CueCode: Humanize APIs, without the headache

CS 410W

Lab 1 draft

John P. Hicks

08 November 2024

Table of Contents

Listing of Figures

1 Introduction

In CS410 and CS 411W, students will produce a software-based solution to a real-world problem. The problem Team RED seeks to solve is that of creating a framework for turning natural language into REST (Representational State Transfer) API (Application programming interface) calls, using a Large Language Model (LLM).

The software solution is called CueCode.

Recent advances in LLM technology allow more generalized content generation based on foundation models. As such, LLM focused tooling has exploded onto the market (Uspenskyi, 2024). Despite these advances, the application of LLMs to the problem of turning natural language into REST API payloads has not seen a mature implementation yet.

Because of the lack of standardized frameworks for this use case, software developers are forced to learn LLM technology and build one-off solutions to generating API payloads from natural language. This requires extra cost in time and staff. Furthermore, progress on making applications use AI features is stopped by the risks involved in trusting LLMs' decision-making capabilities (Nexus, 2024; Tyen, 2024); any good solution to the REST API payload generation problem must include the opportunity for humans or business rules to validate the payload before it is sent.

2 CueCode Product Description

CueCode will offer a complete framework for application developers to integrate with a service for intelligently translating natural language to REST API payloads. CueCode has first-class support for humans and business rules in the loop of payload generation, as compared with other approaches to the problem. This will allow developers to begin using AI in a risk-aware

manner while the technology is still improving, giving them a head start on preparing their applications and service offerings for the years to come.

2.1 What does it do?

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.

CueCode will be a Web application that, with its supporting off-the-shelf services, offers a Developer Portal experience for configuring the application and another service for performing natural language to REST API translation suggestions. Developers can integrate their applications with CueCode by using the CueCode client library for their programming language of choice.

2.2 Significance, Uniqueness and Innovation

Getting an LLM to produce Web API payloads and validating them is a specialized task requiring skills that many Web and fullstack developers do not possess (Uspenskyi, 2024).

Since these developers are those most often building business applications, a solution for turning natural language into REST API payloads should take into consideration how easy it will be for developers with other skillsets to use the tool. CueCode gives a good foundation for Web and fullstack developers to turn natural language into REST API payloads in their applications.

2.3 What It Accomplishes

CueCode will be a full framework and service to turn natural language into Web API payloads; nothing like it exists for arbitrary REST APIs. CueCode will work with any REST API defined with an OpenAPI specification (*OpenAPI Specification - Version 3.1.0 | Swagger*, n.d.).

2.4 Problem Solution

Developers will upload their OpenAPI specification to CueCode's Developer portal, then answer questions about the API definition and structure as needed.

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 make integration with the CueCode service seamless.

2.5 Major Components (Hardware/Software)

The CueCode solution will consist of a backend Python application, Ollama service, PostgreSQL (Postgres) database with the pgvector extension, and a third-party identity provider. The Python application will require use of the SpaCy library, which allows natural language structuring and named entity recognition (*SpaCy · Industrial-Strength Natural Language Processing in Python*, n.d.).

Ollama is an application that allows communication with LLMs over a standardized HTTP API.

Both Ollama and SpaCy can be optimized when running on GPU hardware, so the CueCode project team has arranged with the ODU Computer Science Systems Group (CS Systems Group) to reserve GPU compute resources during the Spring 2025 semester, for our implementation phase of the project (CS Systems group, personal communication Oct 2024).

CueCode will require hardware capable of running the following systems separately:

- Ollama 3.1 (minimum) running the 70 billion parameter model (minimum)
    - The CS Systems group already runs Llama models (personal communication).

- A Python application using SpaCy, running on either CPU or GPU (*Install SpaCy · SpaCy Usage Documentation*, n.d.).

3 Identification of Case Study

The two fictional user personas describe the people who would benefit from CueCode's development:

- Steve, a fullstack developer, needs to integrate text-to-API-payload features but finds he needs to roll his own solution and understand NLP and LLM technology.
- Case study of Patricia, who needs to make an appointment at a hospital whose booking system already uses REST APIs.

Two general use-cases are supported by CueCode (Figure 1 below):

1. Human review of suggested API calls
2. Batch processing of textual data, issuing API calls without human review.

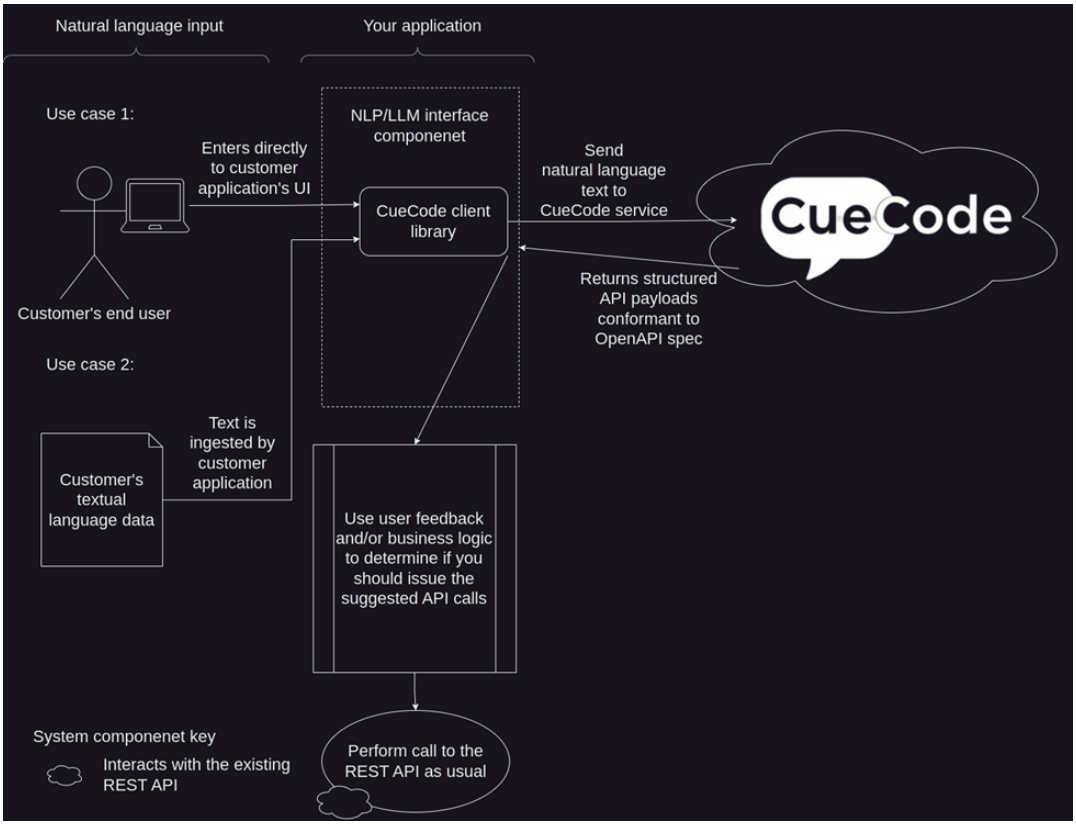

Figure 1 Two use-cases showing CueCode's API payload generation concept.

4 CueCode Product Prototype Description

The CueCode prototype will limit the scope of features to those required to turn natural language to REST API calls. Other potentially beneficial features will be excluded to focus on delivering a working prototype of the core CueCode functionality.

4.1 Prototype Architecture (Hardware/Software)

The prototype will focus on REST APIs defined with OpenAPI specifications, using JSON content types only.

The application will use 12-factor application development practices, to allow for containerization and other modern application development and deployment practices (Adam

Wiggins, 2017). To make deployment and development easier during prototyping, the Python backend will group all functionality in one Python application, separated by modules.

The Ollama, database, and authentication services will run as separate applications within the CueCode system.

Customers will connect their applications to CueCode via the client libraries supplied by CueCode, using authentication credentials they obtain from the Developer Portal Web application that developers can use to configure CueCode for use with their OpenAPI-defined REST APIs.

4.2 Prototype Features and Capabilities

CueCode's feature set will limit the kinds of API endpoints against which CueCode can be configured. For example, GraphQL is another API format that CueCode would eventually support if it were a real product.

Further research may also require hypermedia be included in the target API's responses or that the OpenAPI specifications used have certain properties to define relationships between target API's entities, to ease implementation for CS411.

Depending on the team's work capacity, the prototype might or might not include non-core features that could aid in the hypothetical commercialization of CueCode, such as an idea to create a marketplace where commonly used Web APIs (e.g., Google Drive) can have their CueCode configuration shared among CueCode users, making a Web 2.0 content sharing dynamic (Sean Baker, personal communication Oct 2024).

4.3 Prototype Development Challenges

The prototype's algorithm development is highly dependent on the quality of results from the LLM.

Because LLMs are non-deterministic (Nexus, 2024; Uspenskyi, 2024), it may prove difficult to instruct an LLM for a task so specific as selecting an API endpoint. Thurs, we face an open question of whether CueCode should allow the LLM choose which REST API endpoint to generate data, or for CueCode to implement a cosine similarity search algorithm, similar to work done by Zafin's engineers and others (Mark Needham, 2023; Zafin, 2023).

However the LLM is prompted, it will return a text response that needs to be validated to confirm the API payload conforms to the OpenAPI specification. A combination of prompt template and LLM Function calling can be used to ensure that the generated API payload is compliant with the OpenAPI specification provided enforcement (Mark Needham, 2023; *Microsoft/Prompt-Engine*, 2022/2024; *Stanfordnlp/Dspy*, 2023/2024).

Determining the relationships between entities will challenging. Doing so involves not just parsing natural language into a structured grammatical parse tree using Spacy (*Linguistic Features · SpaCy Usage Documentation*, n.d.), but it also requires mapping that structure to the API's entity relationship structure, as best CueCode can determine the API's structure from the API spec.

Another challenge will be developing a sorting algorithm for ensuring that the order in which API calls are made doesn't invalidate them because of unmet data dependencies. This algorithm will require using placeholders for data not knowable until after an API call is made. For example, this would happen when creating an entity, A, and several other entities related to it. The algorithm would need to ensure that the API request to create A would be issued prior to any requests creating entities that are related to A.

5 Conclusion

As seen by a few of our most challenging development questions, CueCode will remove much complexity from the fullstack developer's code, enabling natural language interaction with

REST APIs in a reusable, operationalized framework. CueCode will help developers leverage the generation capabilities of LLMs, while also controlling the risks thereof. This allows developers to humanize APIs, without the headache.

6 Glossary

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 (*What Is Hypermedia?*, n.d.).

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.

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 in order to access and consult internet resources.

7 References

Adam Wiggins. (2017). *The Twelve-Factor App*. https://12factor.net/

Install spaCy · spaCy Usage Documentation. (n.d.). Install SpaCy. Retrieved November 8, 2024, from https://spacy.io/usage

Linguistic Features · spaCy Usage Documentation. (n.d.). Linguistic Features. Retrieved November 8, 2024, from https://spacy.io/usage/linguistic-features

Mark Needham. (2023, July 26). *Returning consistent/valid JSON with OpenAI/GPT*. https://www.youtube.com/watch?v=lJJkBaO15Po

Microsoft/prompt-engine. (2024). [TypeScript]. Microsoft. https://github.com/microsoft/prompt-engine (2022)

Nexus, P. (2024, July 16). *Large language models make human-like reasoning mistakes, researchers find*. Tech Xplore. https://techxplore.com/news/2024-07-large-language-human.html

OpenAPI Specification—Version 3.1.0 | Swagger. (n.d.). Retrieved September 10, 2024, from https://swagger.io/specification/

SpaCy · Industrial-strength Natural Language Processing in Python. (n.d.). Retrieved September 26, 2024, from https://spacy.io/

Stanfordnlp/dspy. (2024). [Python]. Stanford NLP. https://github.com/stanfordnlp/dspy (2023)

Tyen, G. (2024, January 11). *Can large language models identify and correct their mistakes?* Google Research. http://research.google/blog/can-large-language-models-identify-and-correct-their-mistakes/

Uspenskyi, S. (2024, September 19). *Large Language Model Statistics And Numbers (2024)—Springs*. https://springsapps.com/knowledge/large-language-model-statistics-and-

numbers-2024, https://springsapps.ai/blog/large-language-model-statistics-and-numbers-2024

What Is Hypermedia? (n.d.). Smartbear.Com. Retrieved November 8, 2024, from https://smartbear.com/learn/api-design/what-is-hypermedia/

Zafin, E. (2023, August 15). Bridging the Gap: Exploring use of Natural Language to interact with Complex Systems. *Engineering at Zafin*. https://medium.com/engineering-zafin/bridging-the-gap-exploring-using-natural-language-to-interact-with-complex-systems-11c1b056cc19