Lab 2 – Software Requirements Specification (SRS)

John Hicks

Old Dominion University

CS411W

Dr. Sumaya Sanober

Mar 28, 2025

Version 2

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

This document provides developers with a technically oriented and comprehensive description of CueCode's requirements. It is a continuation of the CueCode product description for CS411W [1], which described the CueCode product from a non-technical perspective. This document will address the requirements for the CueCode prototype, in contrast to the real-world product described by Team Red in CS410.

1.2 Scope

The prototype version of CueCode will be developed to begin development of a product that uses and extends a particular combination of large language model (LLM) and cosine similarity search technology described by Zafin [2]. In addition to building product-oriented features to Zafin's microservice experiment, CueCode further develops Zafin's approach to generating API payloads by including a payload ordering algorithm to ensure data dependencies are not overlooked when the system determines the order in which its generated API requests should be executed by client systems.

The constrained feature set of the CueCode prototype enables timely development of the system's core functionality.

1.3 Definitions, Acronyms, and Abbreviations

API Payload (informal): Information that is sent together with an API request or response. This data, which can be organized in JSON or XML forms, usually includes the details needed by the client to comprehend the answer or by the server to carry out an action.

CueCode Developer Portal: A web-based platform that allows easy API creation with NLP-generated requests and gives developers access to CueCode's tools, API configuration, and integration workflow management.

HTTP Header: Additional metadata, such as the content type, authentication information, or caching instructions, are transmitted with HTTP requests and answers. Headers give context, which improves communication.

HTTP (Hypertext Transfer Protocol): The protocol that specifies the format and transmission of messages between web clients and servers. The type of request is determined by the HTTP methods (GET, POST, etc.).

Hypermedia: inter-linked content on the Internet. In the context of REST APIs, hypermedia allows REST APIs to be more or less RESTful, as defined by Roy Fielding and following authors [3].

OpenAPI specification: a description of an API’s available resources, which follows one of the versions of the OpenAPI specification format [4].

Representational State Transfer (REST): A set of design guidelines for networked apps that use stateless, cacheable, and consistent HTTP processes to facilitate interaction. Through the use of common HTTP techniques, REST allows clients to communicate with servers by modifying resources that match an expected structure [5].

RWP: Real-world product, as contrasted with the prototype.

URL (Uniform Resource Locator): A web address that indicates where a resource is located on the internet. Protocol (such as HTTP/HTTPS), domain, and resource path are all included in URLs. They are necessary to access and consult internet resources.

1.4 References

[1] John Hicks, “CS411W Lab 1: CueCode Product Description,” Feb. 28, 2025.

[2] E. Zafin, “Bridging the Gap: Exploring use of Natural Language to interact with Complex Systems,” Engineering at Zafin. Accessed: Sep. 10, 2024. [Online]. Available: https://medium.com/engineering-zafin/bridging-the-gap-exploring-using-natural-language-to-interact-with-complex-systems-11c1b056cc19

[3] “What Is Hypermedia?,” smartbear.com. Accessed: Nov. 08, 2024. [Online]. Available: https://smartbear.com/learn/api-design/what-is-hypermedia/

[4] “OpenAPI Specification - Version 3.1.0 | Swagger.” Accessed: Sep. 10, 2024. [Online]. Available: https://swagger.io/specification/

[5] “What is a REST API? | IBM.” Accessed: Mar. 09, 2025. [Online]. Available: https://www.ibm.com/think/topics/rest-apis

[6] “Against LLM maximalism · Explosion.” Accessed: Sep. 10, 2024. [Online]. Available: https://explosion.ai/blog/explosion.ai

[7] Y. Su, A. H. Awadallah, M. Khabsa, P. Pantel, M. Gamon, and M. Encarnacion, “Building Natural Language Interfaces to Web APIs,” in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, Singapore Singapore: ACM, Nov. 2017, pp. 177–186. doi: 10.1145/3132847.3133009.

[8] “Dramatiq: background tasks — Dramatiq 1.17.1 documentation.” Accessed: Mar. 09, 2025. [Online]. Available: https://dramatiq.io/

[9] “Evaluation | Genkit,” Firebase. Accessed: Sep. 14, 2024. [Online]. Available: https://firebase.google.com/docs/genkit/evaluation

[10] B. Lorica, “Expanding AI Horizons: The Rise of Function Calling in LLMs,” Gradient Flow. Accessed: Oct. 24, 2024. [Online]. Available: https://gradientflow.com/expanding-ai-horizons-the-rise-of-function-calling-in-llms/

[11] “Function Calling.” Accessed: Sep. 14, 2024. [Online]. Available: https://platform.openai.com/docs/guides/function-calling

[12] *microsoft/prompt-engine*. Microsoft, 2024. Accessed: Oct. 24, 2024. [Online]. Available: https://github.com/microsoft/prompt-engine

[13] Mark Needham, *Returning consistent/valid JSON with OpenAI/GPT*, (Jul. 26, 2023). Accessed: Oct. 24, 2024. [Online]. Available: https://www.youtube.com/watch?v=lJJkBaO15Po

[14] “Tool/function calling | LangChain.” Accessed: Sep. 14, 2024. [Online]. Available: https://python.langchain.com/v0.1/docs/modules/model_io/chat/function_calling/

[15] R. Merritt, “What Is Retrieval-Augmented Generation aka RAG?,” NVIDIA Blog. Accessed: Sep. 25, 2024. [Online]. Available: https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/

[16] “spaCy · Industrial-strength Natural Language Processing in Python.” Accessed: Sep. 26, 2024. [Online]. Available: https://spacy.io/

[17] *openapi-spec-validator: OpenAPI 2.0 (aka Swagger) and OpenAPI 3 spec validator*. Accessed: Mar. 09, 2025. [OS Independent]. Available: https://github.com/python-openapi/openapi-spec-validator

1.5 Overview

This document covers first an overall description of the CueCode product and system, then it specifies specific requirements, separated by their relevance to the Runtime and Configuration algorithms described in section 2.1 below. Following section 2.1 is discussion of the product functions, which describe the prototype's functionality contrasted with the real-world product's. Section 2.3, "User Characteristics", describes CueCode's users and their roles in the system. The CueCode prototype's constraints are described in Section 2.4. Section 2.5 outlines the prototype project's assumptions and dependencies.

2 Overall description

2.1 Product Perspective

CueCode will offer a complete framework for application developers to integrate with a service for translating natural language to REST API payloads while giving calling code the opportunity to act on the data before issuing API requests.

CueCode will translate natural language in text format into a series of correctly ordered REST API payloads, which a client application can then issue to the target API after further processing and/or human review of the API call suggestions.

The customer's calling code can process payloads using further business rules or provide users the opportunity to review the payloads before they are sent to the target API. This approach reduces the risk of generating API issuing API requests whose data is generated (inferred) by a system, a requirement becoming more common in as chatbots and other Large Language Model (LLM) tools make natural language interfaces with software more commonly requested [2], [6], [7]. Both human-review and further processing use-cases are illustrated in Figure 1 below.

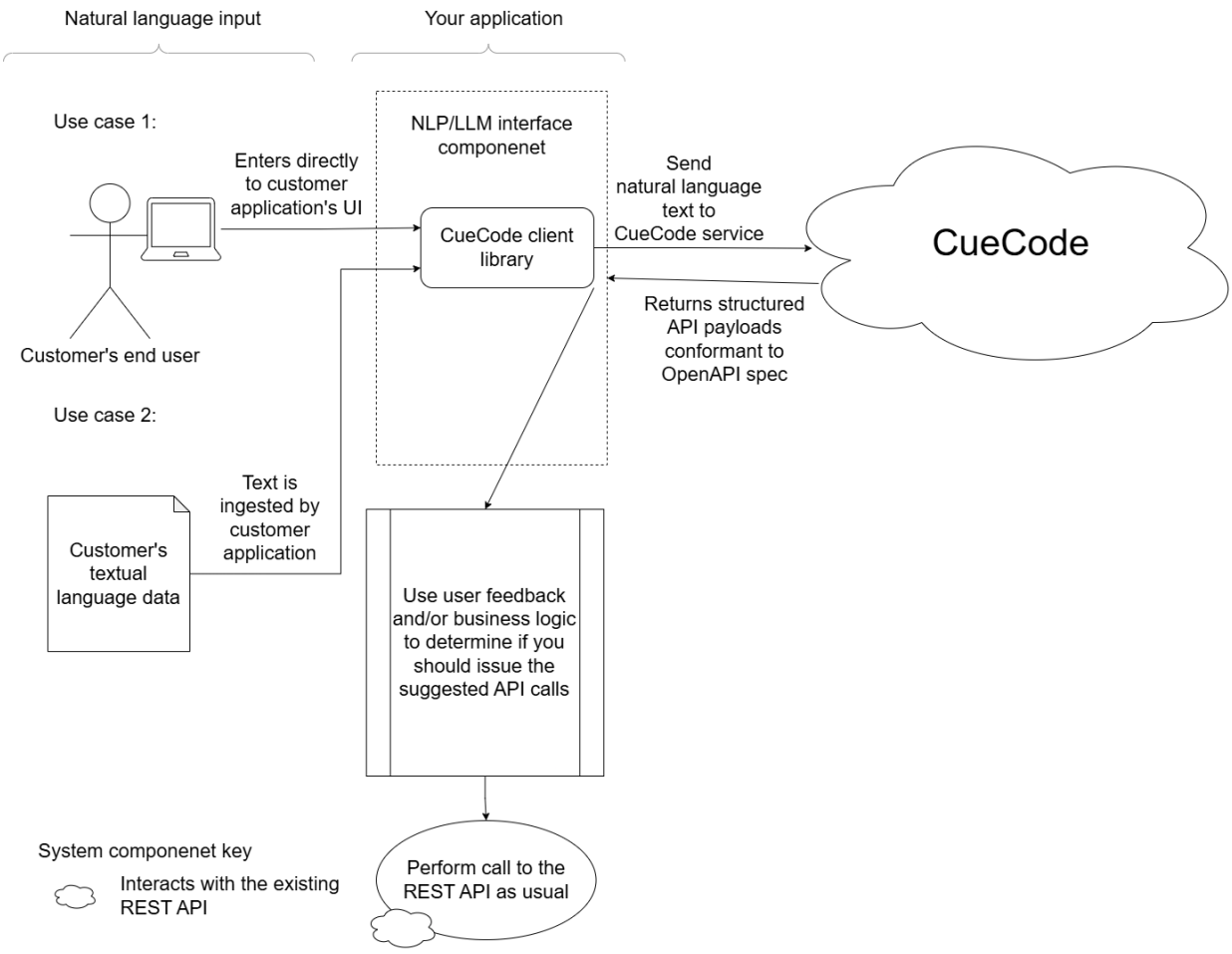

Figure 1 Two use-cases showing CueCode's API payload generation concept.

The CueCode Developer Portal is a concept that features prominently in the real-world product (RWP). For the prototype, the Developer Portal exists only to receive an OpenAPI specification and start the CueCode Configuration algorithm, which prepares CueCode to generate API payloads at runtime for a target API described in the OpenAPI specification. Figure 2 below shows the major components of the Developer Portal and how they relate to the Configuration algorithm.

[This space intentionally left blank.]

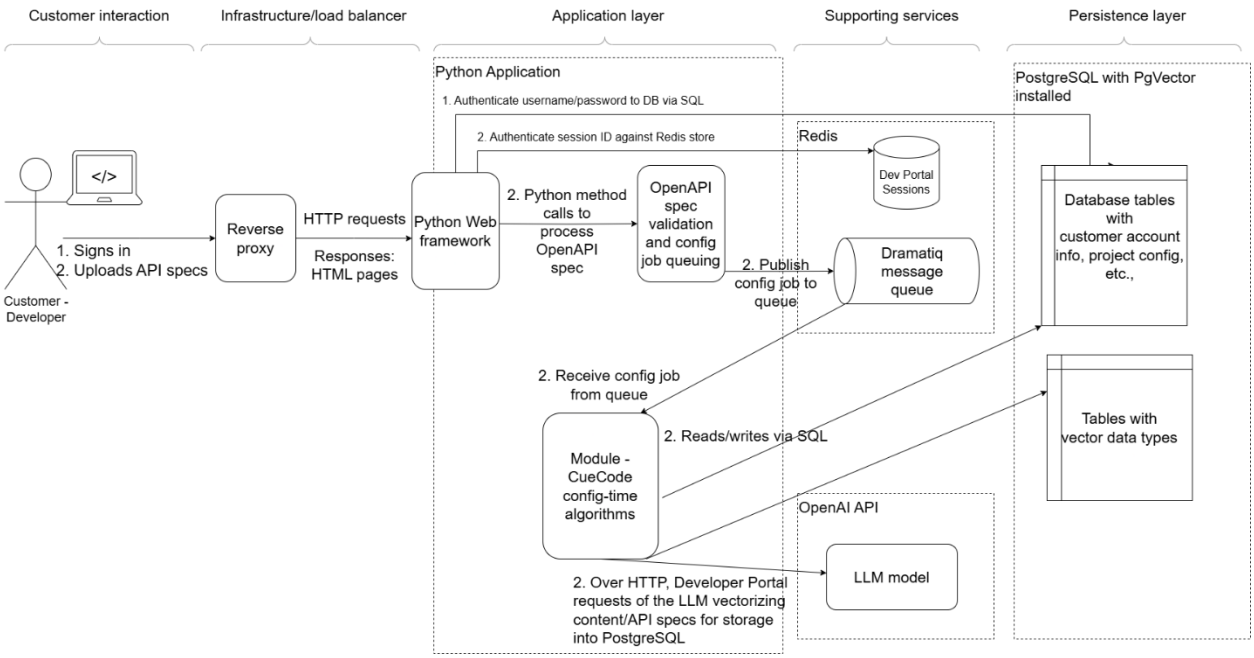

Figure 2 The major components of the CueCode Developer Portal and Configuration Algorithm, showing component interactions for authentication (1) and the Configuration Algorithm (2)

The CueCode application is a Python application that is divided into five modules:

1. Common
2. Web framework
3. Configuration algorithm
4. Runtime algorithm
5. Worker process integration code, called Actors [8]

The Python application is stateless - storing all state in either Redis or Postgres - so multiple instances of the CueCode server can be run at one time and in one of two modes.

CueCode server will run in Web server mode and worker mode, depending on how it is invoked. In Figure 2 above, the “Module – CueCode config-time algorithms” would operate in worker mode. Worker mode is enabled by use of the Dramatiq framework [8]. The runtime algorithm operates in Web server mode.

Developers can integrate their applications with CueCode by using the CueCode client library for their programming language of choice.

At runtime, the developer's application can make a request to the CueCode service (itself running over an HTTP Web API). The CueCode service will reply with the generated REST API payload(s) corresponding to the natural language text input given. CueCode will provide client libraries to integrate with the runtime HTTP API and ensure API requests are sent in the correct order to ensure the data dependencies internal to the generated response are met. In the RWP, the CueCode system would also be able to account for data dependencies within the target system, not just within its response.

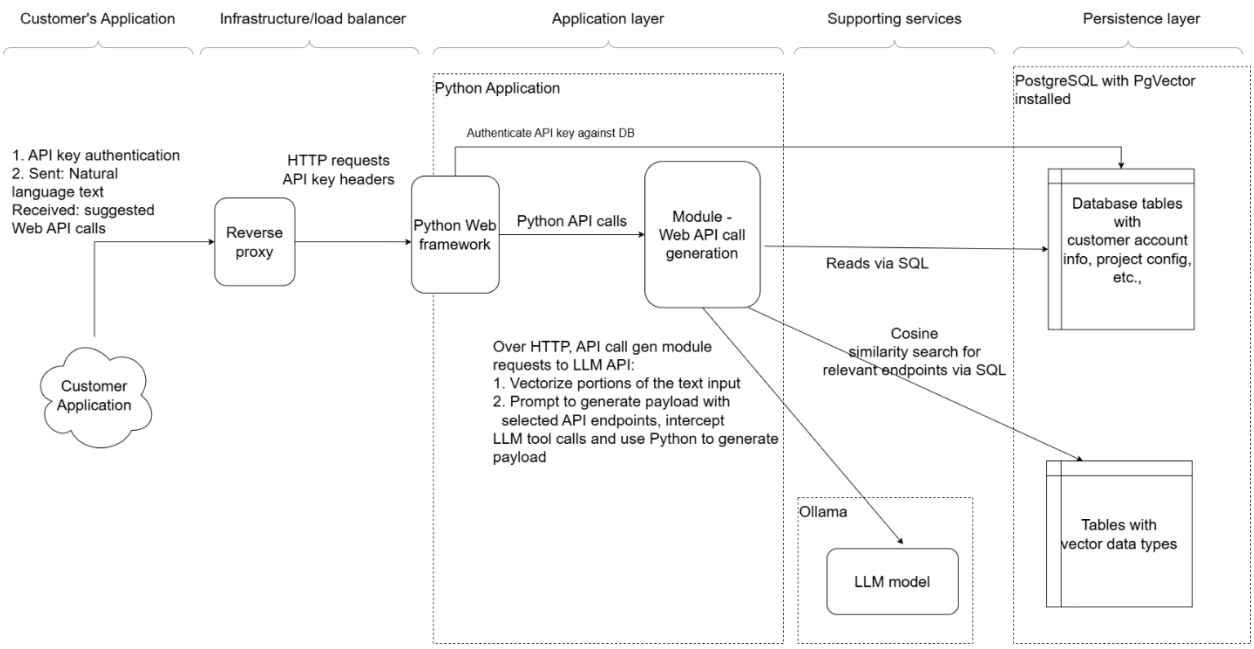

Figure 3 CueCode runtime API authentication (1) and API payload generation (2)

With both Configuration and Runtime major components included, the CueCode system can be represented as the major functional components diagram in Figure 4 below.

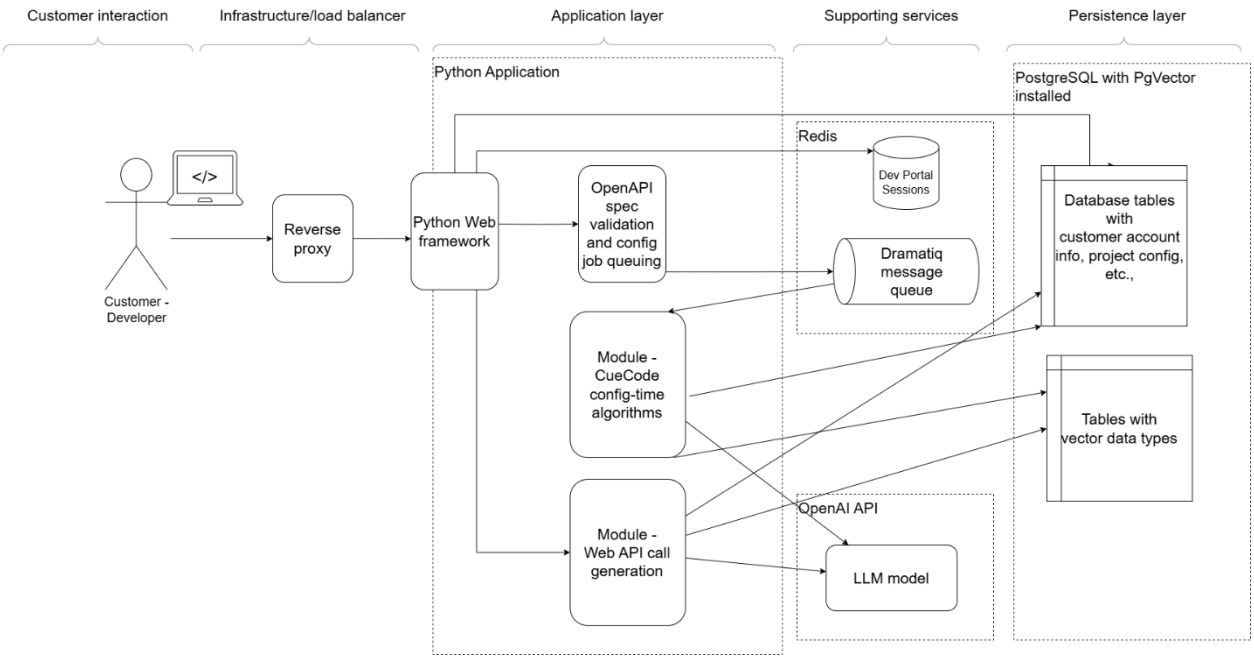

Figure 4 All major functional components of the CueCode system

CueCode has first-class support for humans and business rules in the loop of payload generation [6], as compared with other approaches to the problem [2], [6], [7], [9], [10], [11], [12], [13], [14], [15]. This will allow developers to begin using AI in a risk-aware manner in comparison with competing approaches to this problem, since calling code can process payloads using further business rules or provide users the opportunity to review the payloads before they are sent to the target API.

[This space intentionally left blank.]

2.2 Product Functions

The CueCode prototype will include the following features, compared to the real-world product (RWP).

CueCode's prototype will translate natural language text into Web API requests against a configured OpenAPI specification. CueCode's prototype will not consult the API against which the OpenAPI specification is defined, which would provide an enormous benefit in identifying data objects by name within the natural language. This feature is simply too complex to implement in the time allotted for prototyping. As indicated in table *Table 1* below, the "Map natural language to customer's data entities via search or API call" feature would be available in the RWP.

Category	Feature	Real-world product	Prototype
Portal	Interactive Login / Authentication	✓	✓
	Account sign-up / deletion	✓	
Config	REST API definition management	✓	✓
	Upload OpenAPI specifications		
	REST API configuration wizard	Web-based Wizard and/or yaml file	Only yaml file

	Allow only a subset of valid OpenAPI specs be used	✓	✓
Runtime	Process natural language and turn it into REST API payloads	✓	✓
	Map natural language to customer's data entities via search or API call	✓	
Client libraries	Integrate application with CueCode's NLP API	✓	✓
NLP monitoring	Trace, debug, and report on translation requests in CueCode Web UI	✓	
Marketplace	Share CueCode API configurations with other users	✓	

Table 1 Real-world product (RWP) vs. Prototype features

2.3 User Characteristics

CueCode's first customers are developers, but CueCode will power user experiences for CueCode's the customer's end users, requiring as much accuracy and as close to "natural" ability to understand human intent within language as possible.

CueCode developer users will upload OpenAPI specifications to the system in preparation for runtime API payload generation. CueCode developers also integrate their existing systems with CueCode using the CueCode client code libraries, requiring a streamlined user interface for both API upload via Web browser and a helpful software interface with the CueCode client library.

The prototype will also support the end users of each application targeted by CueCode for payload generation. Operation of the CueCode server will be transparent to end users, and it will not allow them any access to changing the OpenAPI specification.

2.4 Constraints

The CueCode Python server may not use GPU hardware for the prototype design, due to ODU infrastructure constraints for a project of this scope.

The CueCode server prototype may not contact the target API server for data at any time, as it might in the RWP. For example, in the RWP, the CueCode server might need to map a person's name to their customer ID when the person has been referenced in the natural language text.

2.5 Assumptions and Dependencies

The prototype design will not cover several basic scenarios that natural-language-to-API content conversion would require. Already noted is that the CueCode server will not be able to make the mapping from customer name to a customer ID, a scenario very likely needed to convert a natural language description of a customer to a REST API request.

Other dependencies are listed below:

- Languages & Frameworks
    - HTML, CSS, JavaScript – Frontend & UI/UX
    - Python – Backend logic & integrations
    - Bootstrap 5 – Responsive design
    - Flask & Jinja – Backend & templating engine
- Platform & IDE
    - Docker – Consistent environments
    - Visual Studio Code (VS Code) – Preferred IDE
    - Linux & Windows (WSL, Docker) – Dev environment
- API & Components

- OpenAI Client
  - LLM content integration and tool call requests
- Swagger CodeGen – API client library generation
- Spacy.io – NLP processing [16]
- OpenAPI Spec Validator [17] – API spec validation