Community Support

Zenika is a strong supporter of the Open Source community and local tech ecosystems. The company sponsors numerous agile and technical conferences and hosts community events in its offices.

1.4 Services and Strategic Partnerships

1.4.1 Service Offerings

Zenika's value proposition is built around three main axes :

1. **Consulting** : Auditing existing systems, defining architectural strategies, and guiding digital transformation (Agile, DevOps adoption).
2. **Training** : Offering a comprehensive catalog of training courses. Zenika is often an official training partner for major technologies.
3. **Realization** : Developing custom software solutions, MVPs, and industrializing products using modern stacks.

1.4.2 Strategic Partnerships

To deliver the best solutions, Zenika maintains strong partnerships with industry leaders. These alliances allow Zenika to offer certified expertise and early access to new technologies.

Key partners include :

- **Cloud & Infrastructure** : AWS, Google Cloud, Docker, Kubernetes.
- **Data & Search** : Elastic, Confluent (Kafka), MongoDB.
- **Development Ecosystem** : Spring, GitHub, GitLab.

These partnerships reinforce Zenika's capability to handle complex, large-scale projects and provide objective, expert advice on tool selection.

Chapter 2

Model Context Protocol (MCP)

2.1 Introduction to MCP

The Model Context Protocol (MCP) is an open protocol standardized by Anthropic that enables seamless integration between AI applications and external data sources. In the context of our project, MCP serves as the foundational layer that allows the AI assistant to access and interact with the knowledge library, providing contextual information to enhance conversational responses.

2.1.1 The Problem MCP Solves

AI-enabled tools are powerful, but they're often limited to the information manually provided or require bespoke integrations. Traditional AI applications face several challenges :

- **Limited Context** : AI models can only work with information provided in prompts
- **Manual Data Provision** : Users must manually copy and paste relevant information
- **Bespoke Integrations** : Each data source requires custom, hard-coded integrations
- **No Standardization** : Different AI applications implement data access differently

Whether it's reading files from your computer, searching through an internal or external knowledge base, or updating tasks in a project management tool, MCP provides a **secure, standardized, and simple** way to give AI systems the context they need.

2.2 MCP Architecture Overview

2.2.1 Client-Server Architecture

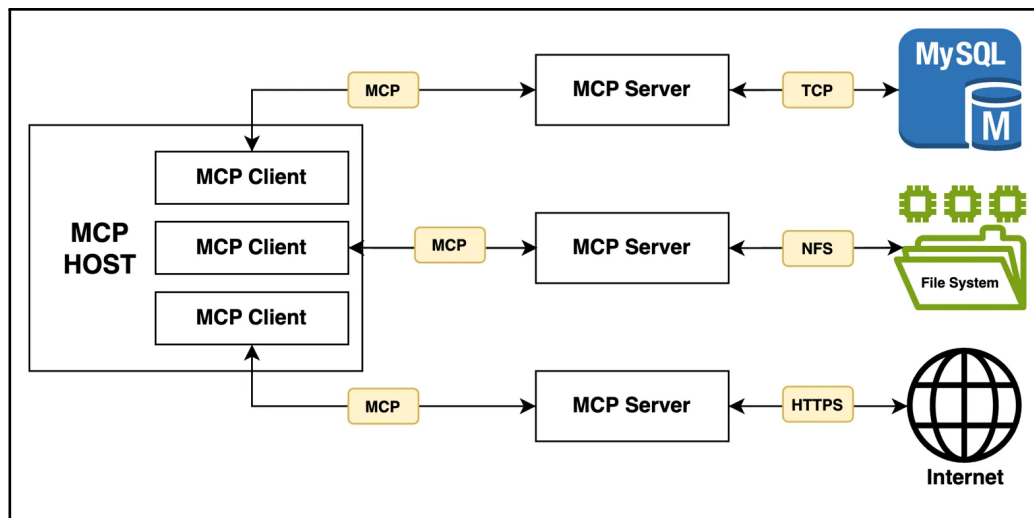


FIGURE 2.1 – MCP Client-Server Architecture

2.2.2 Key Participants

The MCP architecture involves three primary participants :

1. **MCP Host** : The AI application that coordinates and manages one or multiple MCP clients (e.g., Visual Studio Code, Claude Desktop, One Place Chat)
2. **MCP Client** : A component that maintains a connection to an MCP server and obtains context from an MCP server for the MCP host to use
3. **MCP Server** : A program that provides context to MCP clients. Servers can run locally (using STDIO transport) or remotely (using HTTP transport)

2.3 MCP Primitives

MCP defines primitives that specify what context can be shared between clients and servers. These are the core building blocks of the protocol.

2.3.1 Server Primitives

Servers expose three main types of primitives to provide context to AI applications :

Primitive	Description
Tools	Executable functions that AI applications can invoke to perform actions (e.g., file operations, API calls, database queries). Tools are discovered via <code>tools/list</code> and executed via <code>tools/call</code> .
Resources	Data sources that provide contextual information to AI applications (e.g., file contents, database records, API responses). Accessed via <code>resources/list</code> and <code>resources/read</code> .
Prompts	Reusable templates that help structure interactions with language models (e.g., system prompts, few-shot examples). Retrieved via <code>prompts/list</code> and <code>prompts/get</code> .

TABLE 2.1 – MCP Server Primitives

2.3.2 Client Primitives

Clients expose primitives that allow servers to build richer interactions :

- **Sampling** : Allows servers to request language model completions from the client's AI application using `sampling/complete`
- **Elicitation** : Enables servers to request additional information from users via `elicitation/request`
- **Logging** : Allows servers to send log messages to clients for debugging and monitoring

2.4 MCP Communication Flow

The typical MCP interaction follows a well-defined sequence :

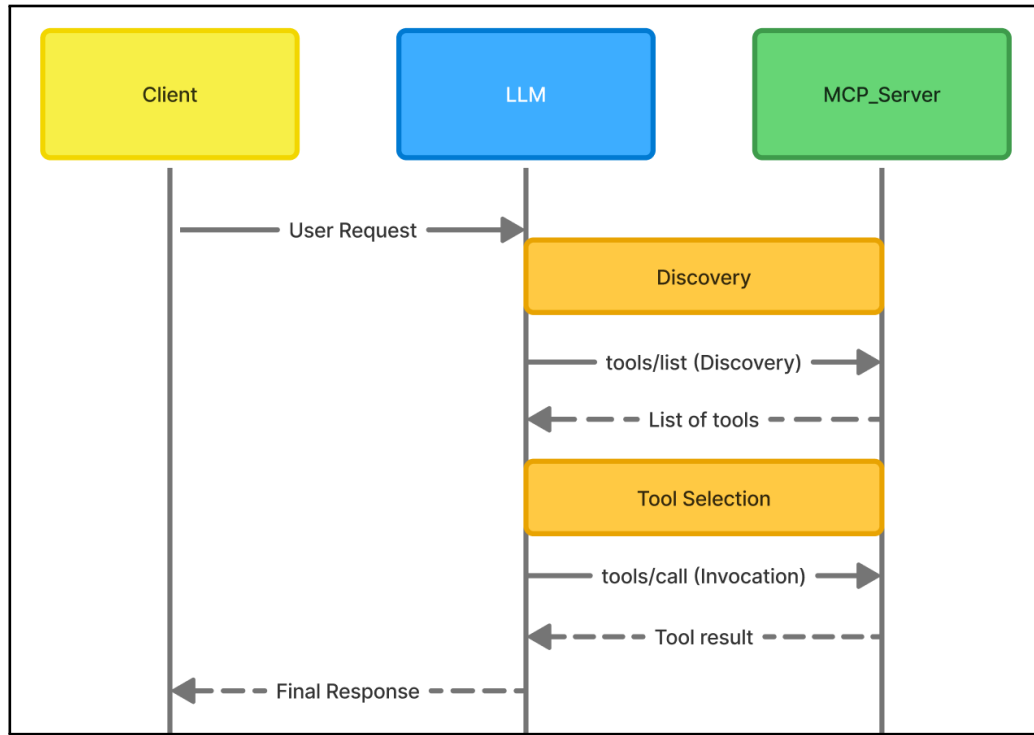


FIGURE 2.2 – MCP Communication Workflow

2.4.1 Understanding the MCP Workflow

2.4.1.1 The Discovery Phase

When a user sends a request to the AI application through the client interface, the Large Language Model (LLM) initiates a discovery phase :

1. The LLM sends a `tools/list` request to the MCP server
2. The MCP server responds with a comprehensive list of all available tools that have been previously implemented and stored
3. Each tool in the list includes its name, description, and input schema (parameters it accepts)

2.4.1.2 The Tool Selection Phase

Once the LLM receives the list of available tools, it enters the tool selection phase :

1. The LLM analyzes the user's request against the available tools
2. Based on the tool descriptions and schemas, the LLM intelligently selects the most appropriate tool(s) to fulfill the user's request

3. The LLM then invokes the selected tool using `tools/call` with appropriate arguments
4. The MCP server executes the pre-coded tool logic and returns the result
5. The LLM incorporates the tool result into its response to the user

2.4.1.3 Limitations and Design Implications

This architecture has important implications for system design :

- **Pre-implementation Required** : Every capability the AI needs must be implemented as a tool beforehand
- **Schema Definition** : Each tool must have a well-defined input schema so the LLM knows how to invoke it correctly
- **Limited Flexibility** : The AI cannot create new tools or capabilities on the fly ; it can only use what has been pre-coded
- **Deterministic Behavior** : Since tools are pre-defined, their behavior is predictable and can be tested thoroughly

This design ensures security, reliability, and predictability, as the AI can only perform actions that have been explicitly programmed and authorized by developers.

2.5 Conclusion

The Model Context Protocol represents a significant advancement in how AI applications interact with external data sources. By providing a standardized, secure framework for context exchange, MCP eliminates the need for bespoke integrations while maintaining strict control over AI capabilities through pre-defined tools.

Chapter 3

Problem and Proposed Solution

3.1 Introduction

Integrating AI assistants with external APIs presents significant challenges in traditional implementations. This chapter examines these challenges and presents One Place Chat’s innovative solution that enables natural language API interactions through intelligent tool generation and semantic matching.

3.2 Traditional API Integration Challenges

3.2.1 The Manual Integration Problem

Traditional approaches to integrating AI systems with APIs face several fundamental challenges that limit scalability and user experience.

3.2.1.1 Manual Tool Definition

The traditional workflow requires developers to manually write code for each API endpoint the AI needs to access. This includes defining request structures, parameter schemas, response parsing, and error handling. For an API with dozens or hundreds of endpoints, this becomes a massive development bottleneck.

3.2.2 Identified Limitations

This manual approach introduces several practical limitations :

- **Development Bottleneck** : Each new API endpoint requires separate implementation, testing, and deployment cycles. Adding support for a new API service can take days or weeks of development time.
- **Maintenance Burden** : As APIs evolve and endpoints change, each integration must be manually updated. API versioning requires duplicate implementations or complex version management logic.
- **Limited Scalability** : Supporting multiple APIs with hundreds of endpoints becomes impractical when each requires manual coding. The codebase grows linearly with the number of supported endpoints.
- **Poor User Experience** : Users must learn specific command syntax or interact with rigid interfaces. Natural language understanding is limited or requires extensive prompt engineering.

3.2.3 Real-World Impact

Consider a practical scenario : An organization wants to integrate their AI assistant with three different API services (user management, inventory, and analytics), each with 50+ endpoints. In a traditional implementation, this would require :

- Manual coding of 150+ individual endpoint handlers
- Custom parameter validation logic for each endpoint
- Separate error handling for each API service

This approach quickly becomes unsustainable as the number of APIs and endpoints grows.

3.3 Proposed Solution : One-Place-Chat

One Place Chat introduces an innovative three-stage approach that transforms API integration from a manual coding process into an automated, intelligent system :

1. **Tool Generation** : Convert OpenAPI specifications into executable **tool definitions**
2. **Vector-Based Storage** : Store tools with semantic embeddings for intelligent retrieval
3. **Natural Language Processing** : Use LLM-powered semantic matching and parameter extraction

3.3.0.1 Core Innovation

The key innovation lies in shifting from **manual API coding** to **automated semantic tool discovery** :

- **Traditional Approach** : Manual coding → Fixed endpoints → Limited scalability
- **One Place Chat** : OpenAPI parsing → Vector embeddings → Semantic matching → Natural language processing

3.3.1 Architecture of the Solution

3.3.1.1 Stage 1 : Automated Tool Generation

Instead of manually coding API integrations, One Place Chat automatically generates tool definitions from OpenAPI specifications :

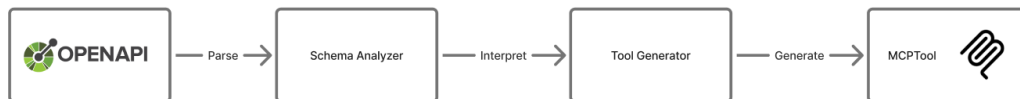


FIGURE 3.1 – MCP Communication Workflow

- Extracts all API endpoints and operations
- Generates input schemas from parameter definitions
- Creates tool descriptions from API documentation
- Produces standardized **MCPTool** objects

3.3.1.2 Stage 2 : Vector-Based Storage with ChromaDB

Generated tools are stored in a vector database with semantic embeddings for intelligent retrieval :

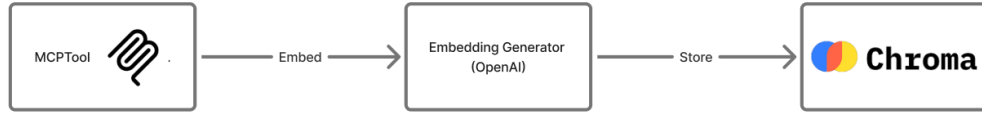


FIGURE 3.2 – MCP Communication Workflow

For each tool, the system :

- Creates a text representation combining name, description, path, and tags
- Generates vector embeddings using OpenAI’s ‘**text-embedding-ada-002**’ model
- Stores the tool definition and embedding in ChromaDB collections
- Enables semantic similarity search for intelligent tool matching

3.3.1.3 Stage 3 : Semantic Matching and Natural Language Processing

When users make requests in natural language, the system uses semantic search to find the right tool :

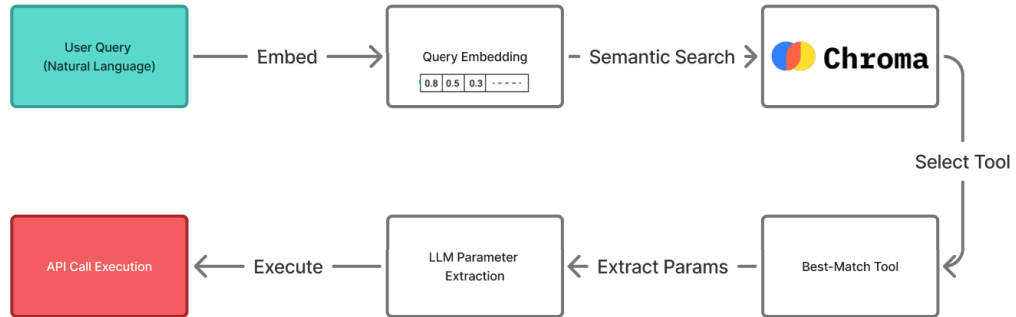


FIGURE 3.3 – MCP Communication Workflow

3.3.2 Implementation Strategy

Our solution implements the three-stage approach through specialized components :

1. **OpenAPI Tool Parser** : Automatically converts OpenAPI specifications into executable tool definitions with complete schemas and parameter mappings.
2. **ChromaDB Tool Loader** : Dynamically loads tools from the vector database without caching, ensuring fresh data and providing semantic search capabilities.
3. **ChromaDB Tool Matcher** : Generates OpenAI embeddings for tools and performs similarity-based matching using vector distances to find the most relevant tool.

3.3.3 Key Advantages

Aspect	Traditional Approach	One Place Chat
Tool Creation	Manual coding per endpoint	Automatic from OpenAPI specs
Scalability	Linear development effort	Handles unlimited endpoints
Intent Matching	Keyword-based, rigid	Semantic embeddings, flexible
Maintenance	Update each integration	Re-parse updated spec
Development Time	Days per API	Minutes per API
User Interface	Specific syntax required	Natural language

TABLE 3.1 – Comparison : Traditional Approach vs One Place Chat

3.4 Conclusion

One Place Chat fundamentally transforms API integration by introducing a three-stage architecture that eliminates manual coding entirely : OpenAPI specifications are automatically parsed into tool definitions, stored as vector embeddings in ChromaDB for semantic retrieval, and matched to user intent through natural language processing. This innovation shifts the paradigm from labor-intensive, endpoint-by-endpoint development to an intelligent, automated system where adding a new API takes minutes instead of weeks, where users interact conversationally instead of learning rigid syntax, and where semantic understanding replaces brittle keyword matching. The result is a scalable, maintainable solution that demonstrates how combining standardized API specifications with modern vector databases and large language models can create truly intelligent interfaces that understand intent, adapt dynamically, and deliver superior user experience without sacrificing technical accuracy or reliability.

Chapter 4

System Architecture and Technical Stack

4.1 Introduction

This chapter presents the technical foundation of One Place Chat, focusing on the overall system architecture, database design, and technology stack. We begin with a high-level architectural overview, detail the database schema for both relational and vector storage, present the technologies employed, and conclude with the class diagram illustrating the system's core structure.

4.2 Overall System Architecture

4.2.1 High-Level Architecture

One Place Chat follows a modern three-tier architecture with clear separation of concerns :

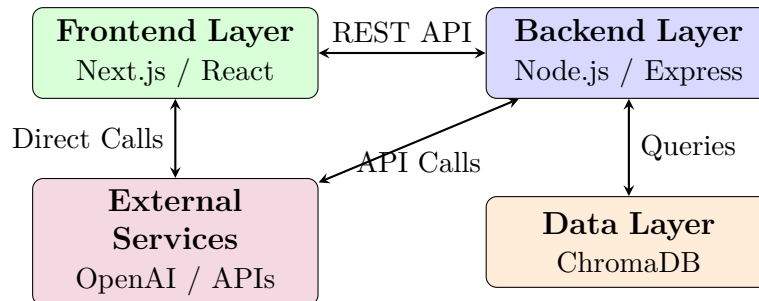


FIGURE 4.1 – Three-Tier System Architecture

4.2.2 Component Overview

The system consists of four primary layers :

1. **Frontend Layer** : User interface built with Next.js and React, handling conversations, knowledge library, and tool management.
2. **Backend Layer** : Business logic using Node.js and Express, including the conversational engine, semantic tool matching, and API orchestration.
3. **Data Layer** : ChromaDB vector database storing tools, conversations, and messages with semantic embeddings.
4. **External Services** : Integration with OpenAI for LLMs and embeddings, plus dynamically discovered third-party APIs.

4.3 Database Architecture

4.3.1 Vector Database Strategy

One Place Chat uses ChromaDB as the primary database, storing all data as vector embeddings with associated metadata :

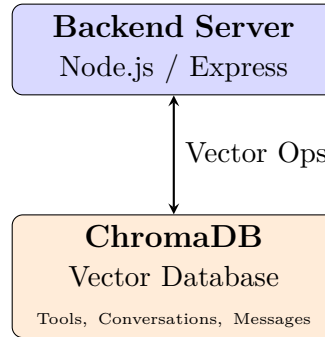


FIGURE 4.2 – ChromaDB Vector Database Architecture

4.3.2 ChromaDB Collections

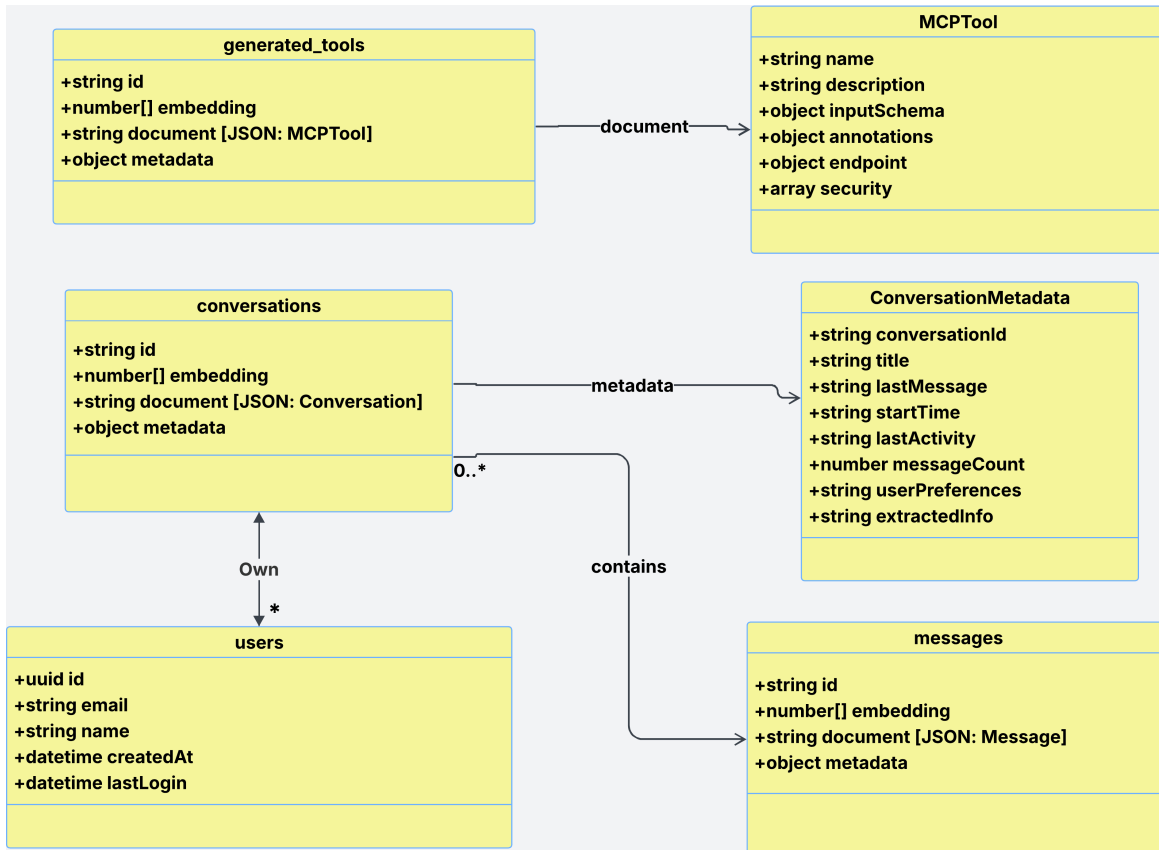


FIGURE 4.3 – ChromaDB Collection Schema and Relationships

4.3.2.1 Collection Descriptions

Users Collection : Stores user profiles with unique identifiers, email addresses, display names, and roles. Each user record includes timestamps for account creation and last login activity.

Generated Tools Collection : Contains API tool definitions automatically generated from OpenAPI specifications. Each tool includes a 1536-dimensional embedding vector from OpenAI's `text-embedding-ada-002` model, enabling semantic similarity search. The document field stores the complete MCPTool definition including name, description, HTTP method, and endpoint path.

Conversations Collection : Manages conversation contexts with references to the owning user. Each conversation stores its embedding for similarity-based retrieval, the complete conversation context as JSON, title, message count, and activity timestamps.

Messages Collection : Stores individual messages belonging to conversations. Each message contains its role (user, assistant, or system), content, timestamp, and an embedding for fine-grained semantic search within conversation history.

4.3.2.2 Collection Relationships

The collections are linked through the following relationships :

- **Users** → **Conversations** : One user can own multiple conversations (1 :N relationship)
- **Conversations** → **Messages** : One conversation contains multiple messages

4.4 System Class Diagram

The following class diagram illustrates the main backend components and their relationships :

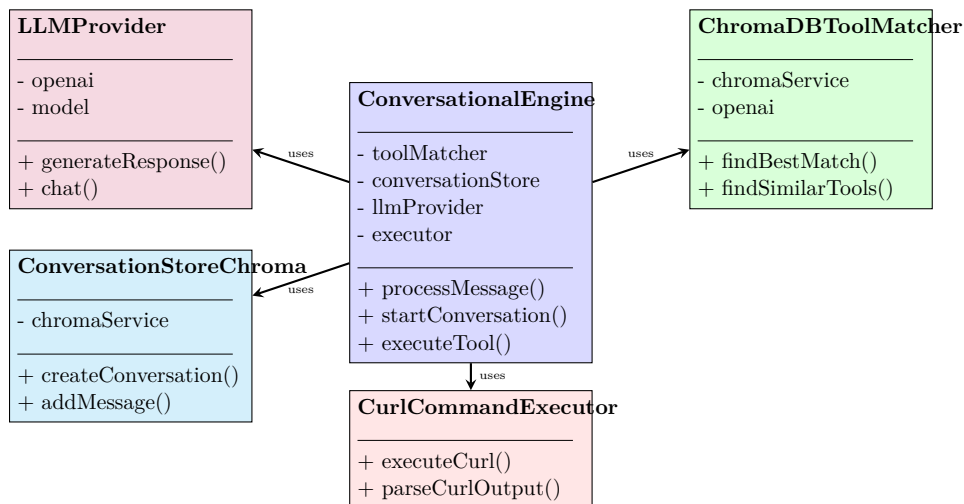


FIGURE 4.4 – Backend System Class Diagram

4.4.1 Class Responsibilities

- **ConversationalEngine** : Main orchestrator coordinating tool matching, parameter extraction, LLM interactions, and API execution.
- **ChromaDBToolMatcher** : Performs semantic matching between user queries and available tools using OpenAI embeddings.
- **ChromaDBService** : Low-level client for ChromaDB operations including embedding storage and similarity search.
- **LLMProvider** : Abstraction layer for large language model interactions, supporting OpenAI GPT models.
- **ConversationStoreChroma** : Manages conversation persistence and message history in ChromaDB.
- **CurlCommandExecutor** : Generates and executes cURL commands to call external APIs.

4.5 Conclusion

The One Place Chat architecture combines a modern three-tier design with a dual-database strategy optimized for both structured data management and AI-powered semantic search. The technology stack leverages industry-standard tools while the class structure ensures maintainability through clear separation of concerns. This foundation enables the system to automatically integrate new APIs, understand natural language queries through vector similarity, and deliver intelligent responses reliably.

Chapter 5

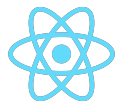
Realisation

5.1 Introduction

This chapter presents the implementation details of One Place Chat, covering the technologies used, frontend and backend architecture, and the key features that make the system unique. We demonstrate how the theoretical concepts discussed in previous chapters translate into a working application.

5.2 Tools and Technologies Used

5.2.1 Frontend Technologies



Next.js 14 / React 18

React-based framework providing server-side rendering, file-based routing, and optimized production builds. Used for building the entire user interface including chat components, sidebar navigation, and tools management panel.



Tailwind CSS

Utility-first CSS framework enabling rapid UI development with responsive design. Combined with shadcn/ui component library for accessible, customizable interface elements.



TypeScript

Statically-typed JavaScript providing type safety, enhanced IDE support, and improved code maintainability across both frontend and backend codebases.

5.2.2 Backend Technologies



Express.js

Minimalist web framework for building RESTful APIs with middleware support. Handles all HTTP requests, routing, and API endpoint management.



OpenAI API

Integration with OpenAI's GPT models for natural language understanding and the `text-embedding-ada-002` model for generating 1536-dimensional semantic embeddings used in tool matching.

5.2.3 Database and Storage

**ChromaDB**

Open-source vector database designed for AI applications. Stores all application data including tools, conversations, and messages with vector embeddings for semantic similarity search.

5.2.4 Development Tools

**Git / GitHub**

Version control and collaboration platform for source code management, feature branching, and code review workflows.

**Postman**

API development and testing platform used for testing backend endpoints, validating request/response structures, and debugging API integrations.

5.3 Frontend Architecture

5.3.1 Component Hierarchy

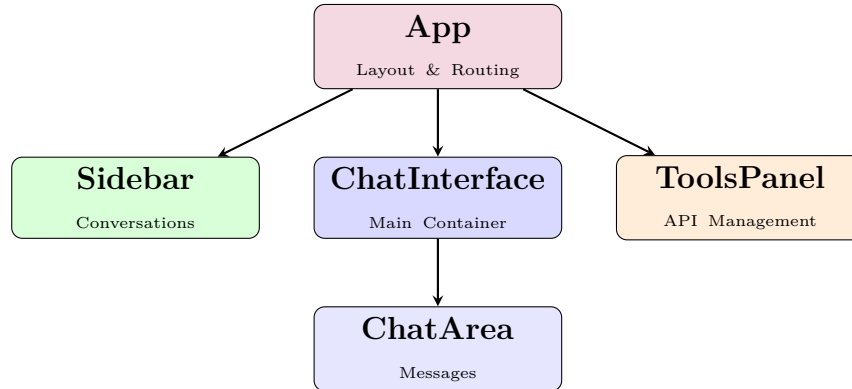


FIGURE 5.1 – Frontend Component Hierarchy

5.3.2 Key Components

Sidebar : Manages conversation navigation, displaying previous sessions with titles and timestamps, and providing new conversation creation.

ChatInterface : Main container orchestrating conversation state, API communication, and real-time message streaming between components.

ChatArea : Renders message bubbles with role-based styling, displays tool execution results, and handles clarification requests with markdown support.

ToolsPanel : Provides OpenAPI specification upload, displays available tools with metadata (method, path, parameters), and manages tool refresh operations.

5.4 Backend Architecture

5.4.1 Core Engine Workflow

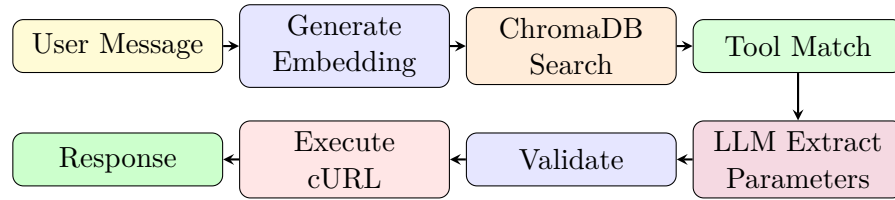


FIGURE 5.2 – Backend Request Processing Flow

OpenAPI Parsing : Converts API specs into executable MCPTool objects with paths, methods, and schemas.

Semantic Matching : Generates embeddings via OpenAI, performs similarity search in ChromaDB, returns ranked matches.

LLM Extraction : Extracts structured parameters from natural language using GPT, validates and requests clarification if needed.

API Execution : Generates and executes cURL commands, formats responses for display.

5.5 Application Interfaces

5.5.1 Main Chat Interface

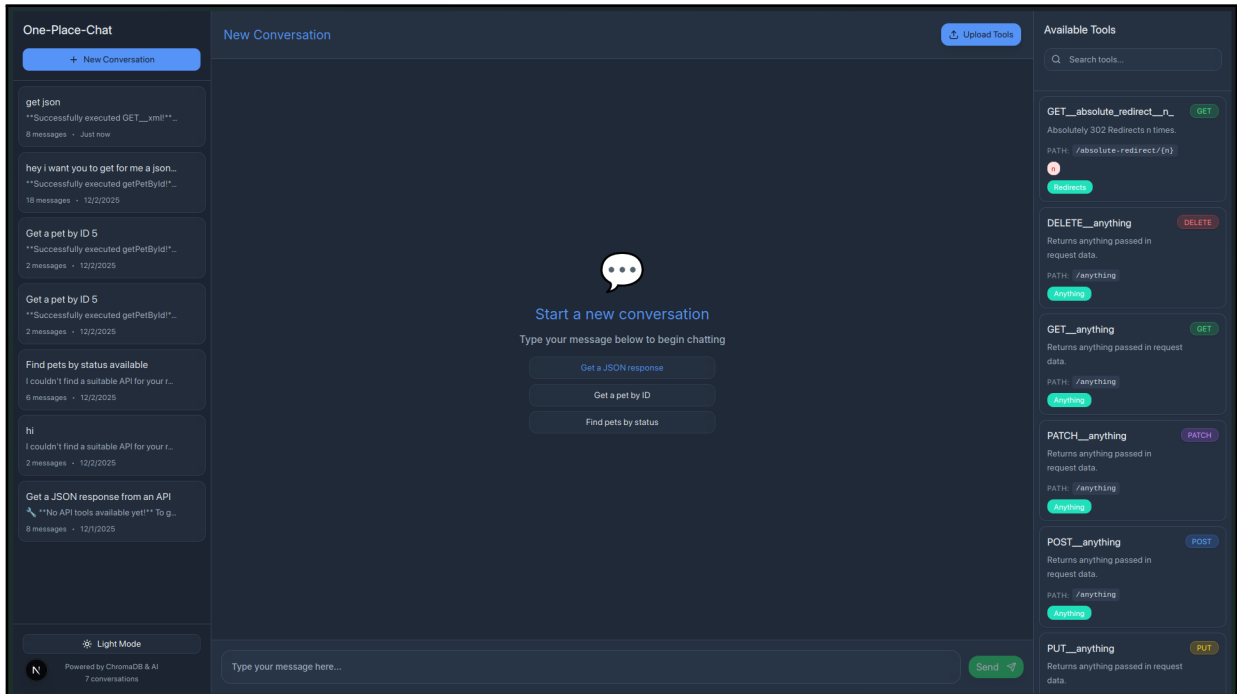


FIGURE 5.3 – One Place Chat Main Interface

Key features of the main interface :

- **Sidebar** : Displays conversation history with timestamps and quick access to previous sessions
- **Chat Area** : Central message display with role-based styling for user and assistant messages
- **Tools Panel** : Shows available API tools loaded from OpenAPI specifications
- **Input Field** : Natural language input for API requests

5.5.2 Chat Message Response

The system displays API responses in a formatted, user-friendly manner :

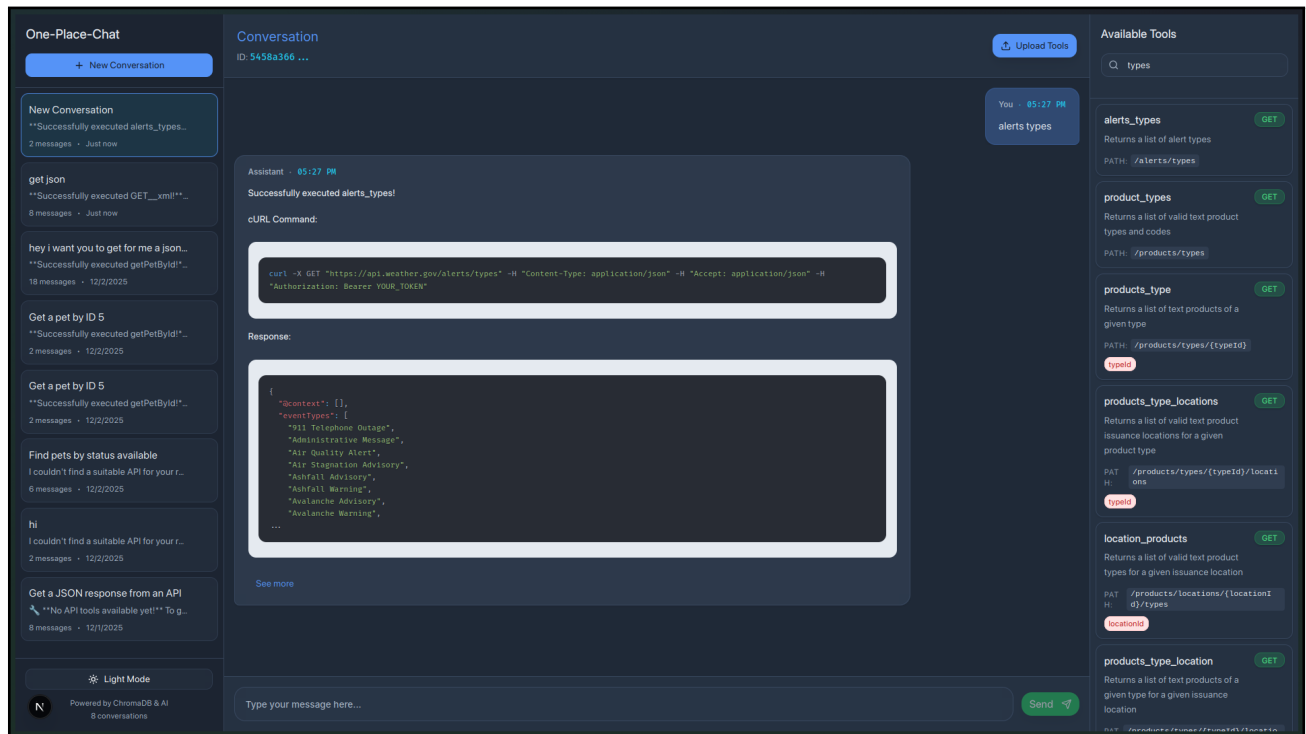


FIGURE 5.4 – Chat Message Response Display

Response features include :

- Formatted JSON output for API responses
- Markdown rendering for rich text formatting
- cURL command display for transparency
- Error handling with helpful messages

5.5.3 Tool Description Panel

When a user selects a tool, the panel displays its complete specification including the tool name, HTTP method badge, endpoint path, and a description of its functionality. The parameters section lists all required and optional fields with their types, helping users understand what information is needed.

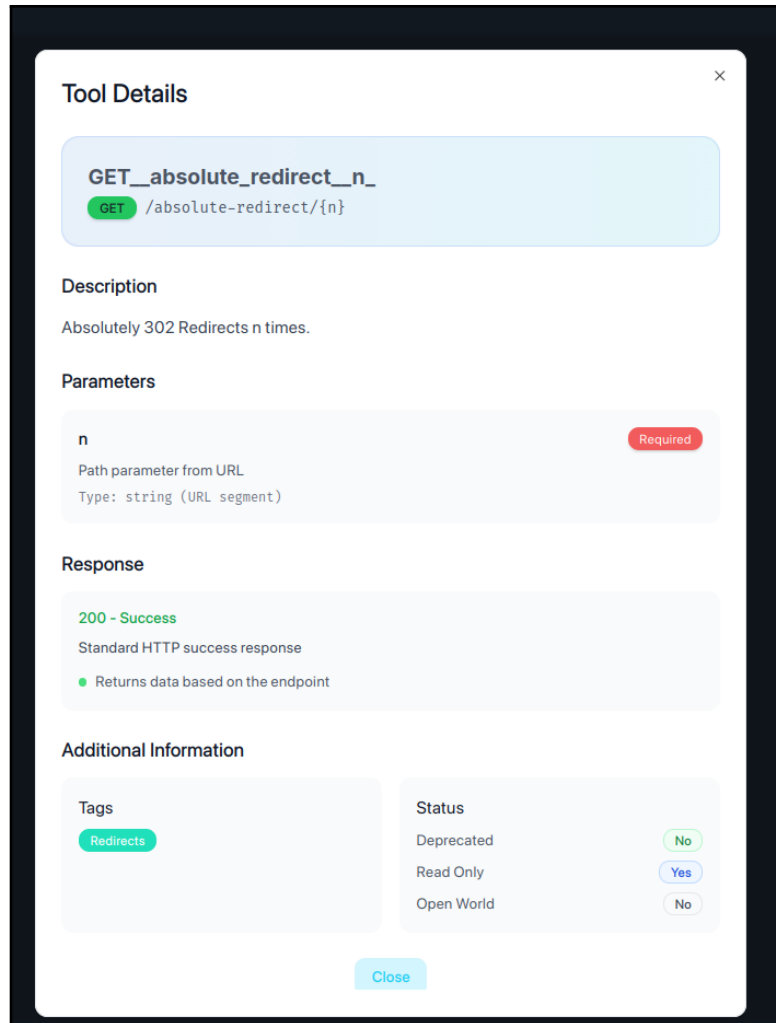


FIGURE 5.5 – Tool Description Panel

The panel content includes :

- **Name** : Tool identifier generated from the API operation
- **Description** : What the tool does and its purpose
- **Endpoint** : Full API path with HTTP method (GET, POST, PUT, DELETE)
- **Arguments** : Required and optional parameters with data types
- **Response** : Expected response format and structure
- **Status** : Tool availability and deprecation warnings

5.5.4 Documentation Upload

The documentation upload interface allows users to add new API tools :

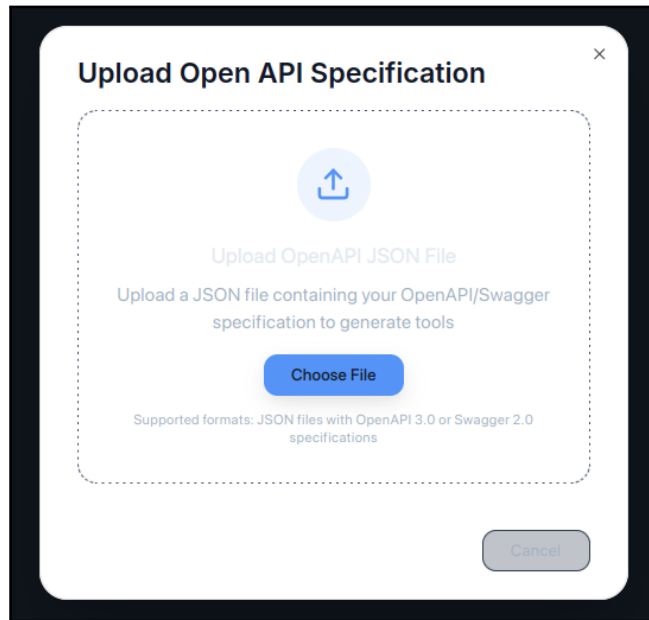


FIGURE 5.6 – OpenAPI Documentation Upload Interface

Upload features include :

- Drag-and-drop file upload for OpenAPI specifications
- Support for JSON and YAML formats
- Automatic tool generation from uploaded specs
- Visual feedback on upload progress and success

5.6 Conclusion

This chapter presented the complete implementation of One Place Chat, from the technology stack to the user interfaces. The frontend, built with Next.js and React, provides an intuitive conversational experience through well-structured components : Sidebar for navigation, ChatInterface for message orchestration, ChatArea for display, and ToolsPanel for API management.

General Conclusion

This internship project at Zenika has been a transformative journey in understanding how artificial intelligence can revolutionize the way humans interact with technology. One Place Chat represents a significant step forward in making API interactions accessible to everyone, regardless of their technical background.

The technical achievements of this project include the development of a three-stage tool management pipeline, the integration of OpenAI embeddings for intelligent tool discovery, and the creation of a user-friendly interface built with modern web technologies. These components work together seamlessly to deliver a chat experience where users can interact with any API simply by describing what they want in natural language.

We extend our sincere gratitude to our supervisors and the entire Zenika team for their guidance, support, and expertise throughout this project. Their mentorship has been instrumental in transforming theoretical concepts into a functional, innovative solution.

In conclusion, One Place Chat demonstrates that AI-powered interfaces can democratize access to complex technical systems. This project marks not only the completion of an internship but also the beginning of a broader vision for intelligent, conversational software that serves users of all technical levels.