"Humanize Web APIs, without the headache"

Team RED
Old Dominion University
CS 410
Fall 2024

CueCode

Table of Contents

I. Table of Contents ..... Slide 2 (Everyone)
II. Team Bios ..... Slide 4 (John)
III. Feasibility ..... Slide 7 (John)
- A. Why talk with Web APIs? - The Societal Problem ..... Slide 8
- B. Why CueCode? - Problem Statement ..... Slide 9
- C. Elevator Pitch ..... Slide 10 (John)
- D. Quick example ..... Slide 11 (John
- E. Problem Characteristics - NLP/LLM challenges ..... Slide 12 (John)
- F. Problem Characteristics ..... Slide 13 (Kobe)
- G. Problem Characteristics - NLP/LLM challenges ..... Slide 14
- H. Current Process Flow ..... Slide 15 (Chase/John)
- I. Solution ..... Slide 24 (Chase)
- J. Solution Statement ..... Slide 25 (Chase)
- K. Solution Characteristics ..... Slide 26 (Chase)
- L. Solution Process Flow ..... Slide 27 (Andrew)
- M. What It Will Do ..... Slide 29 (Freddie)
- N. What It Will Not Do ..... Slide 30 (Freddie)
- O. Competition Matrix ..... Slide 31 (Kobe)

IV. Design ..... Slide 37
- A. Who are we building for? User roles ..... Slide 39 (Freddie)
- B. Feature table - Developers and publishers ..... Slide 40 (Freddie)
- C. Real-world vs. Prototype feature ..... Slide 41 (Freddie)

V. User interface ..... Slide 42 (Diya/Sean)
- A. User interface - outline ..... Slide 43
- B. Input and Payload Management ..... Slide 44
- C. Client Library Interaction ..... Slide 45
- D. Major Functional Components ..... Slide 46
- E. Major Functional Components Diagram - Configuration ..... Slide 47 - 50 (Chase)

Table of Contents - Cont.

I. Summary of key Algorithms ..... Slide 51 (John)
- A. Configuration algorithm summary ..... Slide 52
- B. Configuration algorithm details and diagram ..... Slide 53 - 56
- C. Runtime algorithm summary ..... Slide 57 (John)
- D. Split sentences, identify nouns and verbs with SpaCy ..... Slide 58
- E. Perform embeddings, query for relevant endpoints, prompt LLM ..... Slide 59
- F. Sort payloads according to data dependencies ..... Slide 60

II. Data design and management ..... Slide 61 (Kobe)
- A. Database design ..... Slide 62
- B. Entity Relation Diagram (ERD) ..... Slide 63
- C. Database management ..... Slide 64
- D. Software and hardware tools ..... Slide 65
- E. Software / hardware tools ..... Slide 66 (Freddie/Andrew)
- F. Development Tools ..... Slide 67 (Diya)

III. Work breakdown structure ..... Slide 70 (Sean)
- A. Work breakdown structure overview ..... Slide 71 - 76

IV. Sprint breakdown ..... Slide 77 (John)
- A. Sprints ..... Slide 78 - 83

V. Risk management ..... Slide 84
- A. Risks - Customer, Operational, Regulatory ..... Slide 86 (Fred,, John, Andrew, Diya)
- B. Risks - Technical ..... Slide 89 - 94
- C. Risks - Mitigation landscape ..... Slide 95
- D. Conclusion ..... Slide 96 (John)
- E. Questions and discussion ..... Slide 97 (John)
- F. Appendix A ..... Slide 98
- G. Algorithms ..... Slide 102 (John)

VI. User Stories ..... Slide 114
- A. User Stories: Initiative core functionality ..... Slide 115
- B. User Stories: As A Developer ..... Slide 116

VII. References ..... Side 117

CueCode

Team Bios - 1/3

John Hicks

A part-time Computer Science Major at ODU, transfer student from Tidewater Community College (TCC) where he earned his Associate of Science with a specialization in Computer Science. John has been employed full-time in software development and IT roles during most of his time in school. John began his journey into software development when his parents' small business needed a website upgrade from Microsoft Frontpage to WordPress. On understanding WordPress's hook and filter mechanism, John's imagination was kindled in wondering what other ways of writing software there might be. That curiosity turned to flame and was formed into skill with the help of many friends, family, internet contributors, workplace mentors and school faculty.

Freddie Boateng

A Computer Science major with a minor in Cybersecurity. He is from Northern Virginia and currently working as a Cybersecurity Engineer with Zachary Piper Solutions. He strives to always improve and stay updated to the world of technology, enabling him to reach his goals

CueCode

Team Bios - 2/3

Kobe Franssen

Full time Computer Science major at ODU while also working part time as System Administrator at the ODU Computer Science Consultant Group. Experienced in Java, Python, C++ and API handling such as with Discord Bots. Love working on cars and has 3 cats.

Diya Patel

A Junior at ODU, pursuing a Bachelor's degree in Computer Science. She is interested in learning about the newest advancements in web development and artificial intelligence. She has an ongoing desire to take on new tasks and expand her skill set.

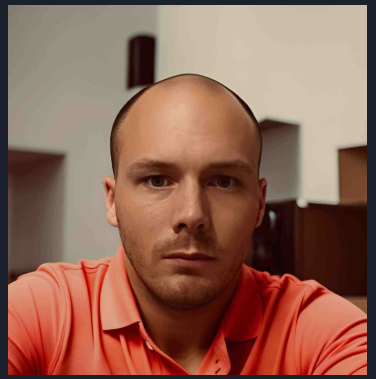
Andrew Bausas

I am a computer science major from Virginia Beach. I am to improve my skills and eventually use them to make games.

Team Bios - 3/3

Sean Baker

Sean's journey into computer science has been unconventional and spans both time and institutions. A transfer student from Piedmont Virginia Community College (PVCC), Sean earned his associate degree in computer science in 2016, but his tech journey began much earlier. At 14, he built his first WordPress site to supplement hsi allowance, which led to articles like "Ten reasons this IPhone will succeed". Since then, rather than pursuing a conventional corporate path, Sean has prioritized creativity and innovation, which has led him to work on projects that push technological boundaries, including contributing to self-driving car technologies with Edison2 and developing die cast automation software for VisiTrak Worldwide and Rockwell Automation. His self-taught, autodidactic learning approach has defined his career. Set to graduate this spring, Sean hopes to pursue a masters degree.



Chase Wallace

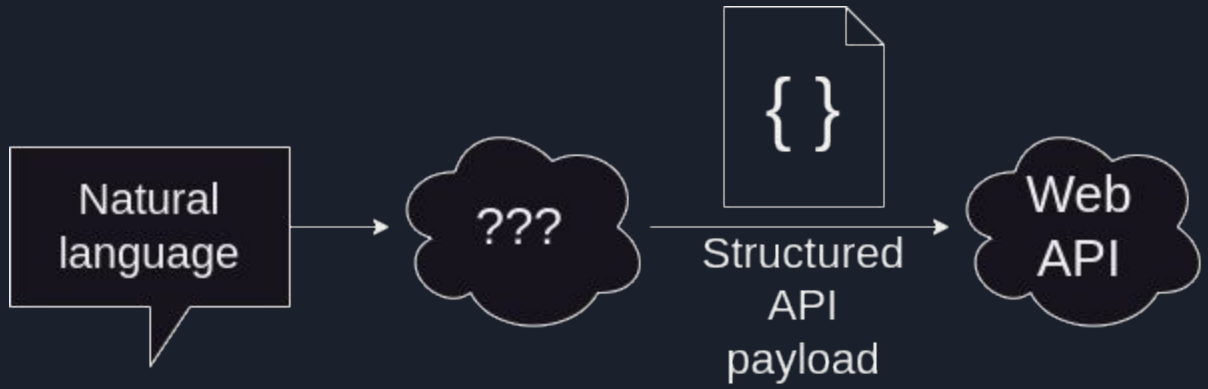
A Computer Science and Biomedical Sciences double major from Norfolk with a strong interest in neuroscience and artificial intelligence. He is always ready to learn new skills and broaden his horizons with challenging new projects.

CueCode

Feasibility

Why talk with Web APIs? - The Societal Problem

- User interfaces don't speak the user's language, but users rely on apps to make things happen.
- Developers are motivated to add Natural Language Processing (NLP) features to their apps, but doing so is painstaking.
- Things happen in Web apps through Web Application Programming Interfaces (APIs).
- It is complex to turn natural language to structured data, which is what Web APIs must receive in order to work (Su et al., 2017).
- Open source contributors and researchers are attempting to use new Large Language Models (LLMs) to create Web API payloads (Zafin, 2023; Tool/Function Calling | LangChain, n.d), but there is not mature tooling in this emerging part of the market.



CueCode

Why CueCode? - Problem Statement

- No LLM tools available for Web API payload generation focus on putting a human or deterministic business logic “in the loop” of payload generation.
- Enterprises and Software-as-a-Service (Saas) applications cannot afford to make every function call an LLM recommends for data entry or triggering actions; that is too high risk.
- The inability for current tools to manage the risks of LLM usage gates Enterprises and SaaS providers from developing AI applications while LLM technology matures.

=> **There’s demand for API call payload generation, but no simple, operationalized, and risk-aware tooling for it.**

CueCode will use this opportunity to commoditize the process of turning natural language into API payloads against existing existing Web APIs.

Elevator Pitch

- CueCode lets a Web application generate REST API calls from natural language with minutes of development time.
- A good OpenAPI specification and a few key questions are all CueCode needs to start generating REST API requests.
- CueCode can add AI features to your app without any backend code changes or specialized NLP or large language model (LLM) skills.
- This allows rapid development of natural language processing features, without having to risk taking humans or business rules out of the loop.

Quick example

- Example input: “I would like to book a hospital appointment on the 20th of October at 2pm for a medical checkup.”
- Output: a full REST API payload for creation of an appointment on the 20th of October, in the structured data format the REST API expects:
  - `POST https://the-appointment-app.com/api/v1/appointments/`
  - `{"client":{"last_name": "Davis", "first_name":"Patricia"}, date: “2024-10-20”, time: “1400” }`
- This output can be shown to the user, put through business logic, or sent immediately to the appointments REST API.
- The developer now has a choice about how to handle the suggested API payload.

CueCode

Problem Characteristics - NLP/LLM challenges

Problems with current NLP/LLM processing for creating API calls:

- Require awareness of prompt engineering and other more complex AI techniques
  - => Time/money upskilling fullstack and Web developers.
- The NLP tools for generating API calls today are stand-alone programs and libraries that don't present a unified, opinionated solution.
  - => Developers are left building one-off solutions.
  - => Heavy boilerplate/in-house frameworks.
- Humans and application logic are kept out of the loop in other approaches; this is high-risk.

Problem Characteristics - use of API specs

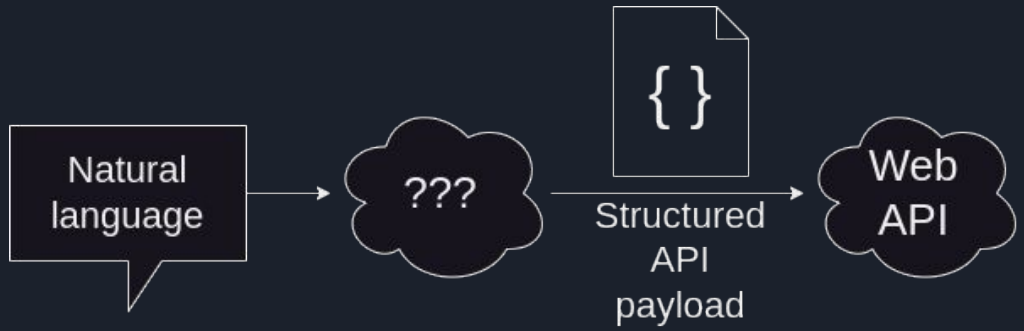
- Since APIs are commonly described with specifications, why not use those?
  - (Keep a clean contract between system components.)
- OpenAPI is the leading industry standard way to describe REST (REpresentational State Transfer) APIs.
- However, there are no complete frameworks that leverage OpenAPI specifications when turning natural language to REST API calls.

CueCode

Problem Characteristics - NLP/LLM challenges

Problems with current NLP/LLM processing for creating API calls:

- Limiting Responses to fit an API Structure Is Difficult
- Lack of Understanding of Entity Relationships
- Absence of a Consistent Framework for Web Developers

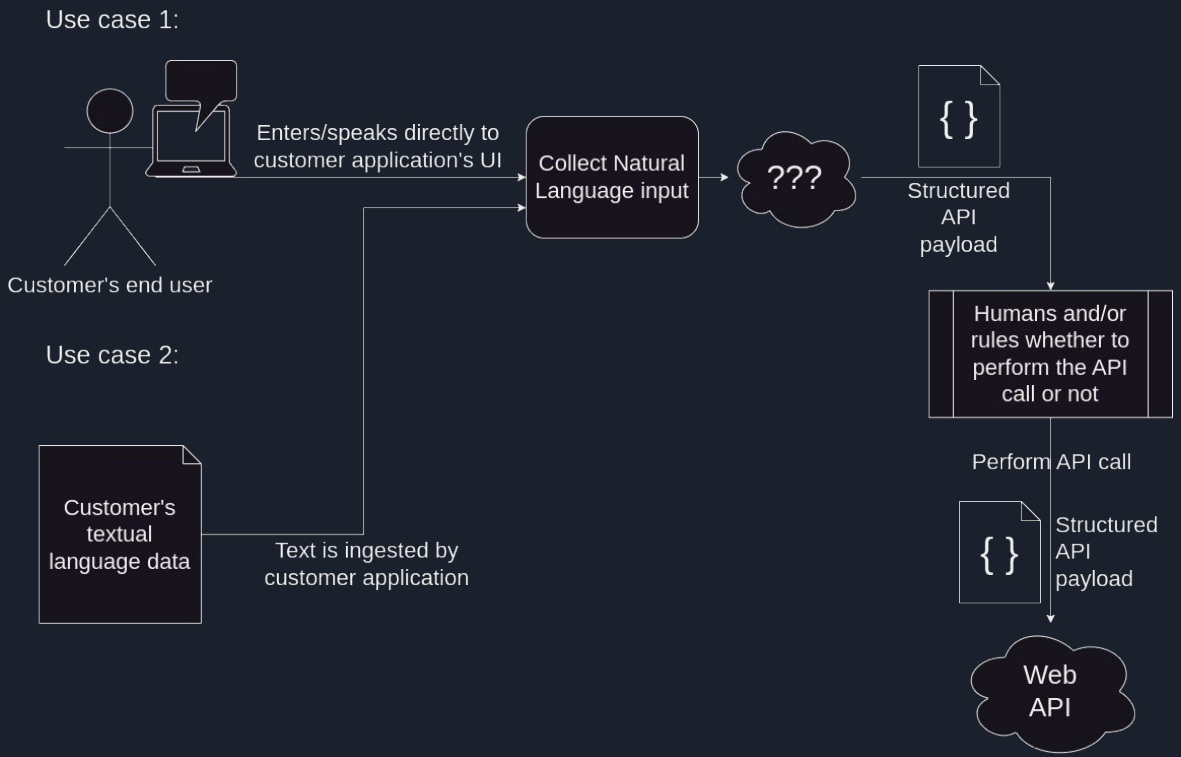


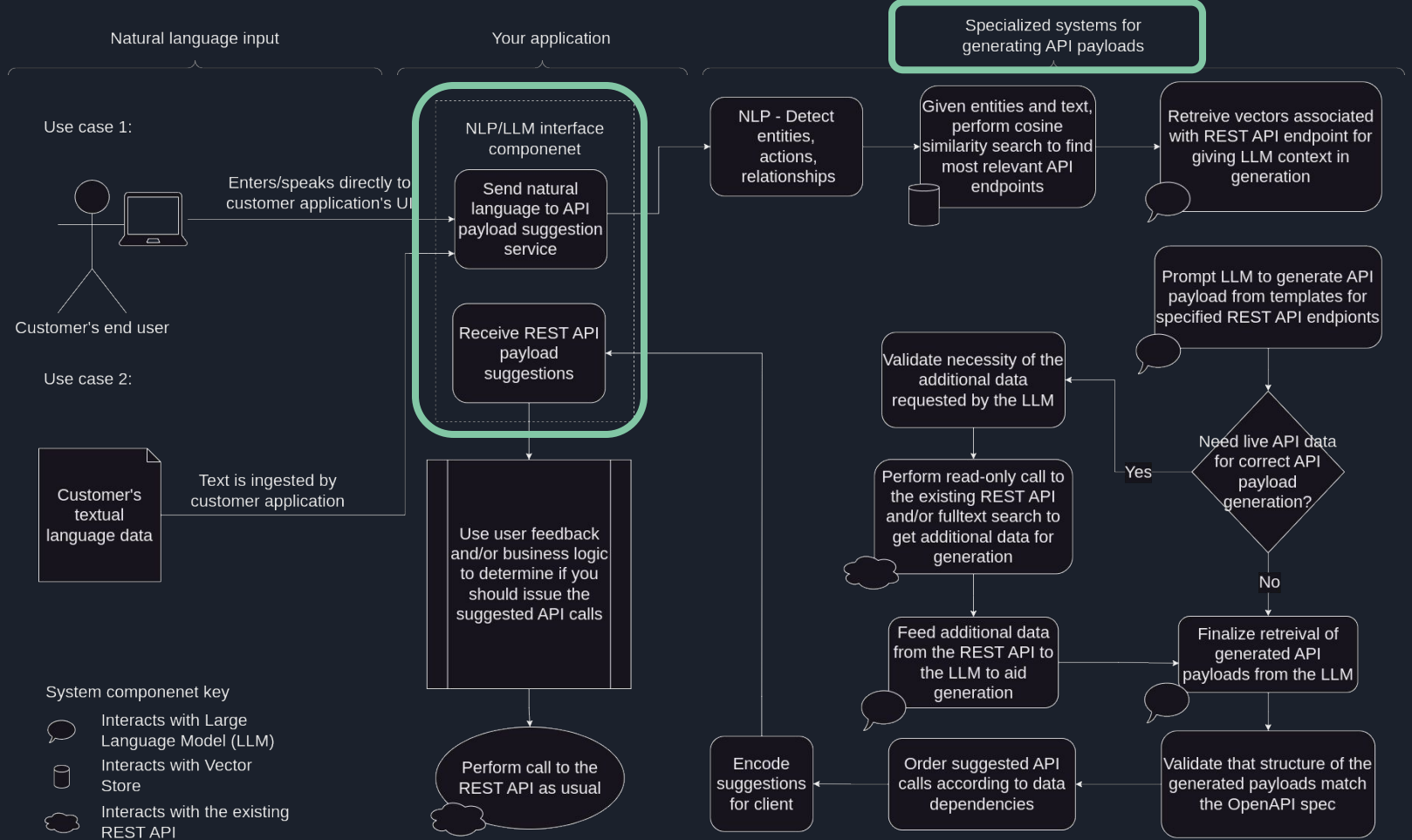
CueCode

Current Process Flow

Two example use cases:

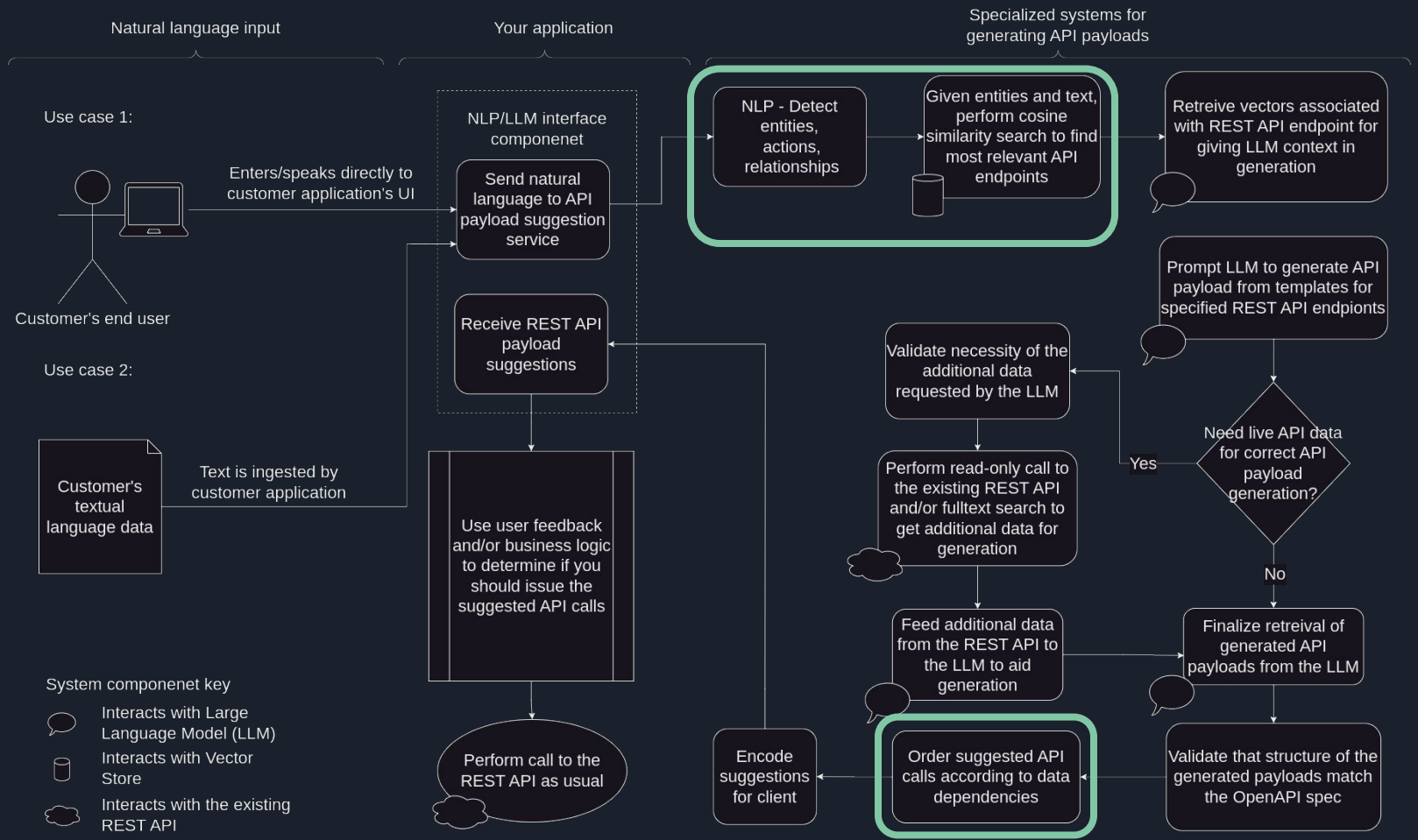
1. Interactive with your end user, validated by user after generation
2. Batch oriented, processed with business logic generation





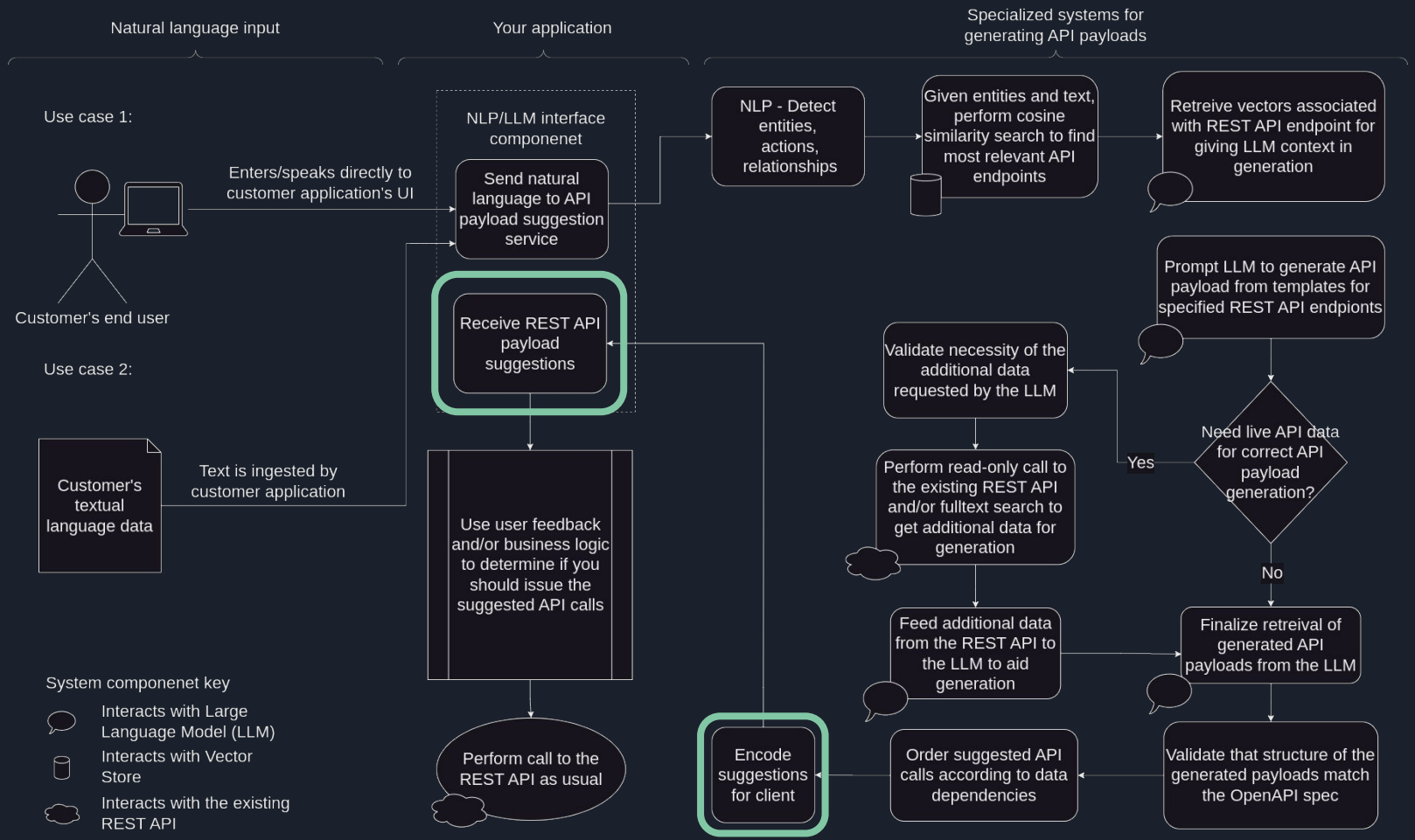
1. Design the interface between the customer's application and the API call generation code.

2. (Not shown) Encode the OpenAPI spec structure for the algorithm to use later when generating and validating payloads.



3. Tag entities and their relationships in the natural language input.

4. (Not Shown) Tell the LLM about the API structure



5. Make the existing application aware of LLM API call suggestions

2.3 Current Process Flow (Validation)

- Verify output is in JSON format (LangChain [9], Guidance AI [6])
- Once an API call is generated, confirm its structure conforms to the schema defined in the OpenAPI spec.
- Confirm that the sequence of data manipulations is consistent with the new/modified entities' relationships.
- For interactive applications confirm the generated API call with the user, and for batch applications, validate the generated API call using business logic.

6. Validate that the generated payloads conform to the OpenAPI spec.

Current Process Flow

A solution for generating API calls would ideally address the following points.

- Design the interface between the customer's application and the API call generation code.
- Encode the OpenAPI spec structure for the algorithm to use later when generating and validating payloads.. Options:
  - In Langchain, build Python classes that define the expected structure of the LLM response (Tool/Function Calling | LangChain, n.d.).
  - OpenAI, use Function Calling schema specification and hope for the best. (OpenAI Platform, n.d.; Tool/Function Calling | LangChain, n.d.)
- Tag entities and their relationships in the natural language input.
- Validate that the generated payloads conform to the OpenAPI spec.
- Tell the LLM about the API structure:
  - One-shot prompt is common in examples, but LLMs struggle to consistently generate responses that are conformant to the spec (Microsoft/Prompt-Engine, 2022/2024).
- Make the existing application aware of LLM API call suggestions:
  - For interactive apps, show the suggestions to the user.
  - For batch processing, push the generated API calls through business logic.
- Validation

No single application or framework on the market addresses all of these concerns, and implementing these solutions manually for each application that wants these features is tedious and requires expertise in using LLMs.

Solution

CueCode will provide a comprehensive service for creating Web API calls from natural language input in a risk-aware, accurate manner that puts developers - and, by extension, users - in control of when API calls are invoked.

Solution Statement

What that means:

Developers will be able to use existing API specifications, which CueCode makes understandable by LLMs, to generate the content of their API calls in conformance with their API spec.

So, our client service representative can provide input to a booking application using CueCode in natural language, “I called Patricia Davis and rescheduled her appointment from August 1st to August 16th.” The application can then use CueCode’s libraries, which have been configured using documentation about the structure of their data, to generate the following Web API request with a JSON request body:

```
POST https://the-appointment-app.com/api/v1/appointments/

{"request":{"reschedule":{"last": "Davis", "first":"Patricia", "from":{"month":8,
"day":1,"year":2024}, "to":{"month":8, "day":1,"year":2024}}}}
```

Which would then be used by the booking application to perform the API call, which will change the appointment date in their database, or prompt the user for additional information.

Solution Characteristics

Problem Characteristics

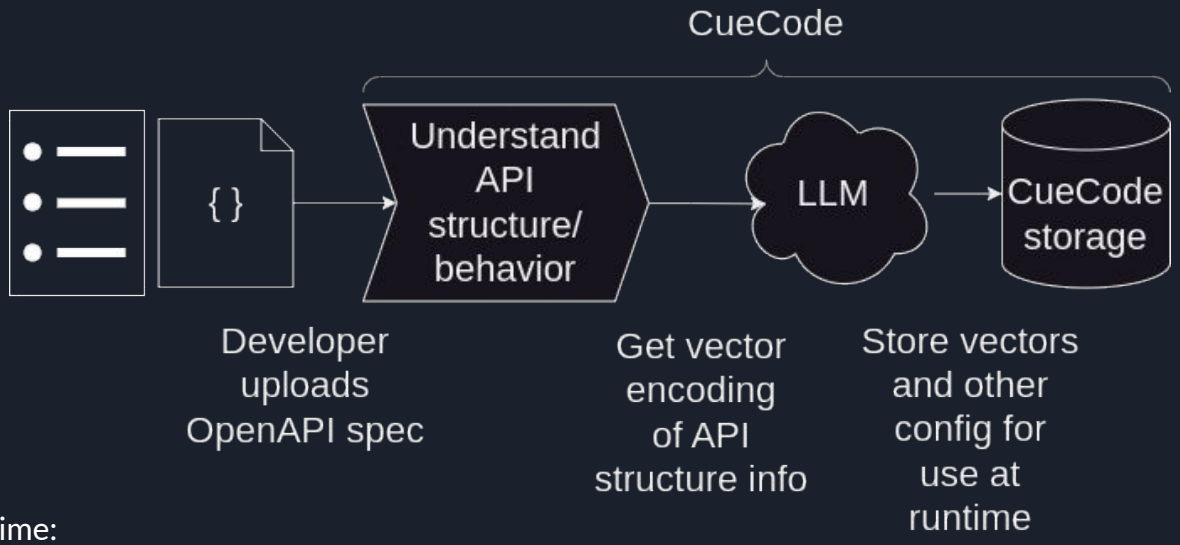
- Forcing end users to fill out lots of forms for input is both limiting and tedious
- There is no easy way to implement using NLP to parse user input for existing applications
- It is difficult to make LLMs aware of the structure of data expected from a natural language prompt
- There is no standardized solution for translating natural language into structured data
- Translating natural language into structured data requires prompt engineering and other skill sets that do not belong to a typical front end or full stack developer
- LLM integration can cause data mutation and incorrect parsing of information

Solution Characteristics

- CueCode leverages LLM technology to parse natural language into structured data to generate API calls, simplifying the process of data entry.
- CueCode provides libraries to front end and full stack developers to easily integrate NLP into their existing applications
- Existing API specifications provide machine-readable input to guide LLMs into parsing user input from natural language, saving developers time and resources
- CueCode facilitates Human-in-the-Loop feedback to allow the end user to review the generated data in the existing user interface

CueCode

Solution Process Flow (configuration)



At configuration time:

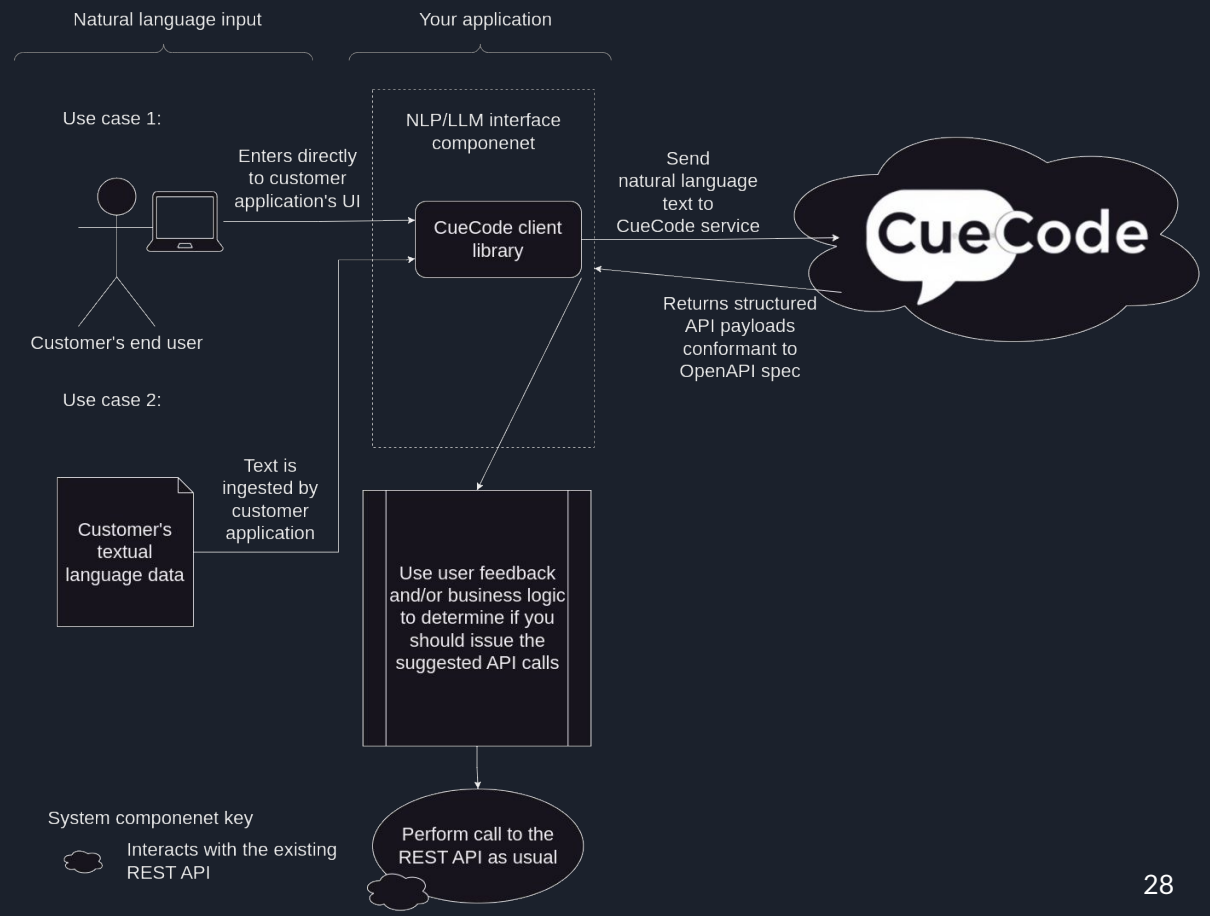
- Developers ensure their OpenAPI specification is accurate.
- Developer uploads their API specification to CueCode via the Developer Portal
- Developer answer a few configuration questions.
- CueCode stores the structure and requirements for the API to aid the LLM in generating responses at runtime.
- All of this is transparent to the Developer's customers/end-users.

CueCode

Solution Process Flow (runtime)

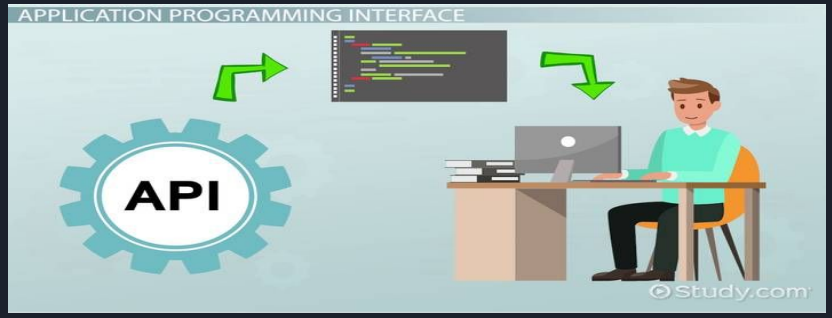
Use CueCode in your application:

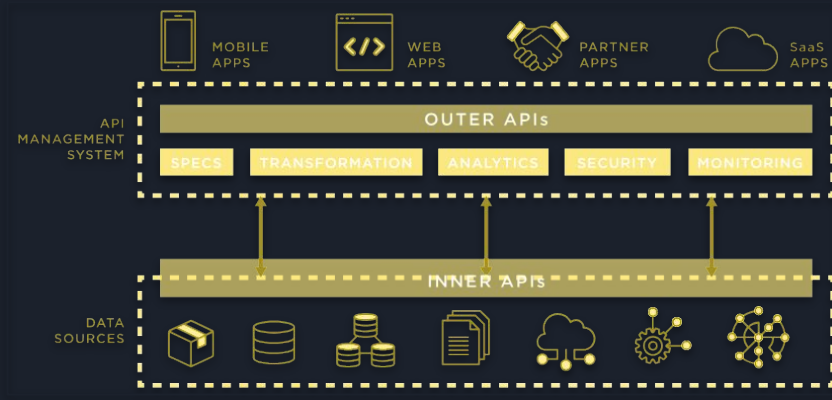
- Program your app to pass natural language text to CueCode libraries.
- Let the CueCode service figure out the structured data contained in the text.
- Use CueCode's extracted structured data within the existing application's data model. e.g.:
  - Show suggestions to the user
  - Perform API calls in a batch job
  - Validate through business rules
  - Whatever the use case requires



What it Will Do

- Will implement NLP capabilities to enable and understand natural language
- Will offer a user friendly interface (API client libraries) that developers can use
- Will provide a Developer Portal web application, where developers can upload API specifications and configure their CueCode service
- Will provide tools for customizing NLP models to fit specific domains/industries ensuring better performance for unique use cases.
- Will include documentation and support resources to help developers implement and troubleshoot various systems effectively.
- Will use REST API specifications, enabling context-aware replies that complement the distinct functionality and data structure of each application.
- Will allow for real time analysis of natural language and REST API call payload generation, enhancing user experience through immediate feedback and interactions.





CueCode

What it Will Not Do

- Will not support languages other than English
- Will not replace human judgment when interpreting language in terms of making subjective decisions beyond its programming.
- Will not act as an AI agent
- Will not provide user-facing applications; developers will need to build their own solutions and install any necessary software/applications they need.
- Will not automatically make API calls on users' behalf; requests must first have human permission before being fulfilled.
- Will not have programming tutorials, developers will need to possess knowledge of programming to utilize CueCode effectively.
- As a student project, CueCode will not generate XML REST API payloads or GraphQL API payloads

Competition Matrix - Introduction

✔- Full Implementation
P - Partial Implementation

CueCode	OpenAI Functions	Google Natural Language API	Spacy.io	LangChain	GenKit	Phone AI Alexa, Siri,...

Competition Matrix - ET

✔- Full Implementation
P - Partial Implementation

Feature	CueCode	OpenAI Functions	Google Natural Language API	Spacy.io	LangChain	GenKit	Phone AI Alexa, Siri,...
Entity recognition	✔		✔	✔		P	✔

Example Prompt:
“I would like to book a hospital appointment on the 20th of October for a medical checkup.”

With entity recognition:
Book, 20th october, medical checkup

CueCode

Competition Matrix - PaP

✔- Full Implementation
P - Partial Implementation

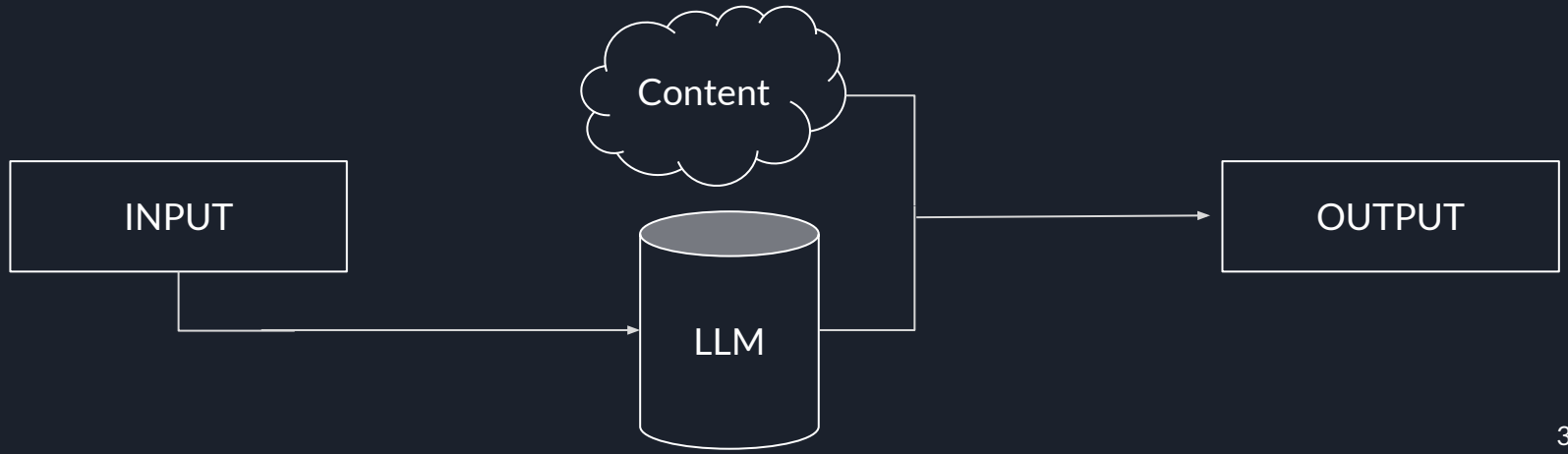
Feature	CueCode	OpenAI Functions	Google Natural Language API	Spacy.io	LangChain	GenKit	Phone AI Alexa, Siri,...
Plug and Play	✔				P	P	P

CueCode

Competition Matrix - RAG

✔- Full Implementation
P - Partial Implementation

Feature	CueCode	OpenAI Functions	Google Natural Language API	Spacy.io	LangChain	GenKit	Phone AI Alexa, Siri,...
Retrieval Augmented Generation	✔	✔			✔	✔	

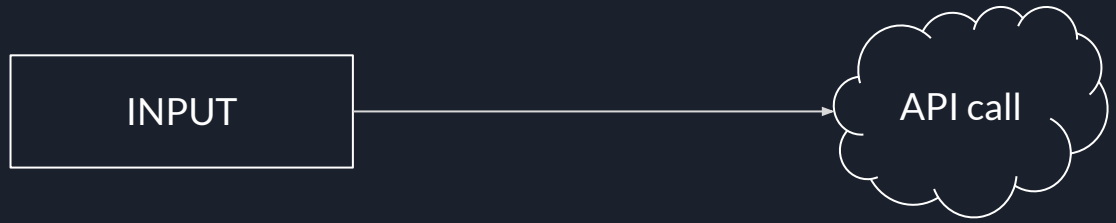


CueCode

Competition Matrix - service

✔- Full Implementation
P - Partial Implementation

Feature	CueCode	OpenAI Functions	Google Natural Language API	Spacy.io	LangChain	GenKit	Phone AI Alexa, Siri,...
API call generation as a service	✔	P	P		P		P



CueCode

Competition Matrix - review

✔- Full Implementation
P - Partial Implementation

Feature	CueCode	OpenAI Functions	Google Natural Language API	Spacy.io	LangChain	GenKit	Phone AI Alexa, Siri,...
Entity recognition	✔		✔	✔		P	✔
Plug and Play	✔				P	P	P
Retrieval Augmented Generation	✔	✔			✔	✔	
API call generation as a service	✔	P	P		P		P

CueCode

Design: How will we do it?

Notes on our engineering and current concept for CueCode's implementation

Features and prototype scope

Who are we building for? User roles

- API Developer
  - Will be able to implement CueCode into the application backend, ensuring that the generated API request conform to the API specifications.
  - Run through tests to ensure the system is generating the correct API request without errors.
- API Documentation Specialist
  - Will ensure that API specifications and documentation are clear, structured so they can be easily understood by CueCode.
- User Interface Developer
  - Will be creating the user interface that interact with CueCode allowing user to input natural language and interact with the system.
  - Ensure the interface provides a smooth, intuitive experience for users to interact with the NLP features.
- Data Security
  - Will ensure that the CueCode system handles natural language input securely and that any data generated through the API calls complies with data privacy regulations
- Customer's end user
  - This is the end user who will submit natural language that will (eventually) get processed by CueCode

CueCode

Feature table - Developers and publishers

Category	Feature				
Portal	Login / Authentication		✔	✔	✔
	Account creation / deletion		✔	✔	✔
Config	REST API definition management		✔	✔	✔
	Upload and manage OpenAPI specifications		✔	✔	✔
	REST API configuration wizard		✔	✔	✔
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	✔	✔	✔	
Marketplace	Share CueCode API configurations with other users		✔	✔	✔

Real-world vs. Prototype feature

Category	Feature	Real-world product	Prototype
Portal	Login / Authentication	✔	✔
	Account creation / deletion	✔	✔
Config	REST API definition management	✔	✔
	Upload and manage OpenAPI specifications	✔	✔
	REST API configuration wizard	✔	✔
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	✔	
Marketplace	Share CueCode API configurations with other users	✔	

User interface

User interface - outline

Login Page

Developer Dashboard

- Summary of API usage, configuration progress, and performance metrics.
- Quick access to "Upload OpenAPI Spec," "View Suggested Calls," and "Monitor Configurations."

Upload and Configuration

- OpenAPI Spec Upload:
  - File upload interface with validation.
  - Feedback on upload success/failure and spec completeness.
- Configuration Setup:
  - List of endpoints from the uploaded OpenAPI spec.
  - Interface to add natural language prompts for each endpoint.
  - Suggestions for optimizing the OpenAPI spec for CueCode.
- Entity Mapping:
  - Tools to map natural language entities to API parameters.
  - Auto-suggestions for commonly used mappings.

Input and Payload Management

- Input Submission:
  - Text input box for testing natural language commands.
  - Option to upload batch text inputs for processing.
- Generated API Calls:
  - Display of structured JSON with suggested API calls.
  - Sequence validation to ensure correct order of execution.
  - Error explanations when no results are generated.

Monitoring and Auditing

- Monitoring Dashboard:
  - Real-time view of input/output auditing.
  - Graphical representation of acceptance rates and usage trends.
- Detailed Auditing:
  - Log of all inputs with corresponding outputs.
  - Error reports for problematic inputs.
  - Configuration feedback for improving results.

Client Library Interaction

- **Library Setup**:
  - Instructions and code snippets for installing and initializing client libraries.
  - Authentication token management for client-side use.
- **API Interaction**:
  - Tools to test integration directly within the portal.
  - Error handling guides for common client-side issues.

Marketplace Integration

- **Search and Use Templates**:
  - Search bar with filters for popular and effective configurations.
  - Preview and copy options for template configurations.
- **Publish Configuration**:
  - Form to upload and share configurations.
  - Visibility settings (public/private).

Help and Support

- **FAQ and Documentation**:
  - Links to guides for client library usage, OpenAPI spec preparation, and configuration.
- **Support Tickets**:
  - Interface for submitting issues and tracking resolution status.

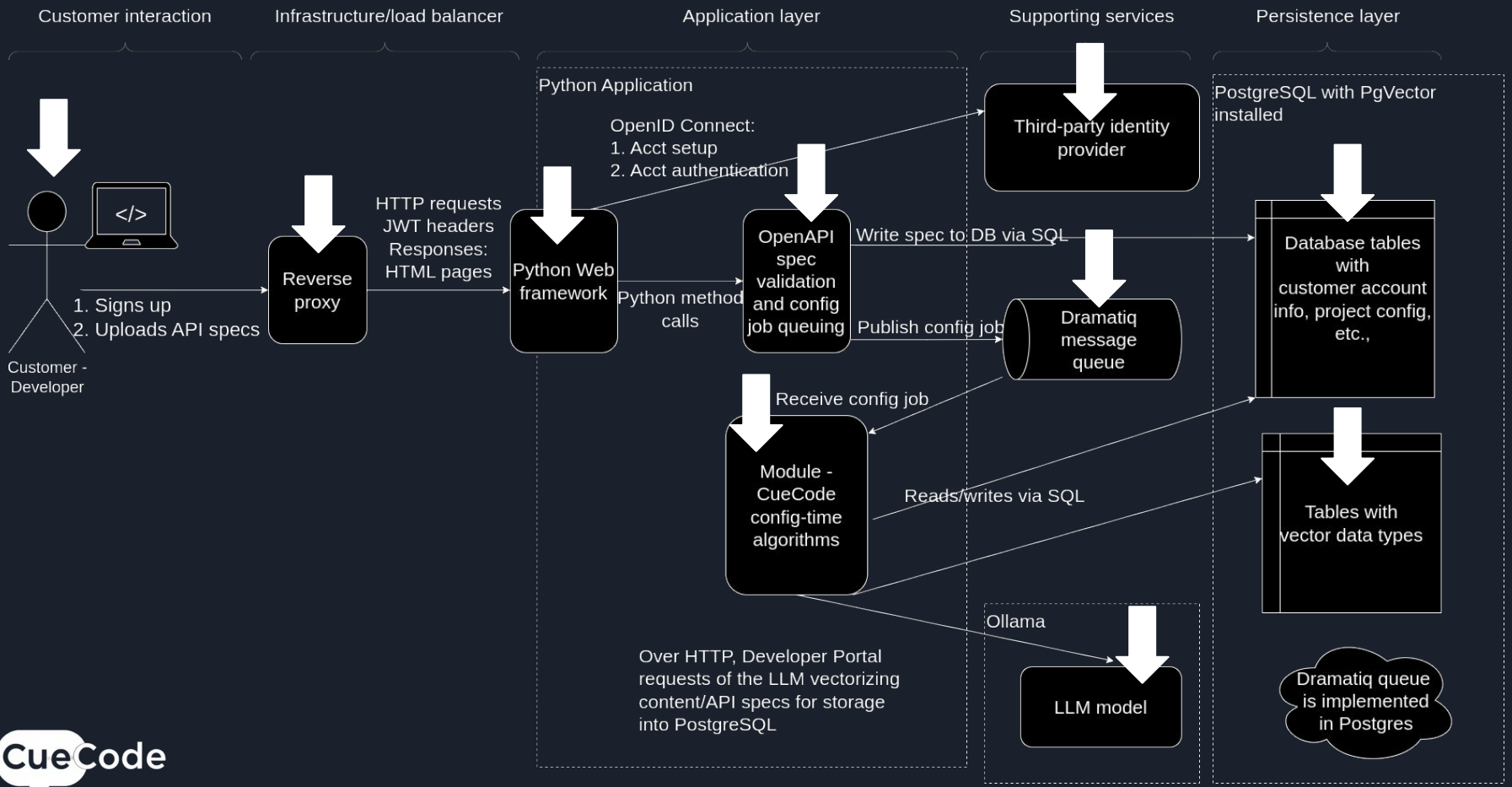
Logout

- Accessible logout option available from any page.

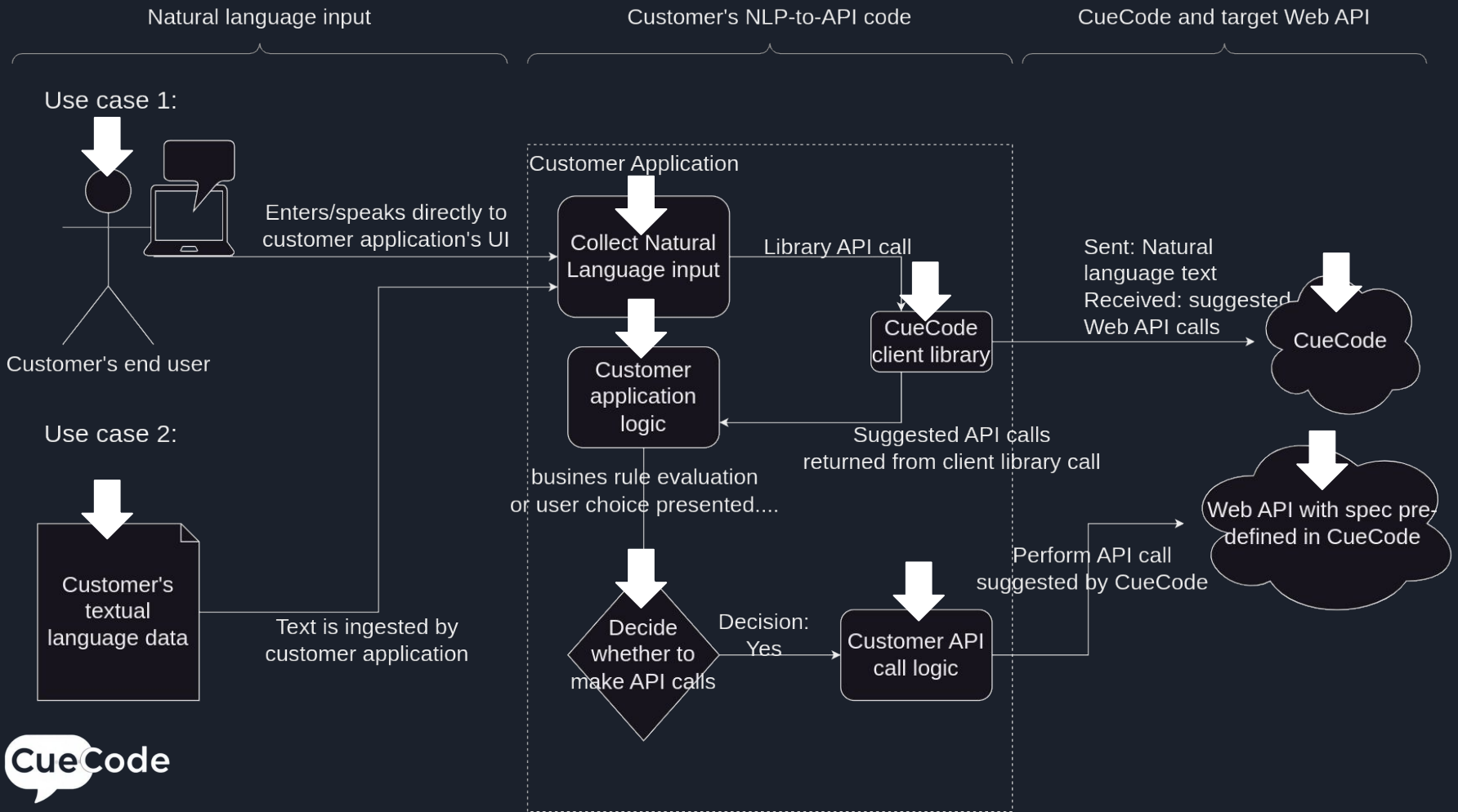
Major Functional Components

- Client libraries for customers to use for integrating with CueCode's service
  - Bindings for the CueCode runtime API
- Python modular monolith:
  - All modules exposed via Flask, a Python Web framework
  - Module: Web API payload Generation- receives natural language input and generates Web API calls from it.
  - Module: Developer Portal - account registration/management, API spec upload, configuration, generation audit and monitoring
  - Horizontally scalable via 12-factor app methodology
- PostgreSQL (Postgres) persistence:
  - PgVector extension for storing vectors generated by the LLM
  - Normal Postgres tables for customer accounts, configuration, generation monitoring and audit information
- Dramatiq message queue
  - Implemented in Postgres; can change pub/sub mechanism to RabbitMQ or Redis as more performance is needed. Simple implementation favored.
- Ollama:
  - A Web service and set of standardized LLM-call APIs that allows us to swap LLMs used while maintaining the same API contract with our Python backend.
- Third-party identity service:
  - For developer portal
  - TBD on how/whether CueCode runtime API traffic would use the same identity provider for authentication.

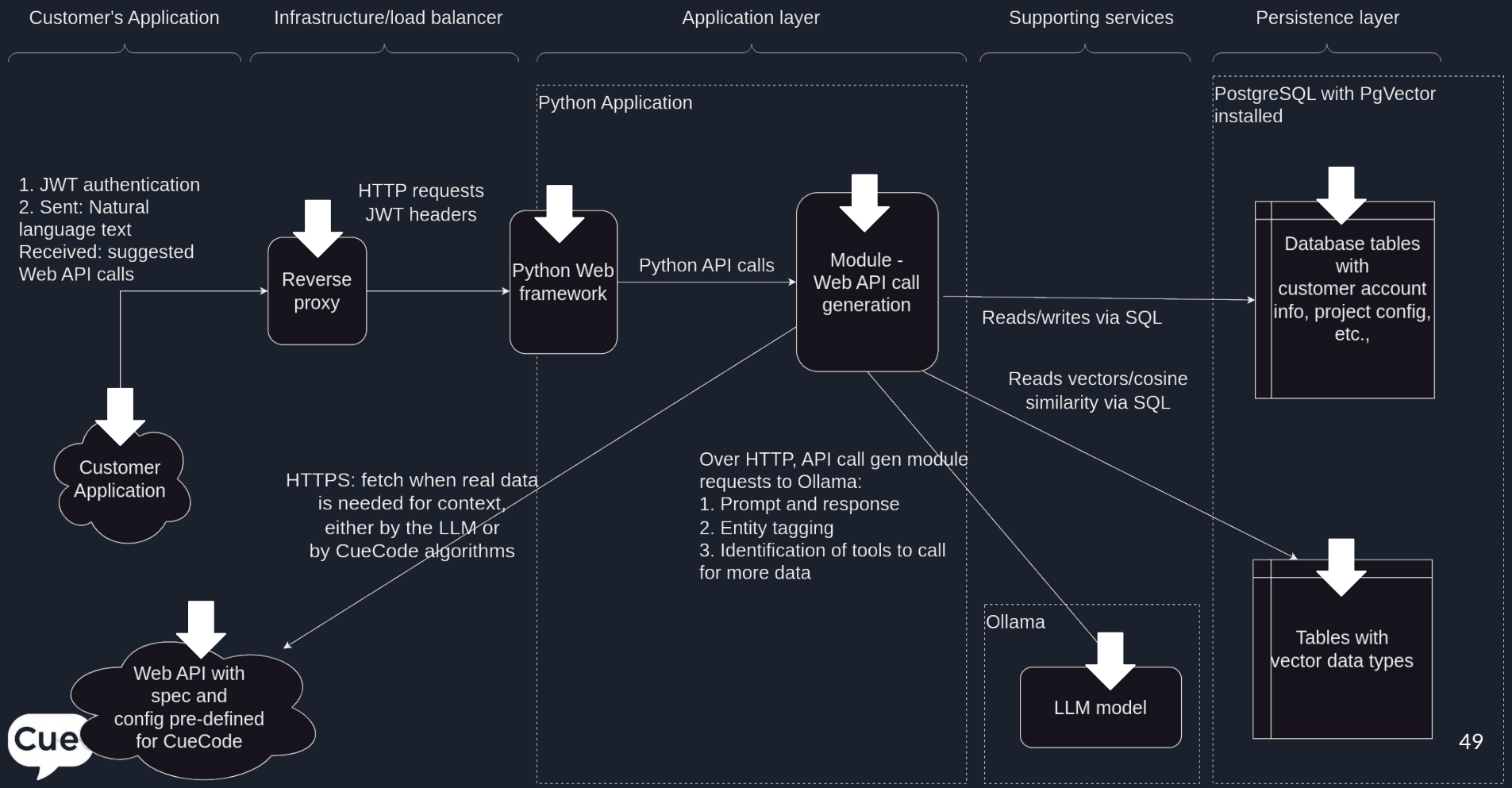
Major Functional Components Diagram - Configuration



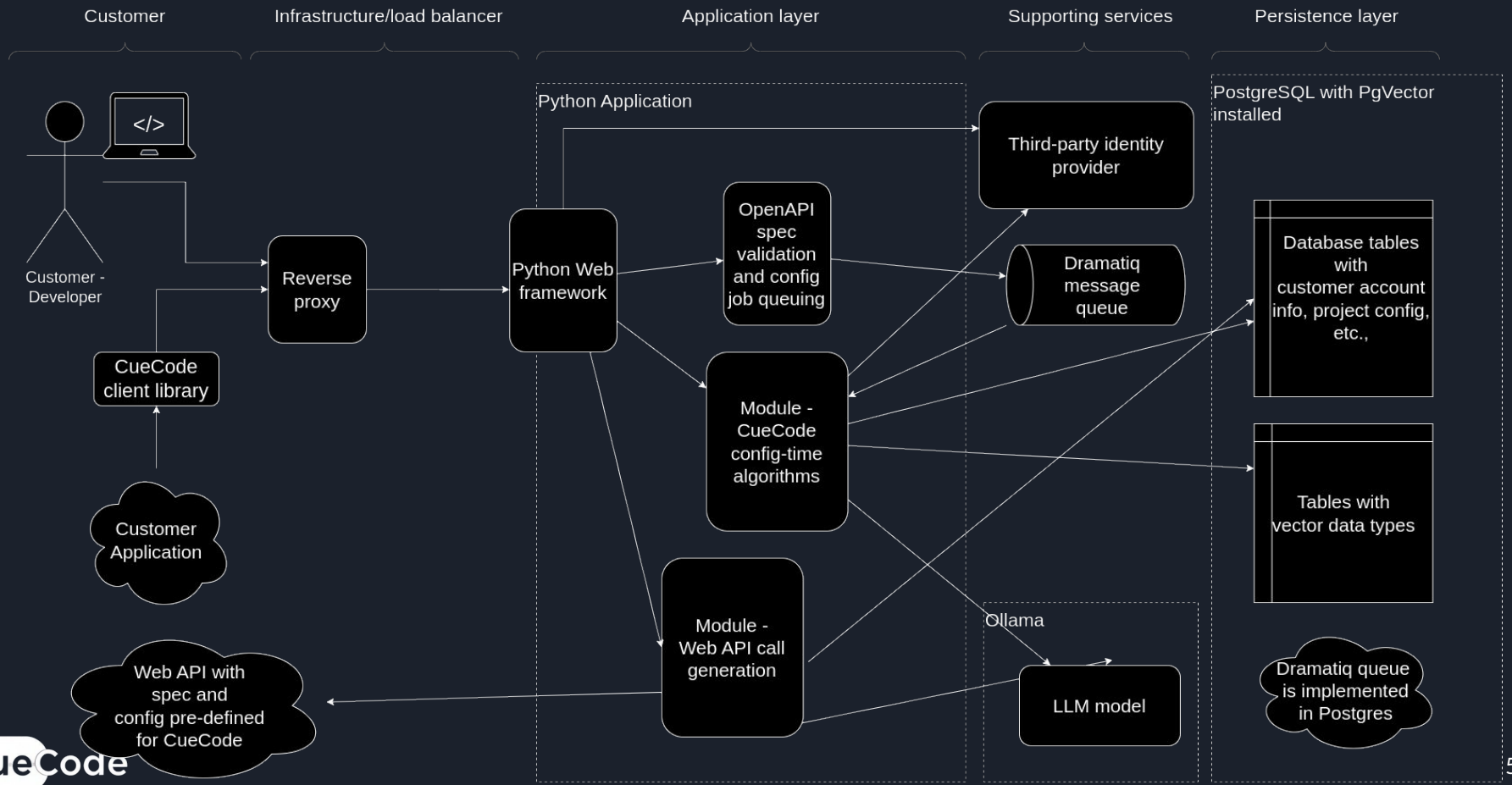
Major Functional Components Diagram - Runtime - Customer Application



Major Functional Components Diagram - Runtime - Payload Generation



Major Functional Components Diagram - Overview



Summary of key Algorithms

Configuration algorithm summary

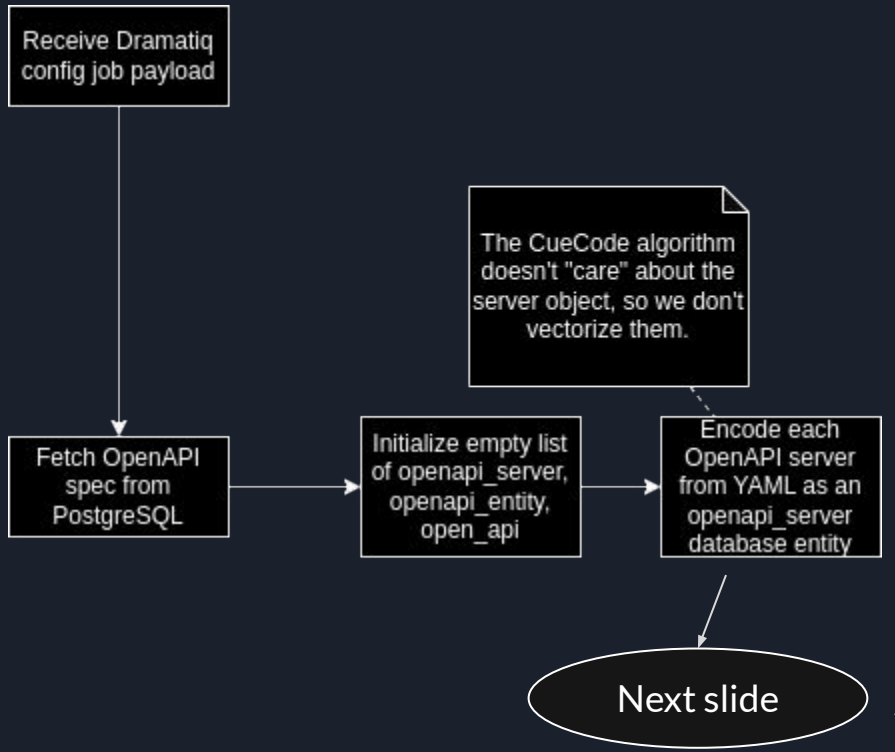
Overview of what the algorithm does:

- Organizes OpenAPI spec into the relational database used to power runtime algorithms.
- Vectorizes of english prompts and keywords and stores them in the relational database to power searches at runtime.
- Stores OpenAPI HTTP endpoints and HTTP verbs as LLM tool calls for later use

Configuration algorithm details and diagram

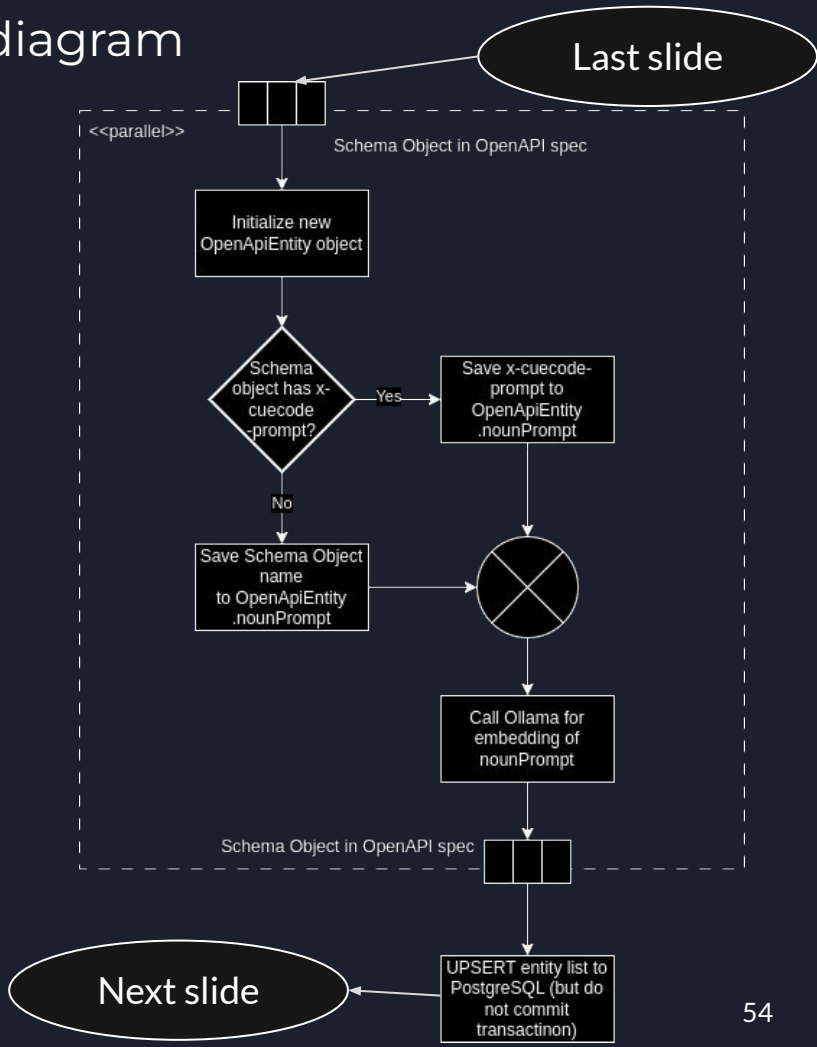
- Prepare lists to be filled with OpenAPI Servers, Entities, and Paths
- Note: Any error results in job retry after a delay that is exponentially increased with each failure.

Examples:
Server: https://api1.example.com
Path: https://api1.example.com/v1/appointments
Entity, appointment: {time: ….., date: .... }



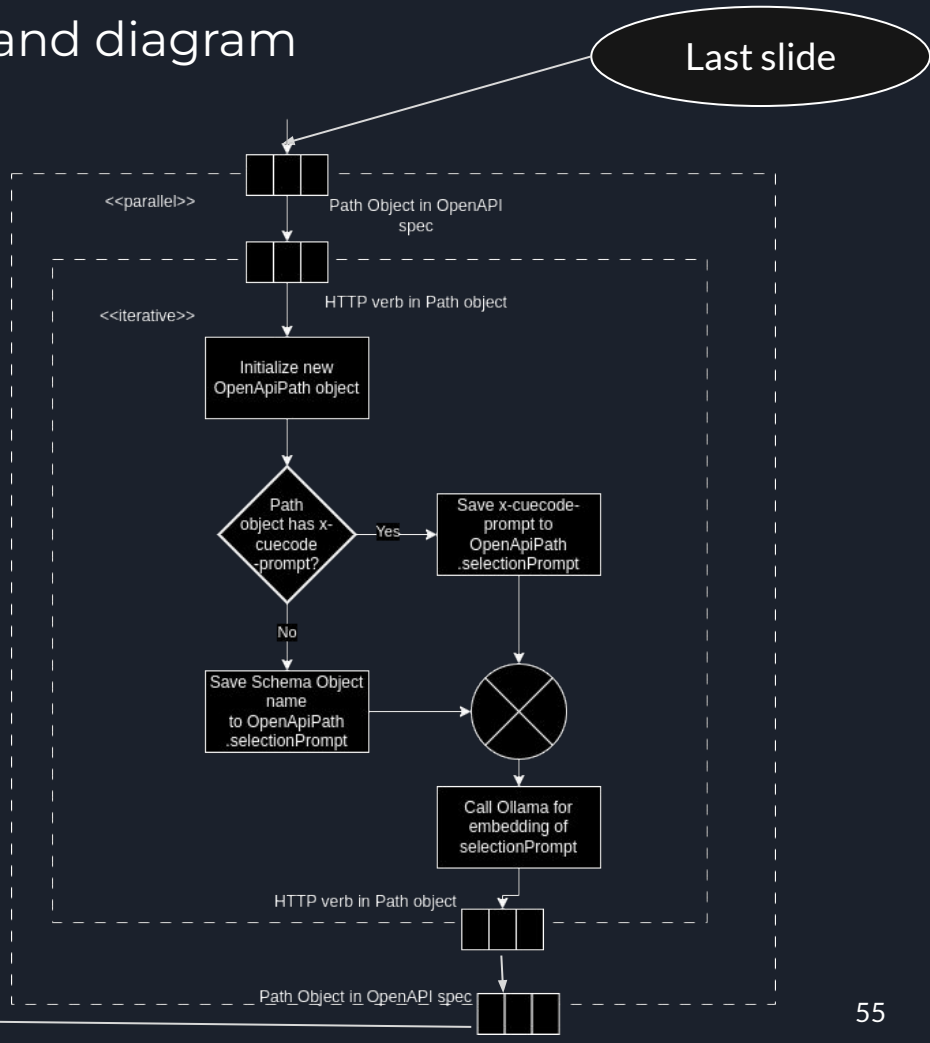
Configuration algorithm details and diagram

- Detect custom prompts if present
- Do entity storage and embeddings
- Handle custom prompts in spec file
- Same error handling as before



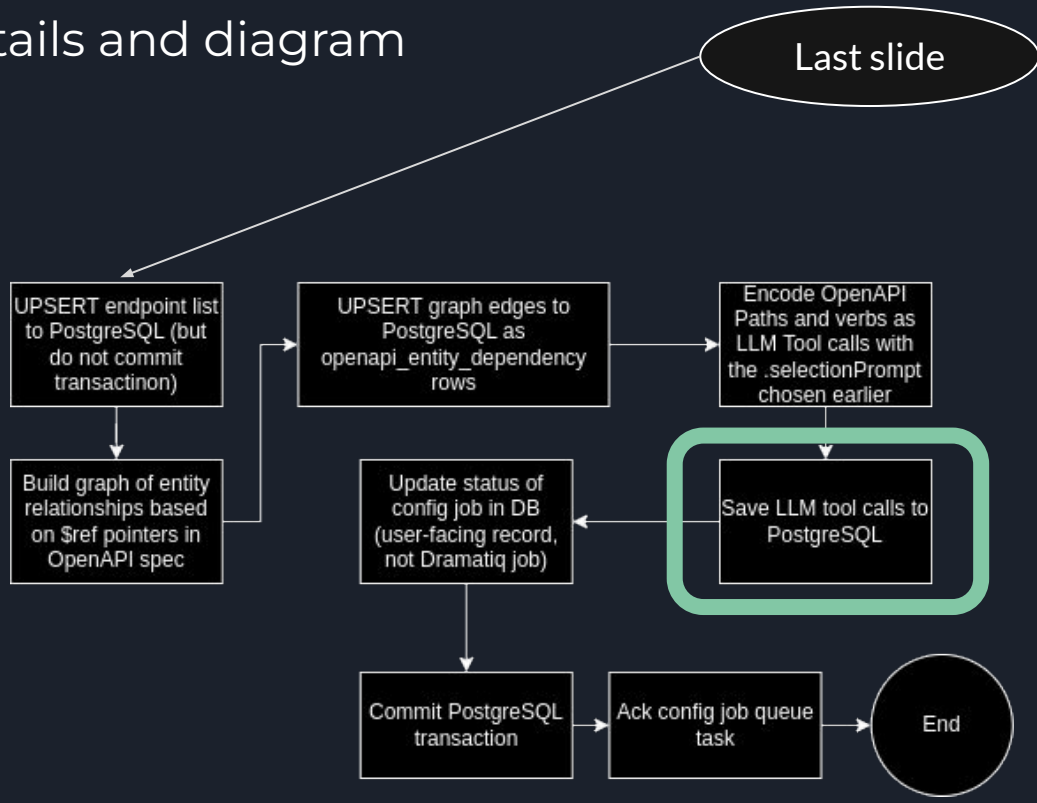
Configuration algorithm details and diagram

- Detect custom prompts if present
- Do path storage and embeddings FOR HTTP EACH VERB available (key difference from Entity processing)
- Handle custom prompts in spec file
- Same error handling as before



Configuration algorithm details and diagram

- Finalize endpoint list within transaction
- Handle REST API entities' dependencies
- Key: Save LLM tool call spec to Postgres
- Same error handling as before

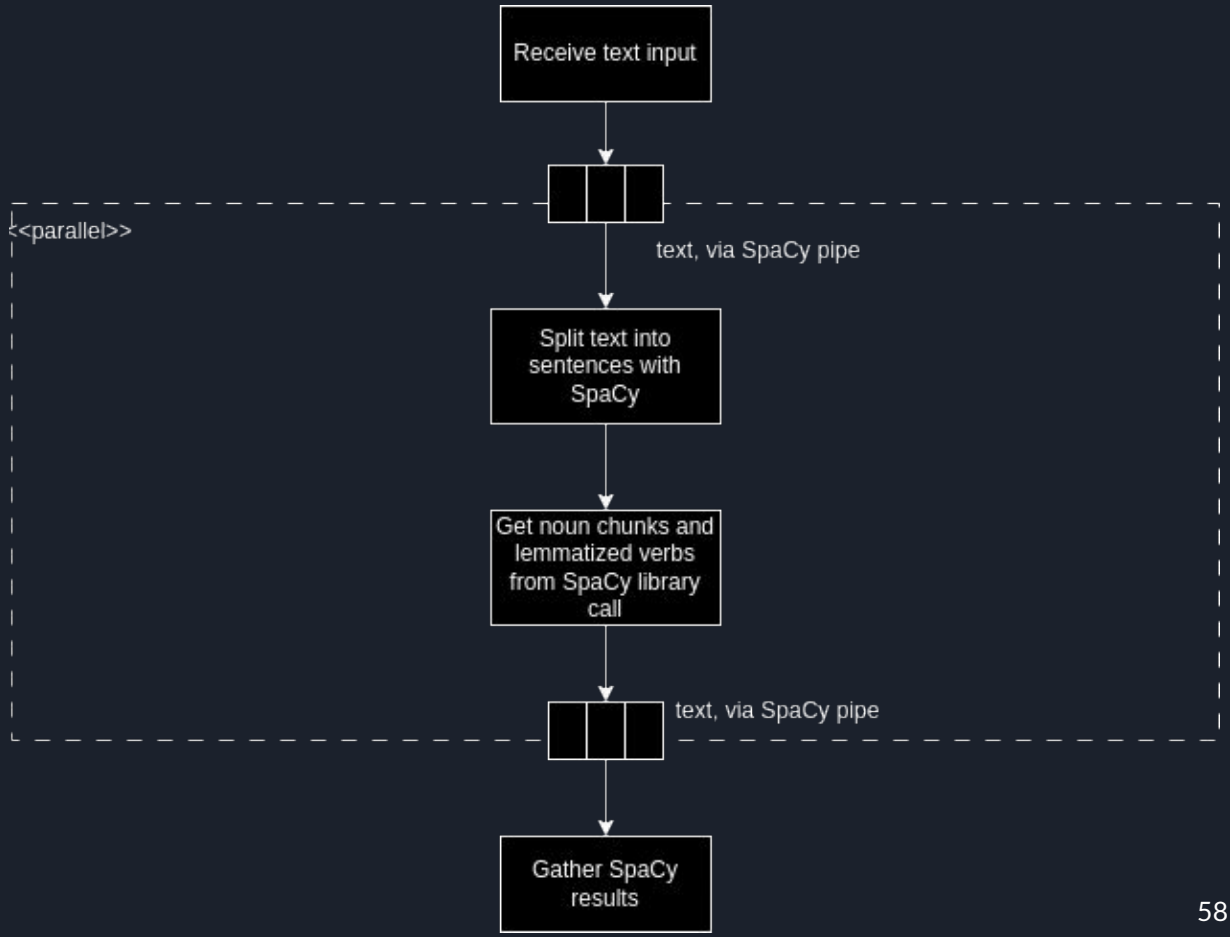


Runtime algorithm summary

- Natural language processing - detect entities, actions, relationships in text.
- Map detected verbs and nouns to HTTP verbs and entities represented by the REST API, respectively.
- Given entities and and verbs, find the most relevant API endpoints for which to generate payloads, since we cannot supply large OpenAPI specifications to the LLM prompt due to prompt overflow.
- (Not in prototype) Determine if live API data is needed to translate natural language, or if data in the natural language is sufficient to create payloads. Fetch any data needed from the target API.
- LLM Function Calls for generating consistently structured JSON output
- Prompt LLM to generate API payload using defined function calls

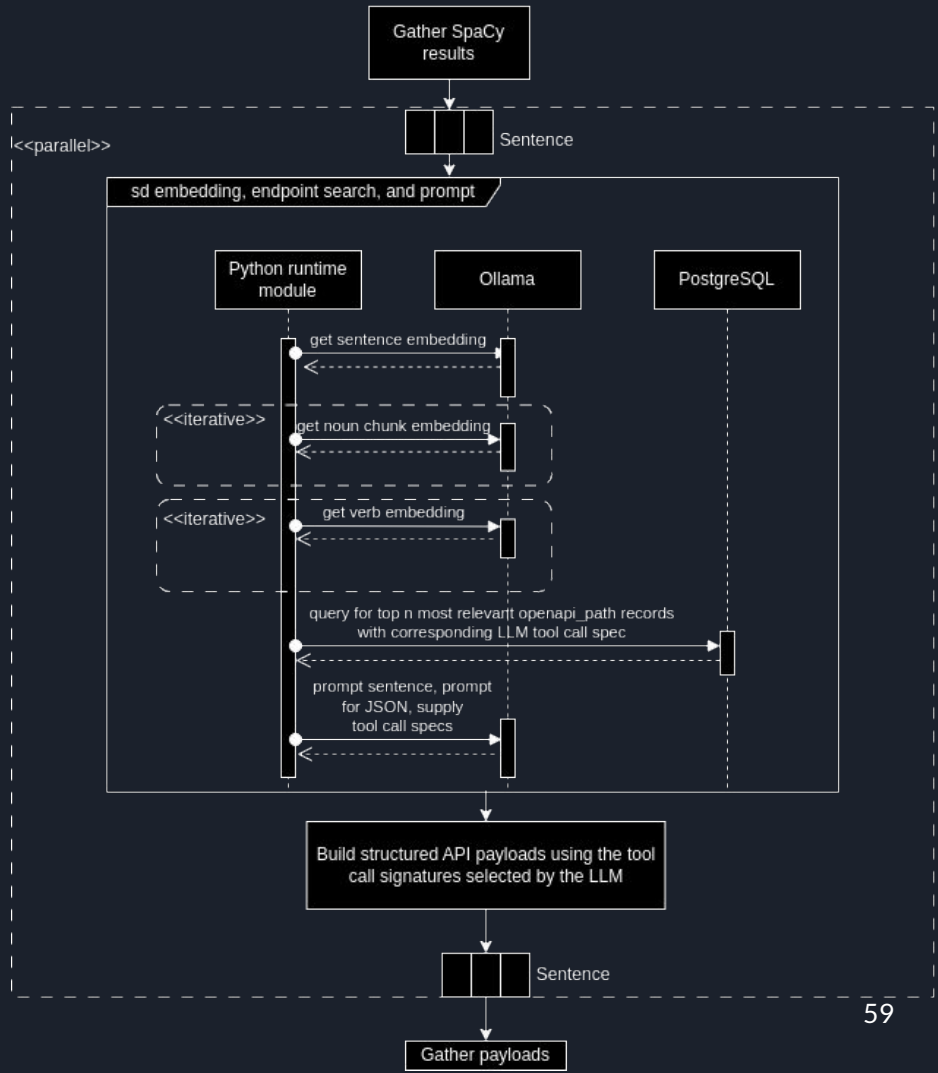
Split sentences, identify nouns and verbs with SpaCy

- Natural language processing - detect entities, actions, relationships in text.
- Get the lemma of each verb in the sentence (SpaCy Linguistic Features - Lemmatization, n.d.)
- Get the noun chunks in each sentence (SpaCy Linguistic Features - Noun Chunks, n.d.)
- SpaCy pipe for parallel processing (Rao, 2024)



Perform embeddings, query for relevant endpoints, prompt LLM

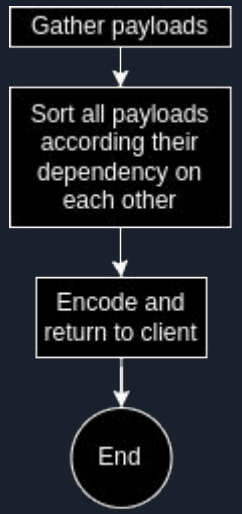
- Embed nouns and verbs
- Cosine similarity search to find the most relevant endpoints, based on endpoint prompt, noun, and verb embeddings, with SQL join logic to bring it together.
  - (Pgvector/Pgvector, 2021/2024; Prabhakaran, 2018; Zafin, 2023)
- The LLM specifies a tool call and with which parameters to call the tool. Our tools generate payloads.
- Optimizations made:
  - Parallel execution of IO bound processes works well in Python (but not CPU bound processes) due to the Global Interpreter Lock (GIL) (Jesse, 2017).
- Potential optimizations left on the table with the prototype implementation:
  - Precomputation of verb and/or noun embeddings
  - Caching layer
  - Use an embedding with fewer dimensions



Sort payloads according to data dependencies

Handling data dependencies between entities referenced or created in generated payloads:

- Sort according to data dependencies, some entities not created yet
- => Symbol table for entities as augment to actual entity IDs like 1234
- Client library receives the symbol table from the CueCode runtime algo
- Data dependencies come from the database, openapi_entity table



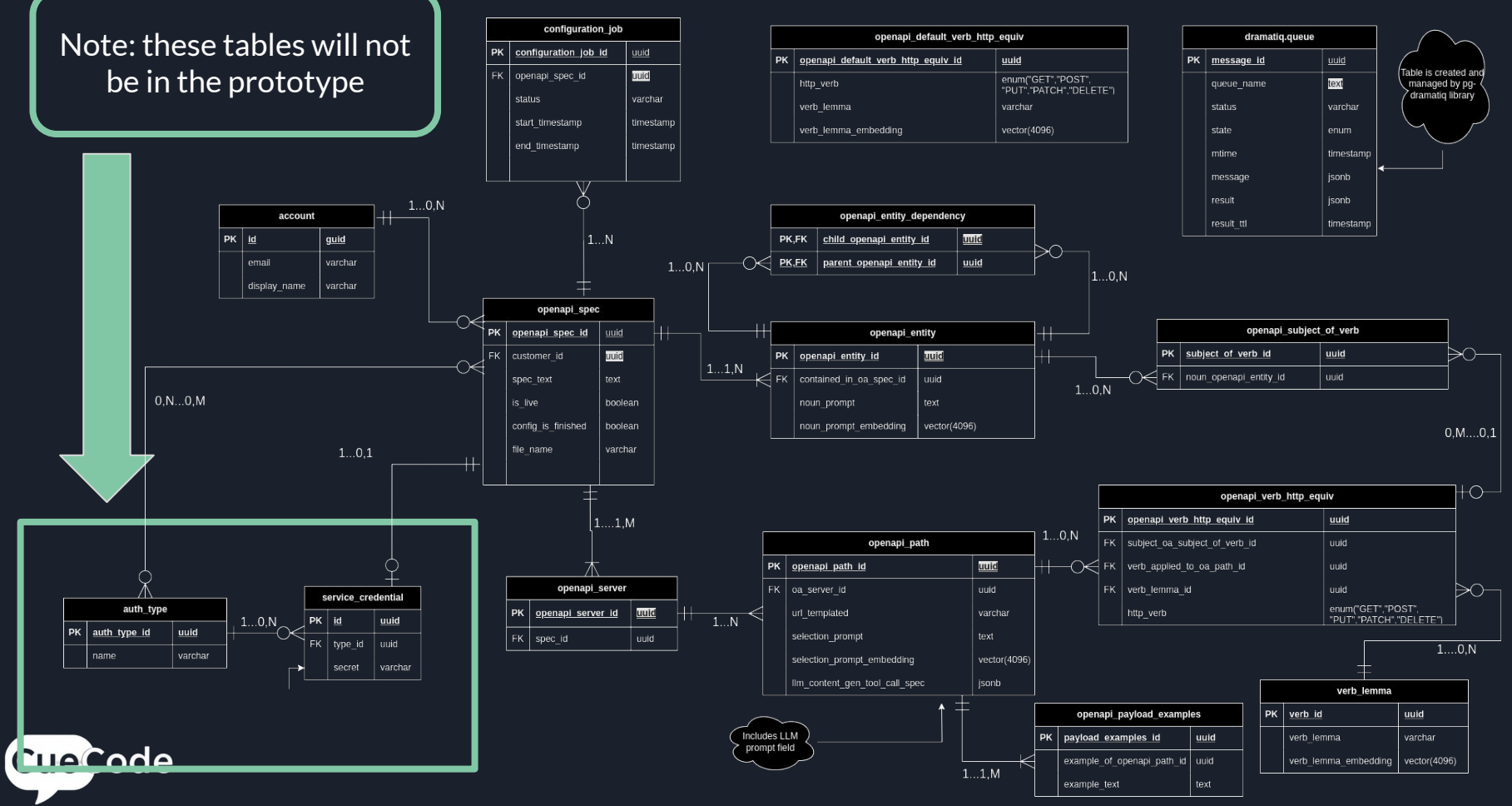
Data design and management

Database design

- Minimum data storage needs for the CueCode algorithms:
  - API spec document (it is simplest to keep in the DB as a TEXT column)
  - OpenAPI Paths, the spec for each API endpoint
  - OpenAPI Schema Objects, which define the entities represented in the API. CueCude stores these as entities in the openapi_entity table.
  - OpenAPI Server object, because all Paths rely on a Server object to give their base URL.
- Need to facilitate three (3) cosine similarity searches for:
  - Endpoint selection - openapi_path table
  - Entity selection - openapi_entity table
  - Verb lemma recognition - verb_lemma table
- Handle asynchronous processing of the OpenAPI specs, as uploaded to the CueCode Developer Portal:
  - Table configuration_job stores info about each configuration run for showing to the user
  - Table dramatiq.queue is an implementation detail of our database-based message broker.
- Normalization of vector tables is optimized to provide vector search while minimizing storage

Entity Relation Diagram (ERD)

Note: these tables will not be in the prototype



Database management

- PostgreSQL
  - Support for PgVector
  - Entity relationship diagram
  - Docker container support
- PgVector extension
  - allows to store vector data -> recommendations / similarity searches
- Dbmate for schema management
- Pg-dramatiq python library for managing asynchronous task queue
- Backups: Automation of pg_dump command-line utility to take backups and imports

Software and hardware tools

Software / Hardware Tools (1 of 2)

Software	Hardware

• Frontend:
  ○ HTML
  ○ CSS
  ○ Bootstrap 5
  ○ JavaScript
• API clients
  ○ Swagger CodeGen
• Application layer
  ○ Python
  ○ Flask Web framework
  ○ Jinja HTML templating
  ○ Dramatiq for asynchronous task processing
• Application libraries
  ○ Spacy.io for NLP
  ○ Ollama Python client
  ○ OpenAPI spec validator
  ○ Authentication:
  ■ Flask>=2.2
  ■ Flask-Session>=0.3.2,<0.6
  ■ werkzeug>=2
  ■ requests>=2,<3
	• LLM
	  ○ Ollama
	  ○ Llama 3.2
	• Third-party
	  ○ identity service
	  ○ Transactional email
	• Testing
	  ○ Jest
	  ○ PyUnit
	  ○ PyLint (enforce module boundaries, etc.)
	• CI/CD
	  ○ GitHub Actions
	  ○ Docker registry
	• Hardware
	  ○ GPU-equipped Kubernetes node(s) in CS Systems Group cluster

Development Tools:

Languages

- HTML, CSS, JavaScript
  - For frontend development and UI/UX.
- Python
  - Backend application logic and integrations.

Frameworks

- Bootstrap 5
  - CSS framework for responsive and modern design.
- Flask
  - Lightweight Python web framework for backend development.
- Jinja
  - Templating engine for rendering dynamic HTML.

Platform

- Docker
  - Containerization for consistent environments across development and production.

IDE

- Visual Studio Code (VS Code)
  - Preferred IDE with powerful extensions for Python and web development.

Developer OS

- Linux
  - Stable and compatible development environment.
- Windows
  - Can be used with WSL or Docker for seamless development.

API and Components

- Swagger CodeGen
  - Generates API client libraries for integrations.
- Spacy.io
  - NLP library for natural language processing.
- Ollama Python Client
  - API interface for interacting with LLM services.
- OpenAPI Spec Validator
  - Ensures uploaded API specs meet required standards.

Database

- PostgreSQL
  - Core database for managing user data and configurations.
- PgVector
  - Extension for storing vector embeddings from LLMs.
- pg-dramatiq
  - Postgres-based message broker for asynchronous task management.

Version Control

- Git with GitHub
  - Manages source code and facilitates collaboration.

Testing Framework

- Jest
  - Frontend testing framework for JavaScript components.
- PyUnit
  - Unit testing for backend Python application logic.
- PyLint
  - Enforces code quality and module boundary rules.

Benchmarking

- Timeit
  - Simple benchmarking tool for python functions
- cProfile
  - More advanced profiling tool for finding bottlenecks in python code

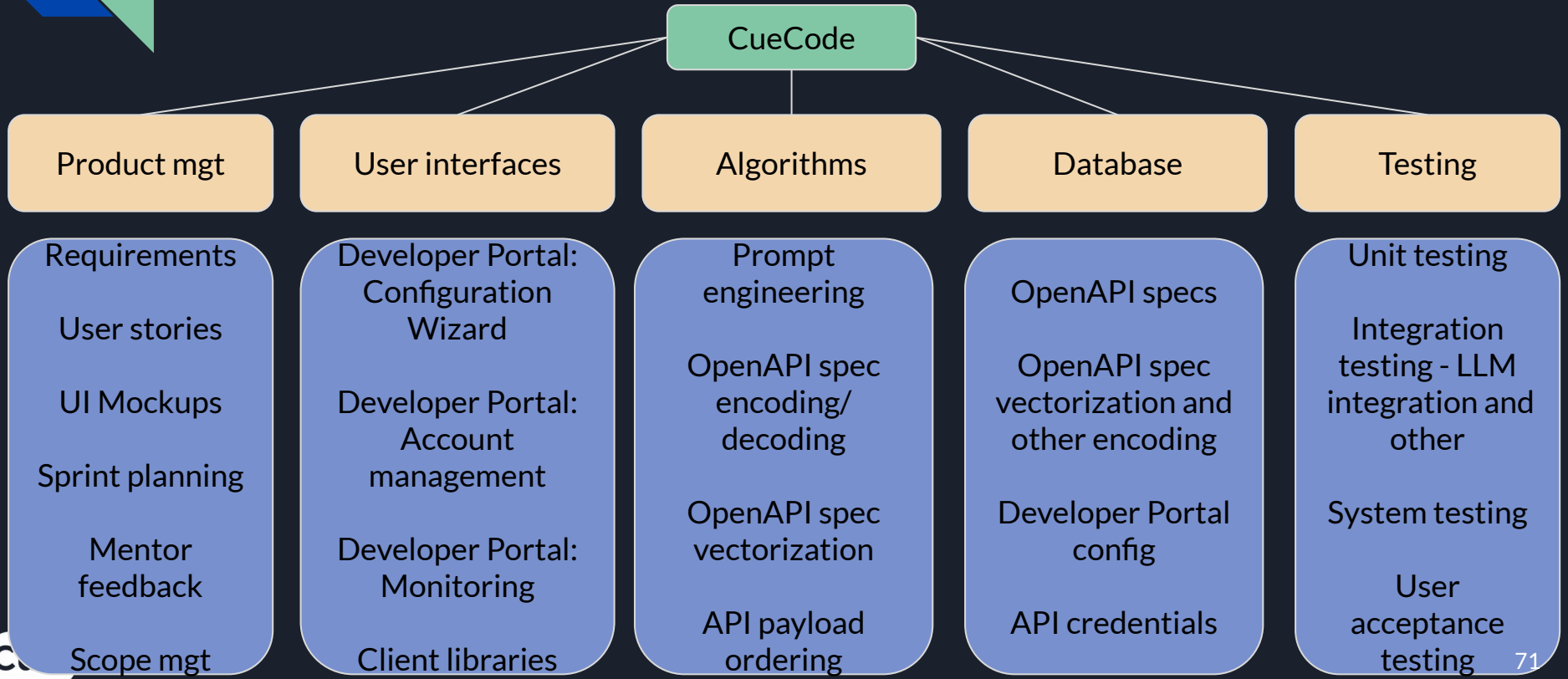
CI/CD

- GitHub Actions
  - Automates testing, builds, and deployments.
- Docker Registry
  - Stores containerized application images for seamless deployment.

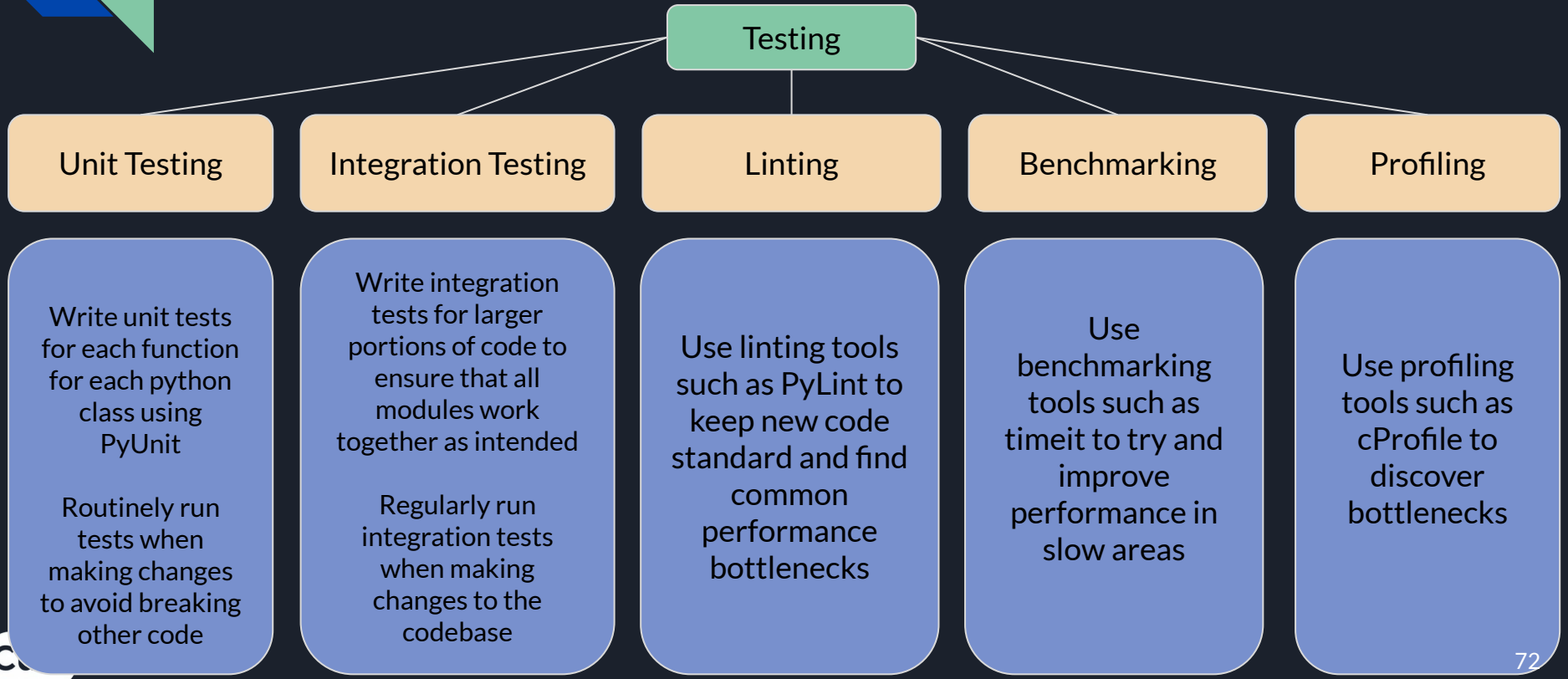
Work breakdown structure

What needs to be done

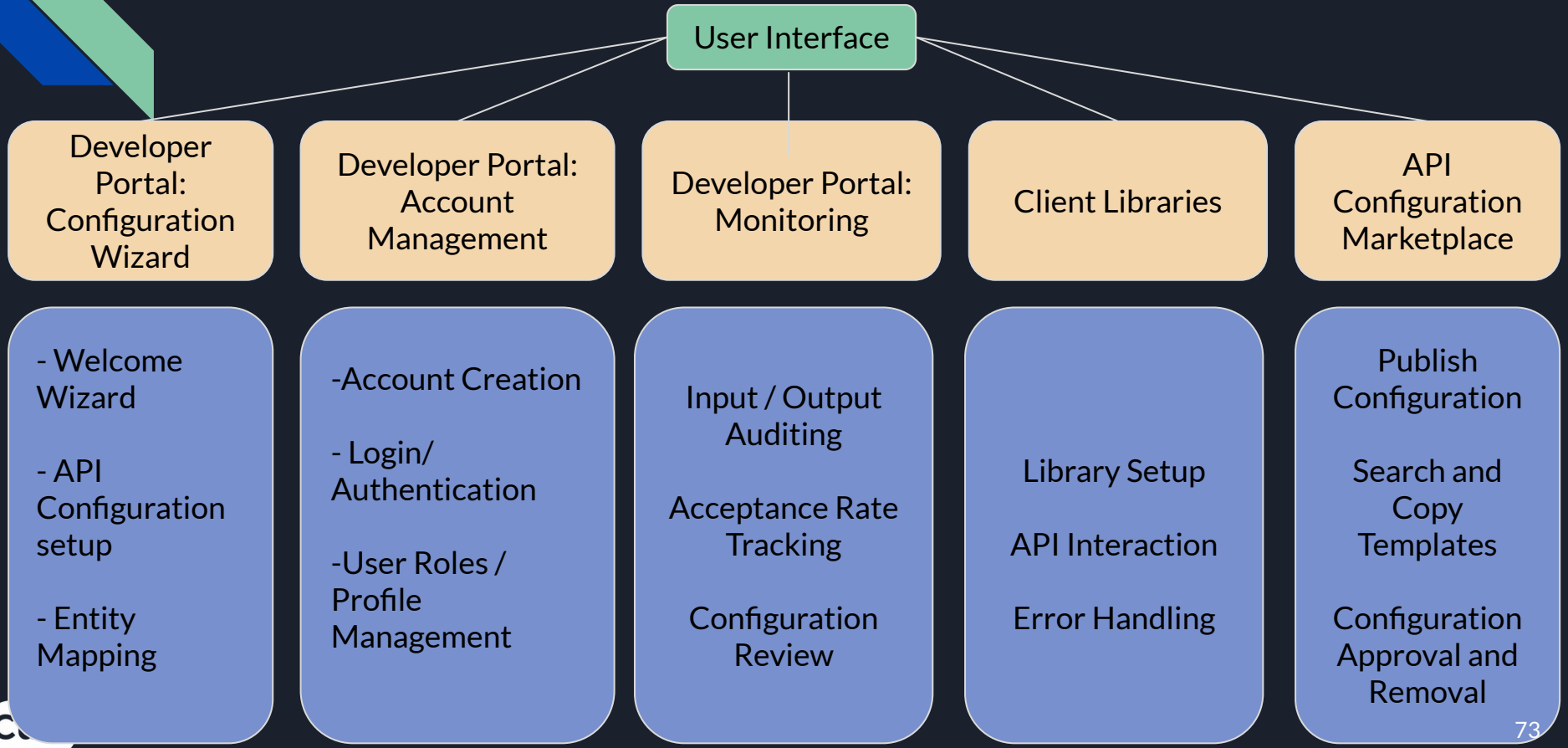
Work breakdown structure overview



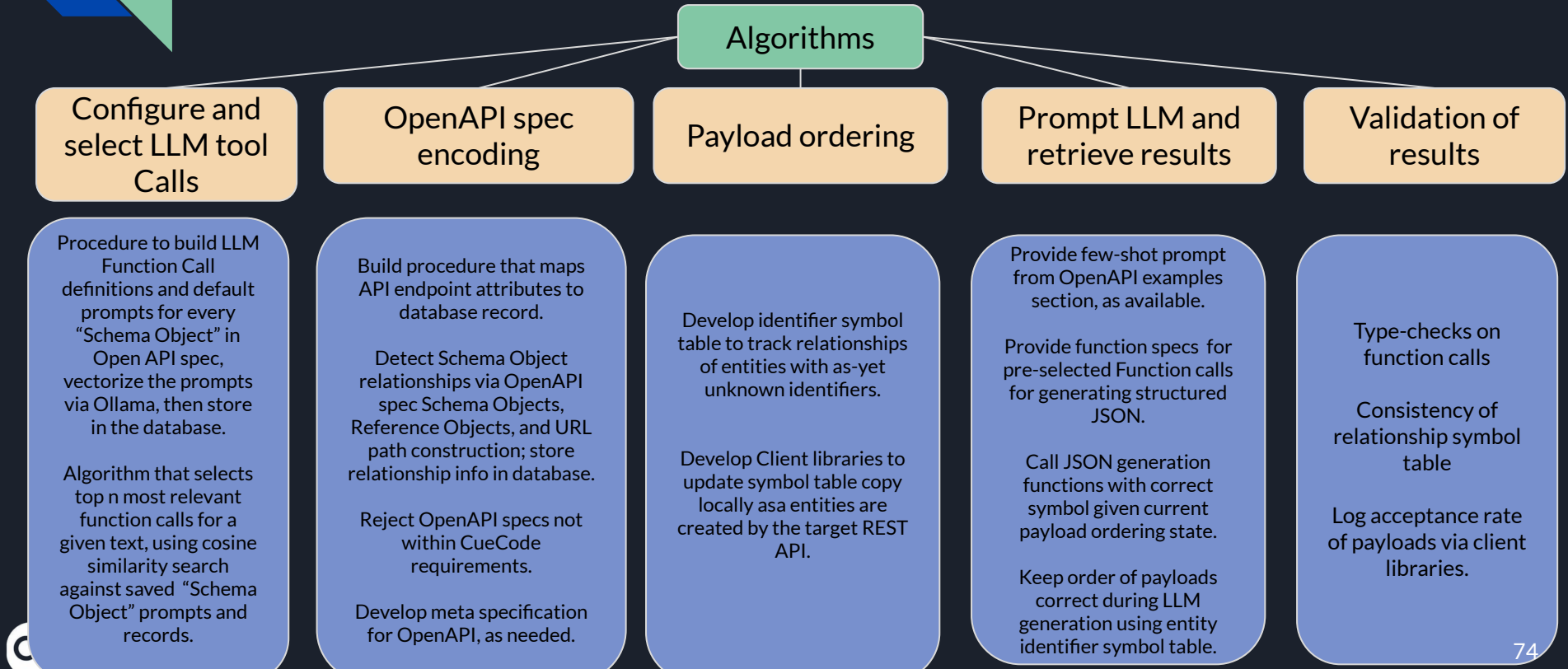
Work breakdown structure - Testing & Benchmarking



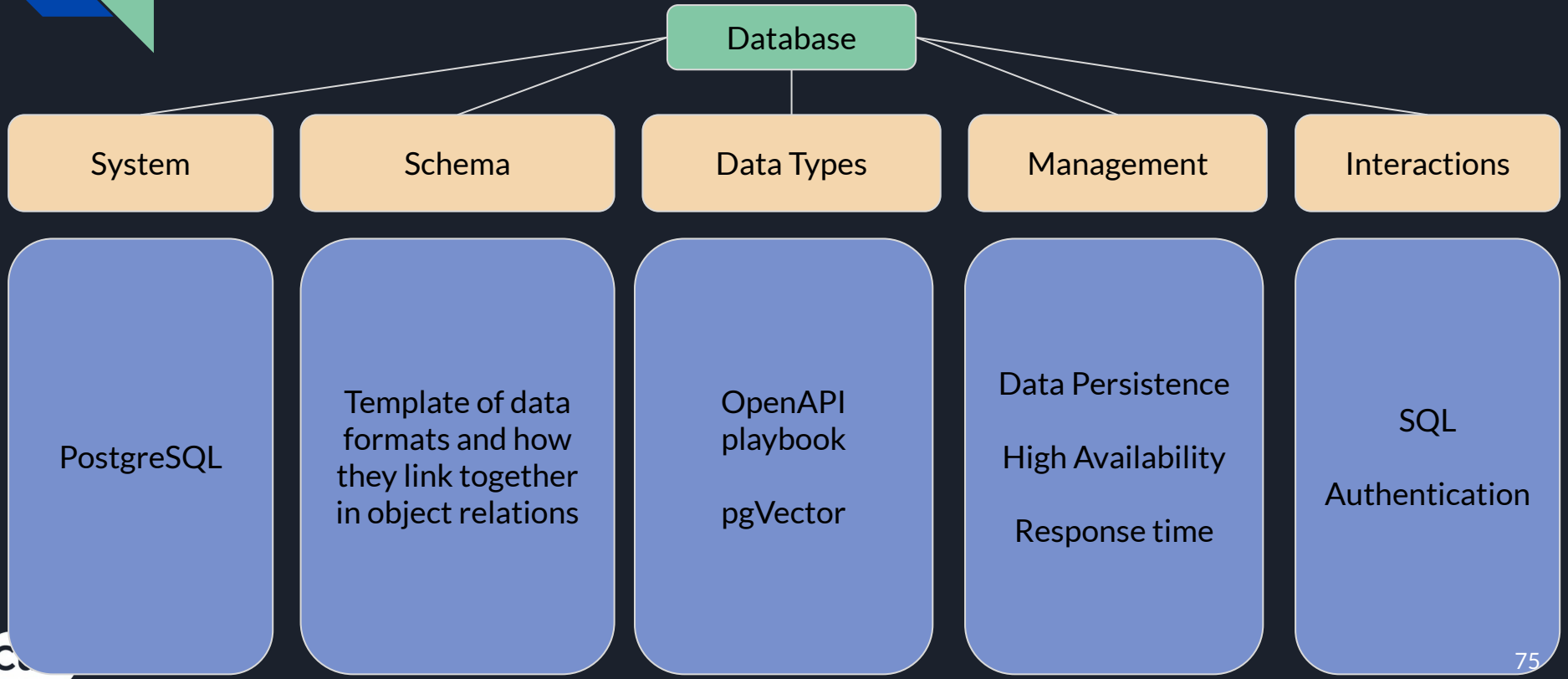
Work breakdown structure - User Interface



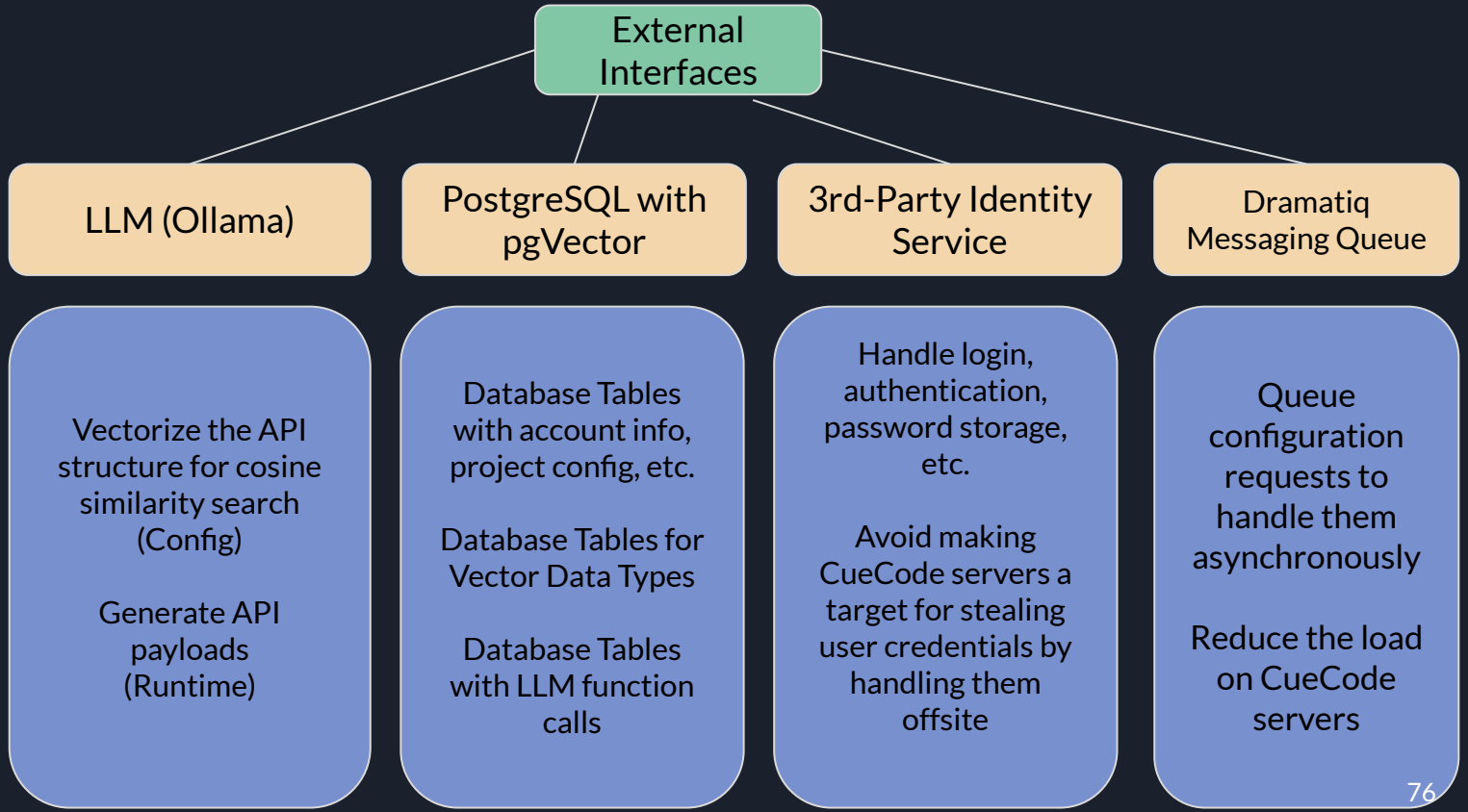
Work breakdown structure - Algorithms



Work breakdown structure - Database



Work breakdown structure - External Interfaces



CueCode

Sprint breakdown

When things need to be done

CueCode project management

- Iterative development
- A few key points of discovery:
  - LLM choice
  - Accuracy of cosine similarity search algorithm
- Critical path:
  - Dev environments
  - Database schema
  - Config algorithm
  - Runtime algorithm

CueCode

Sprint 1....

- CS Systems Group Git access
- CS Systems Group CI/CD pipeline access
- Configure local development environment:
  - Ollama in Docker
  - Postgres 17 with PgVector in Docker
- Document common development tasks and tool commands
- VS Code extensions
- Unit testing framework set up
- Continuous integration set up for running unit tests on Git merge request
- Flask app folder structure
- Python module folder structure and Pylint module boundary enforcement
- As a Developer, I need to create an account.
- As a Developer, I need to login.

CueCode

Sprint 2....

- Tech task: Database schema constructed in support of this sprint's stories.

As a Developer:

- I need to be able to log into CueCode via client library to authenticate and interact with the system.
- I need to upload an OpenAPI spec to CueCode.
- I need to receive feedback on when the configuration process is done.
- I need CueCode to validate that the generated payloads conform to my configured Swagger specification.
    - CueCode config process started.
    - Function call definition construction

CueCode

Sprint 3...

As a Developer:

- I need to be presented with configuration options required to start generating payloads for the given OpenAPI spec.
- I want to add natural language prompts to each endpoint defined in the OpenAPI specification per CueCode's instructions on how to get good results.
- I need CueCode to return zero or more suggested API calls to make, given a text input.
  - Prompt LLM with function call defs, process function calls to produce structured payloads
- I need CueCode to recognize entities described in the natural language input.

Sprint 4...

As a Developer:

- I need to send text to CueCode via client libraries
- I need CueCode to return results in the correct order in which my application would need to issue the API calls.
- I need to issue API requests in sequence, using entities that might not exist yet.
- I need to receive structured JSON representing suggested API calls
- I want CueCode to return why it could not generate results, when none can be generated.

CueCode

Sprint 5....

- Finalize anything else needed.
- Automated testing
- Development of demo examples
- As a Developer, I want to see feedback on how to make my OpenAPI spec more effective when using CueCode.

Risk management

Risks

Risks the CueCode project faces and their mitigations

Our risk coding convention:

- “O” - Operational risks
- “R” - Regulatory risks
- “T” - Technical risks

Risks - Customer, Operational, Regulatory

O1 - Unable to procure GPU Hardware for development.

- **Mitigation approach:** Control
- **Mitigation:**
  - In Spring ‘25, execute an already approved request for GPU time with the CS Systems Group

O2 - CueCode customers may overlook critical security or operational risks when generating API calls.

- **Mitigation approach:** Continue Monitoring
- **Mitigation:** Perform thorough logging, audits to provide detailed error checking tools for developers.

Probability

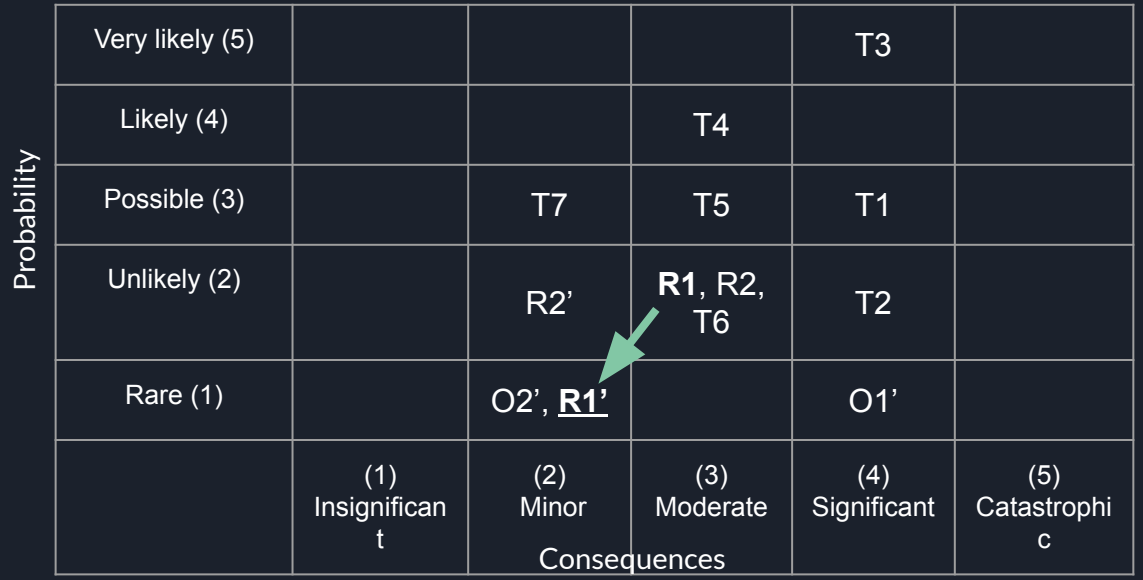
Very likely (5)				T3	
Likely (4)			T4		
Possible (3)		T7	T5	T1	**O1**
Unlikely (2)		R2’	R1, R2, T6	T2	
Rare (1)		**O2’**	**O2**	**O1’**	
	(1) Insignificant	(2) Minor	(3) Moderate	(4) Significant	(5) Catastrophic

Consequences

Risks - Customer, Operational, Regulatory

R1 - The use of API specifications might infringe on proprietary or closed API usage policies, leading to legal issues.

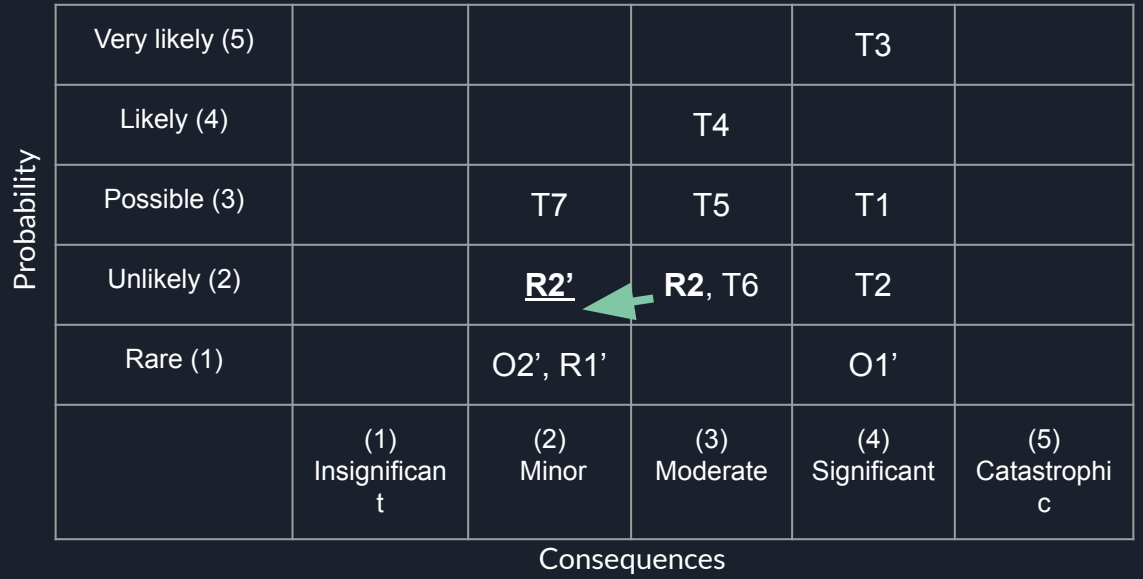
- **Mitigation approach:** Avoid
- **Mitigation:** Check downstream API usage against known limits, check with professionals about API licenses, develop and publish a platform abuse notice process for API providers to use, and stay away from violating proprietary API standards and procedures.



Risks - Customer, Operational, Regulatory

R2 - Storage of API credentials makes CueCode an enticing target for cybersecurity attacks.

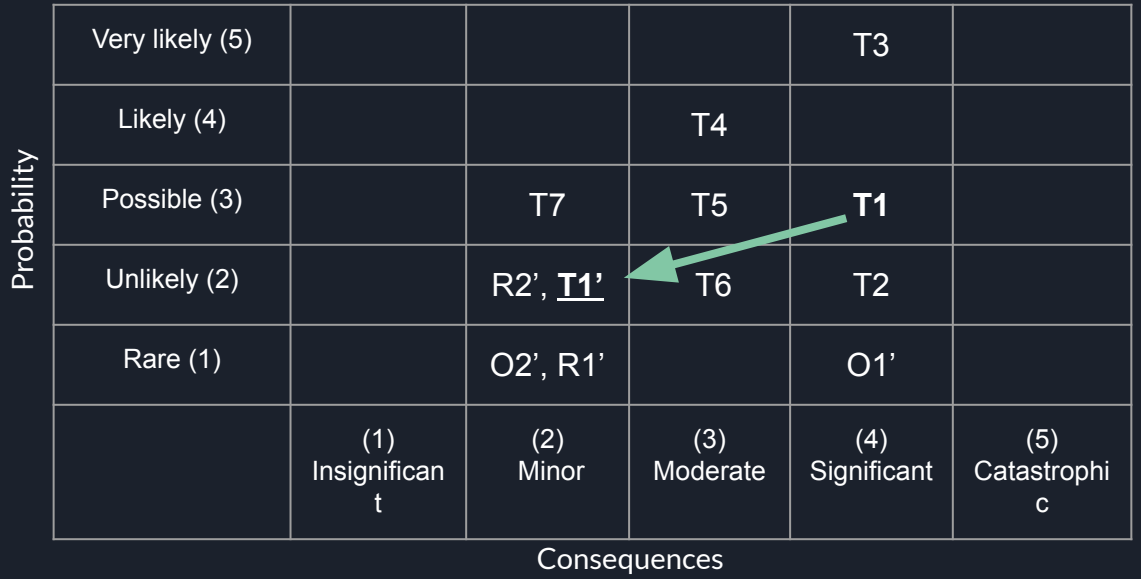
- **Mitigation approach:** Control
- **Mitigation:**
  - Legal - apply terms of use that protect CueCode in the case of data breach.
  - Technical - separate tenant credentials with care.
  - Technical - guide developers to use scoped API keys; use OAuth2 where possible for user-specific data



Risks - Technical

T1 - LLM won't generate API calls without few-shot prompt examples.

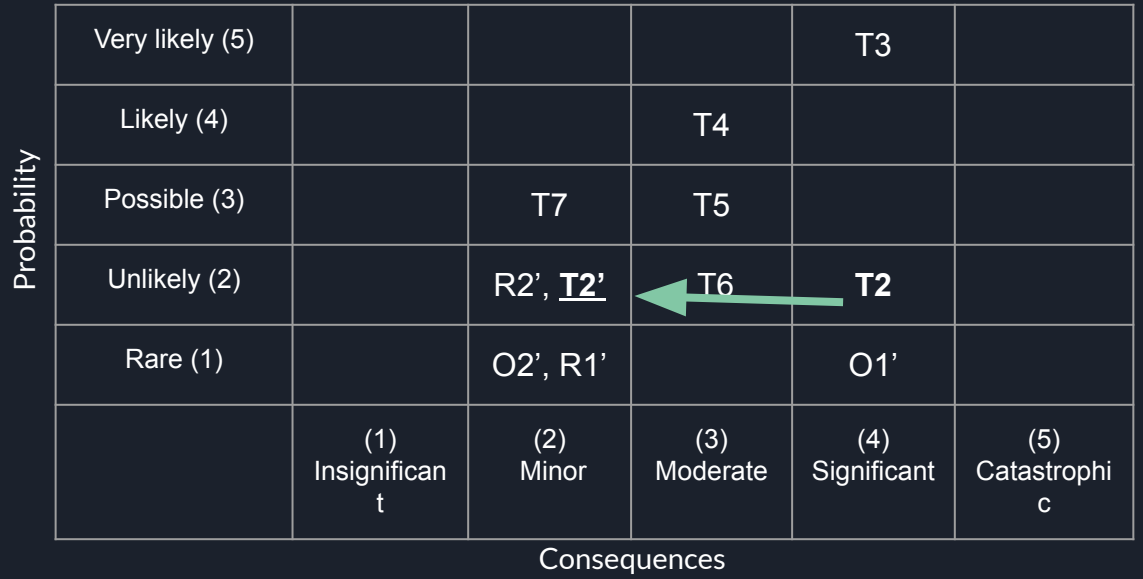
- **Mitigation approach:** Control
- **Mitigation:**
  - Validation process for prompt engineering.
  - Require that developers include a few examples in their OpenAPI specs.



Risks - Technical

T2 - LLM won't generate API calls without hundreds or thousands of examples.

- **Mitigation approach:** Continue Monitoring.
- **Mitigation:**
  - If risk is realized, then pivot to change value propositions and require backend development from the customer



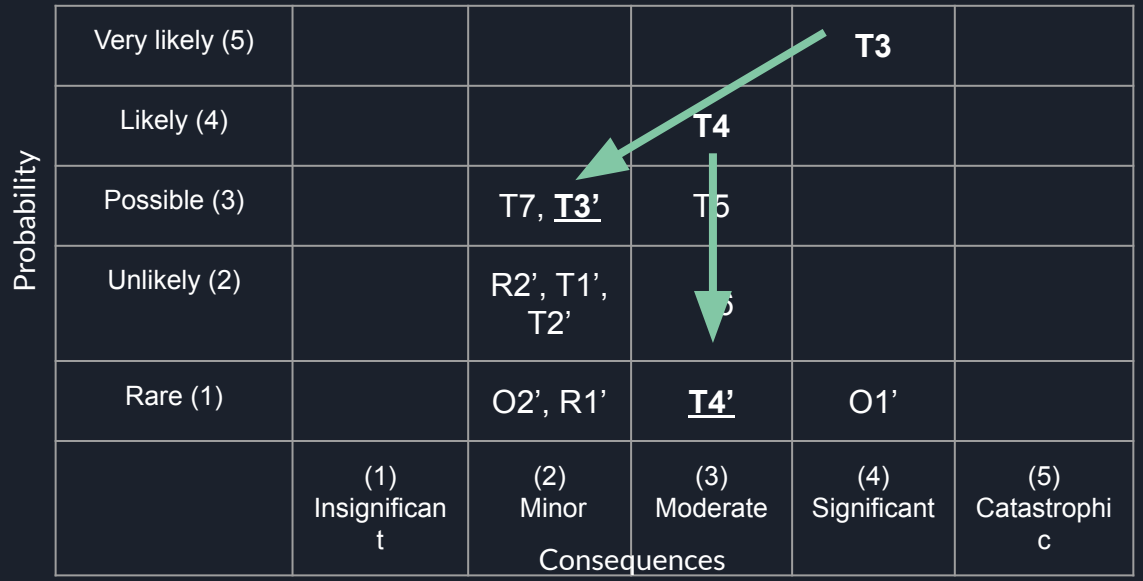
Risks - Technical

T3 - Vastness of frontend API client ecosystem precludes building CueCode client libraries for all popular languages and frameworks.

- **Mitigation approach:** Transfer
- **Mitigation:**
  - Use Swagger CodeGen for our own CueCode backend API.
  - Open-source our client library code.

T4 - Potential exposure of sensitive API information through generated API calls.

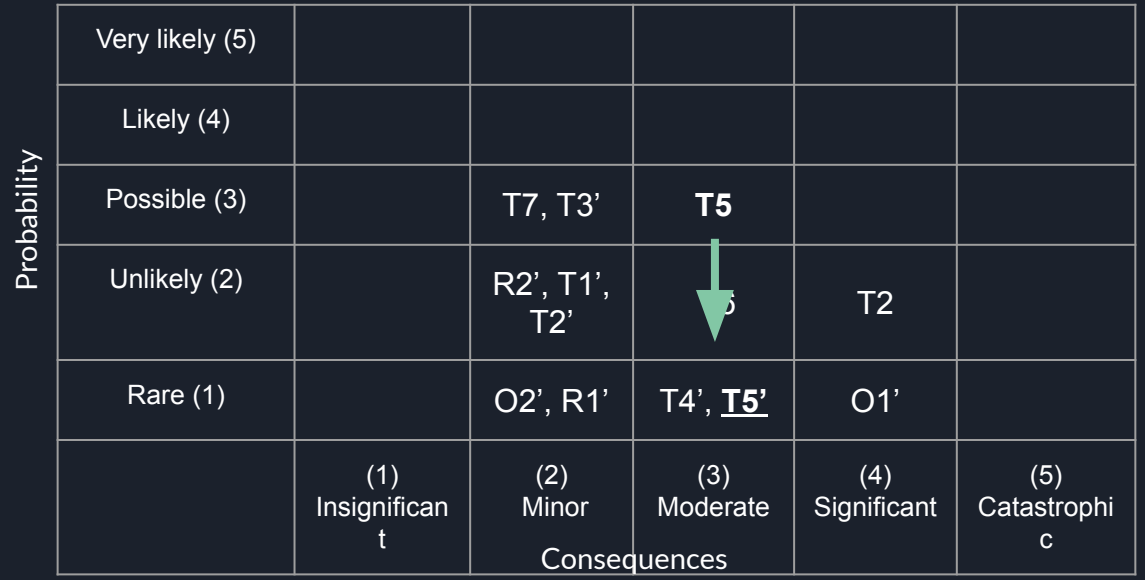
- **Mitigation approach:** Control
- **Mitigation:** separate API authentication and LLM generation concerns in the CueCode payload generation algorithm.



Risks - Technical

T5 - Obsolescence of vendor libraries and services in the greenfield AI market.

- **Mitigation approach:** Avoid
- **Mitigation:**
  - Use OLLama backend communication with the LLM, allowing swappable LLM models according to CueCode's needs.
  - Use PgVector, an extension to the FOSS PostgreSQL RDBMS, for vector storage.
  - Develop a simple Python backend without undue reliance popular AI libraries, most of which are pre-v1 and, incidentally, overfit for CueCode's purpose.



Risks - Technical

T6 - Our validation processes might find that CueCode might require a lot of time to provide accurate results, especially if generating many API payloads.

- **Mitigation approach:** Continue Monitoring
- **Mitigation:** Defer development of frontend libraries until we know whether backend processing takes so long as to require asynchronous processing, instead of request-response.

Probability

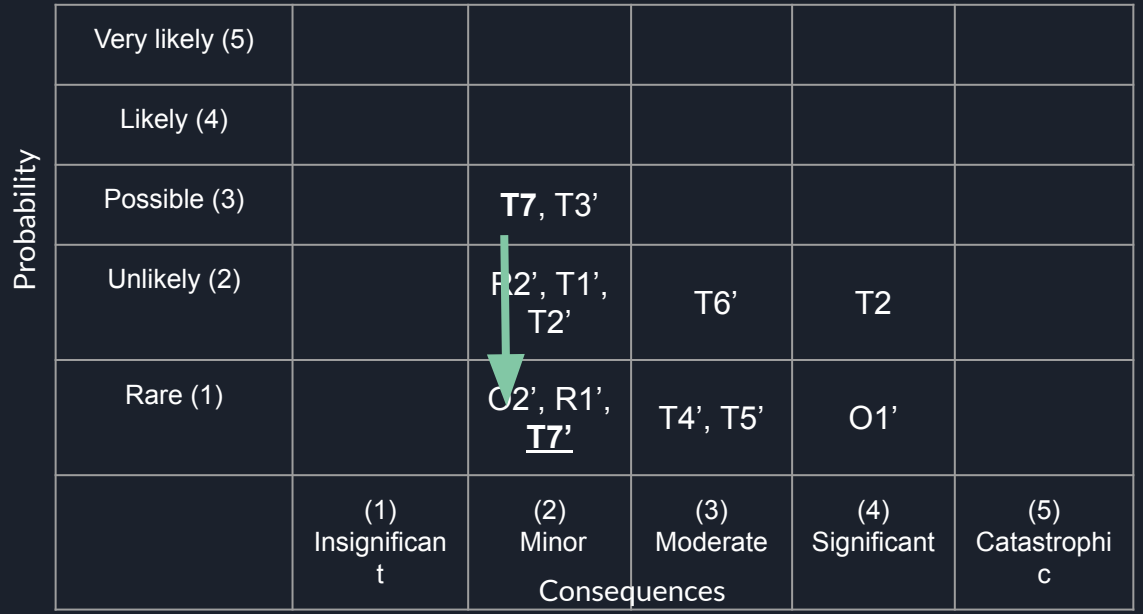
Very likely (5)					
Likely (4)					
Possible (3)		T7, T3’	**T6**		
Unlikely (2)		R2’, T1’, T2’	**T6’**	T2	
Rare (1)		O2’, R1’	T4’, T5’	O1’	
	(1) Insignificant	(2) Minor	(3) Moderate	(4) Significant	(5) Catastrophic

Consequences

Risks - Technical

T7 - Elevated demand may surpass the capacity of the system, resulting in disruptions or delays.

- **Mitigation approach:** Continue Monitoring
- **Mitigation:** As traffic increases, scalability and efficiency are ensured through:
  - Starting development with architecture that allows scaling (containerized 12-factor app)
  - Regular performance testing
  - Load balancing.



Risks - Mitigation landscape

Before

Probability						
(5)				T3		
(4)			T4			
(3)		T7	T5	T1	O1	
(2)			R1, R2, T5, T6	T2		
(1)			O2			
	(1)	**(2)**	**(3)**	**(4)**	**(5)**	

Consequences

After

Probability						
(5)						
(4)						
(3)		T3’				
(2)		R2’, T1’, T2’	T6’			
(1)		O2’, R1’, T7’	T4’, T5’	O1’		
	(1)	**(2)**	**(3)**	**(4)**	**(5)**	

Consequences

CueCode

Conclusion

- Accelerates NLP/LLM tech adoption
- Leverages existing tools and techniques
- Develops a framework for developers
- Delights users

=> so that you can Humanize Web APIs, without the headache.

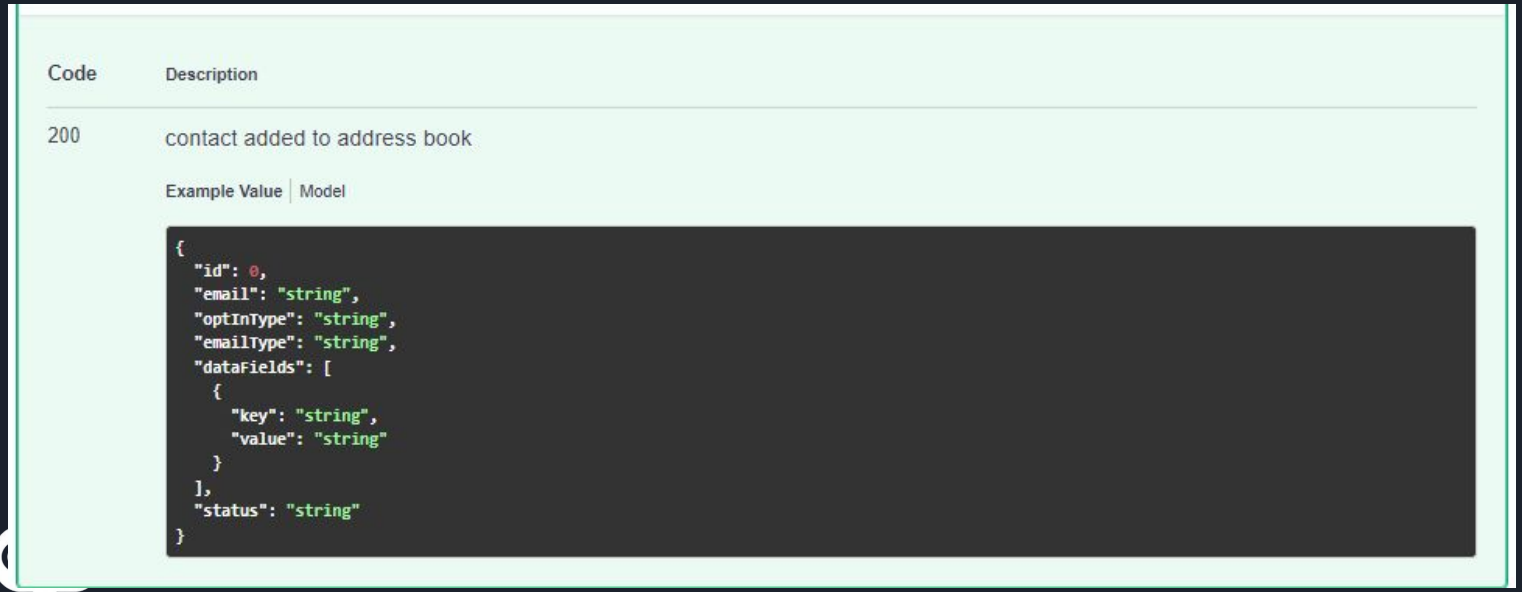
Questions and discussion

Appendix A

REST API tooling

How do we currently get Json Payloads

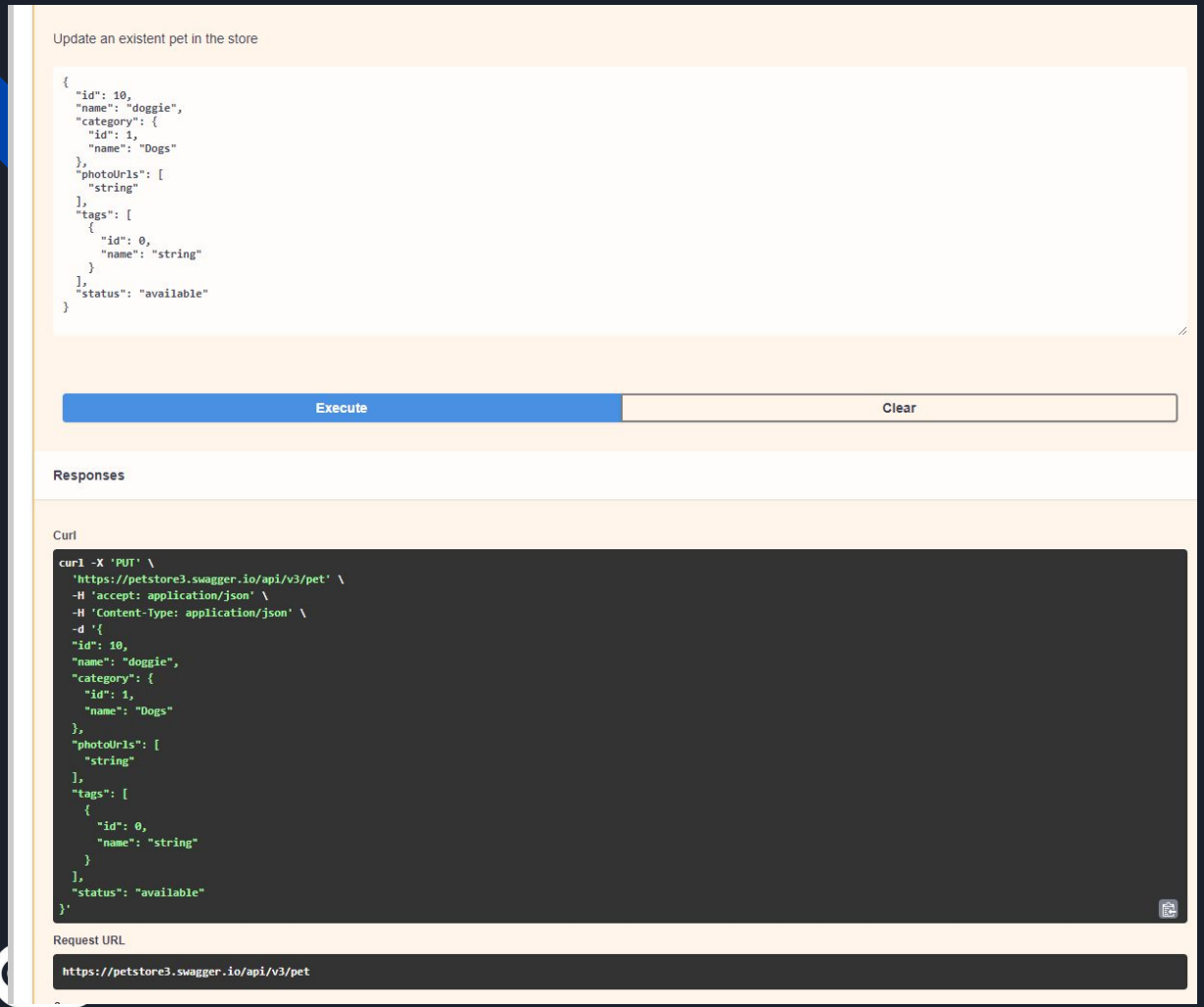
Swagger Hub, Openapi…Postman, requests libraries
And many many more



Code	Description
200	contact added to address book

Example Value | Model

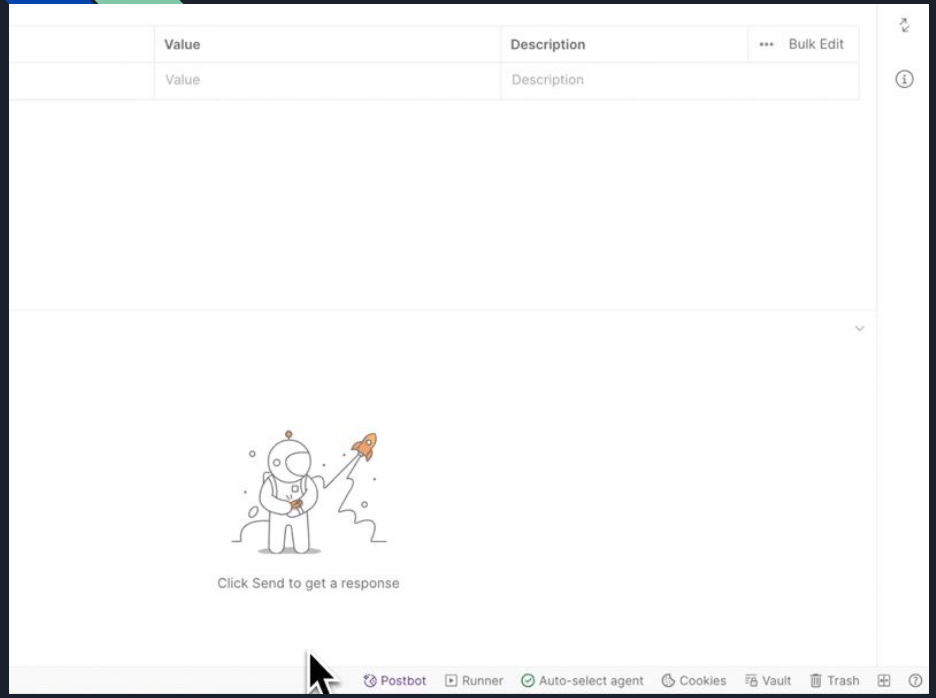
```
{
  "id": 0,
  "email": "string",
  "optInType": "string",
  "emailType": "string",
  "dataFields": [
    {
      "key": "string",
      "value": "string"
    }
  ],
  "status": "string"
}
```

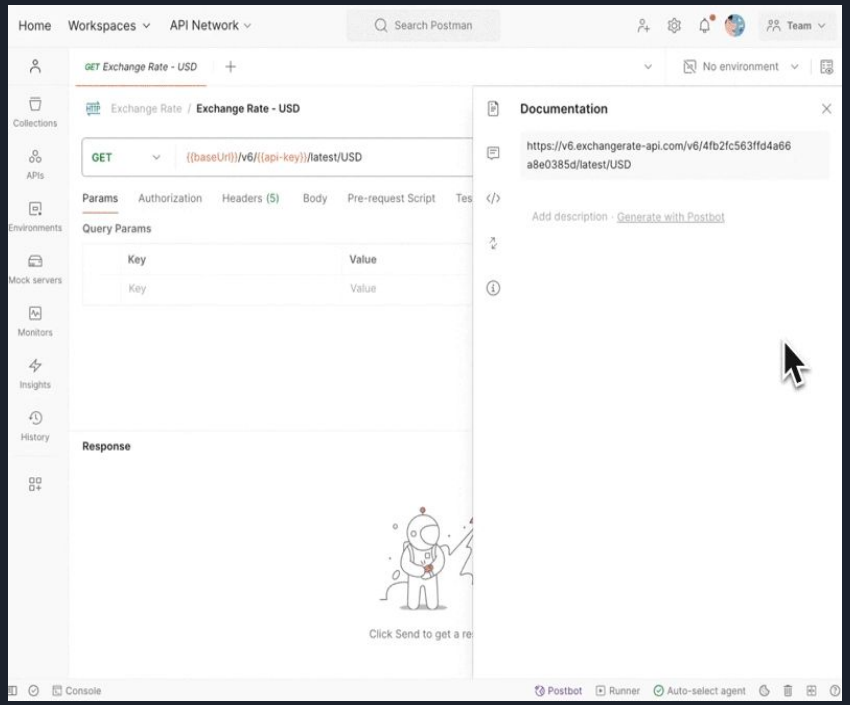


Swagger Hub Example

API TESTING
Mock Servers
API Detection

Example Postman Usage



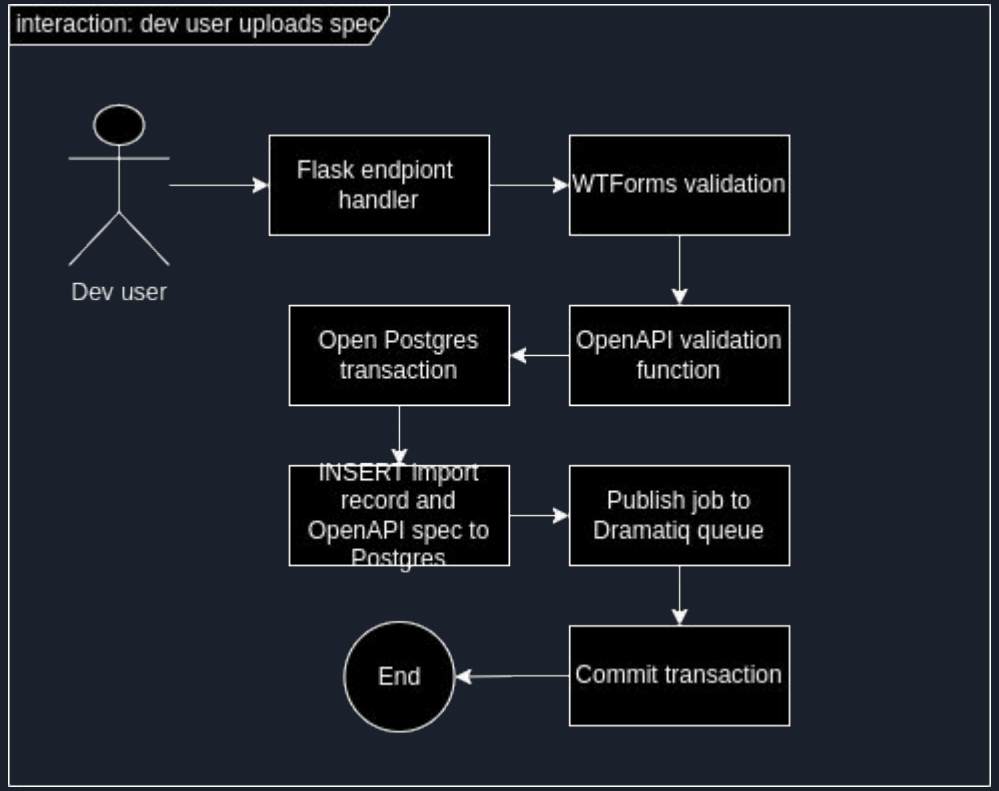


Algorithms

Appendix B

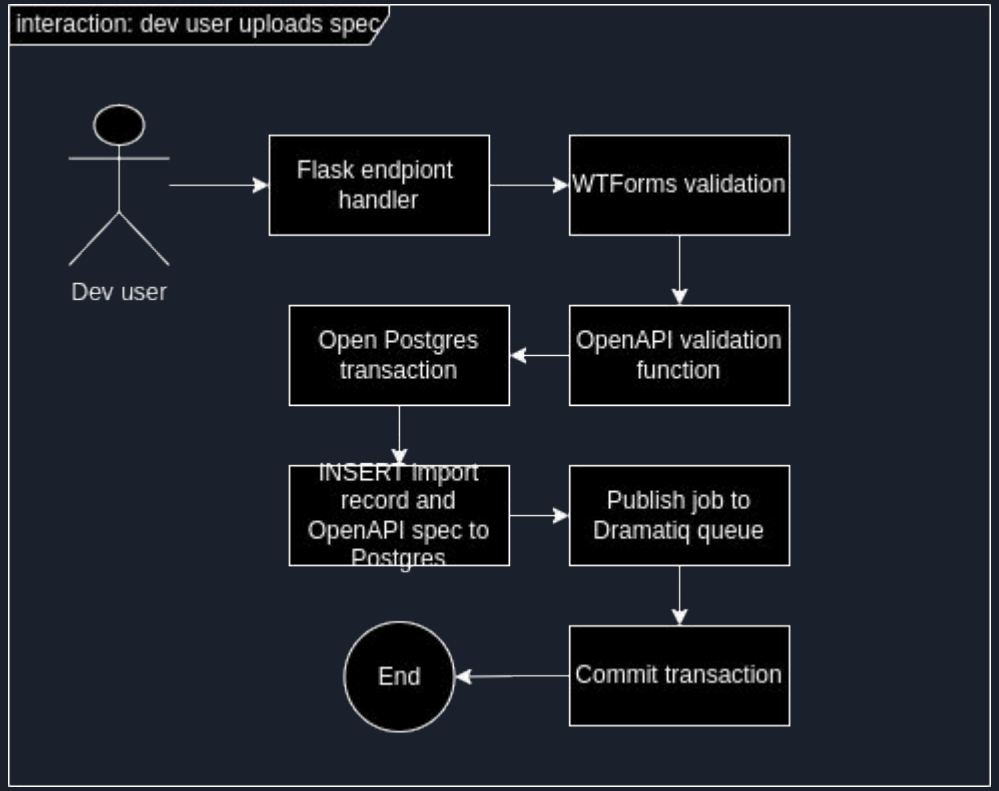
Config time - user interaction: validate and save uploaded OpenAPI spec (1 of 2)

- Receive uploaded file from the Developer Portal spec upload page.
- Perform basic form validation using WTForms
- Validate that the file provided is a valid OpenAPI spec, using the library: https://github.com/python-openapi/openapi-spec-validator
- If OpenAPI spec is valid:
  - Do all of the following with a single Postgres DB transaction, commit if successful:
    - save text of OpenAPI spec into Postgres in openapi_spec table with is_live = false, config_is_finished = false.
    - Create configuration_job record corresponding to the newly



Config time - user interaction: validate and save uploaded OpenAPI spec (2 of 2)

- If OpenAPI spec is valid:
  - Do all of the following with a single Postgres DB transaction, commit if successful:
    - save text of OpenAPI spec into Postgres in openapi_spec table with is_live = false, config_is_finished = false.
    - Create configuration_job record corresponding to the newly uploaded OpenAPI spec.
  - Save job in Dramatiq for configuring OpenAPI specification specifying the OpenAPI spec by ID.



Config time - asynchronous job (Dramatiq): Encode OpenAPI spec for CueCode and set defaults.

- Extract all Server Objects from OpenAPI spec and save as openapi_server.
- Extract all Schema Objects from OpenAPI spec and save each as an openapi_entitity
  - Set noun_prompt as the name Schema Object reference name, for a default. If present, use x-cuecode-prompt.
    - https://swagger.io/specification/#specification-extensions
  - https://swagger.io/specification/#schema-object
- Extract all Path Objects from OpenAPI spec and save each as an openapi_endpoint.
  - Use x-cuecode-prompt if present; otherwise, use Path summary and description from
    - https://swagger.io/specification/#specification-extensions
  - OpenAPI spec are concatenated to form openapi_endpoint.selection_prompt.

CueCode

Config time - asynchronous job (Dramatiq): Encode OpenAPI spec for CueCode and set defaults.

- **Detect Schema Object relationships via OpenAPI spec Schema Objects, Reference Objects, and URL path construction; store relationship info in database.**
  - Find relationships in URLs with ref objects (e.g., “$ref: CustomerID”)
  - Construct openapi_entity relationship graph using OpenAPI spec; save to database in openapi_entity and openapi_entity_dependency tables.
- Set job status to “pending review”.

Config time - user interaction: Validate edits from Developer during preview time

- Triggered when the user attempts to set the spec config job status to Finalizing.
- Validation: Confirm all prompts are filled in before allowing the Developer to pass the Dramatiq job to the next stage.
- Once all prompts filled are in, allow Developer move Dramatiq job to stage “Final Vectorization”.

Config time - asynchronous job (Dramatiq async job library)

- Call Ollama to vectorize all prompts specified in the Dramatiq payload; save to the database. Use one transaction.
- Procedure to build LLM Function Call definitions Path in Open API spec.
  - Ollama docs: https://ollama.com/blog/tool-support
  - Starting point for OpenAPI spec encoding: https://github.com/openai/openai-cookbook/blob/main/examples/Function_calling_with_an_OpenAPI_spec.ipynb
  - Steps:
    - Replace all references (Schema Object or otherwise) with actual values.
    - Adjust OpenAI Cookbook example to encode a dictionary in Ollama format
      - Format as needed here: https://github.com/ollama/ollama-python/tree/main
    - Save as JSONB in the Postgres database, in the corresponding openapi_path record.
- **Update the openapi_spec.config_is_finished value to “true”.**
- Move the Dramatiq job to “Completed” status.

CueCode

Runtime: Algorithm that selects top N most relevant endpoints for a given sentence

Data structures:

```
class Endpoint {
        Name // maps to db column
        Others…. But they aren't relevant in discussion of this algorithm.
        ParentName // for mapping dependencies
}

class EndpointRecommendation {
        sentence: ParsedSentence
        endpoints : List of Endpoint
}

class ParsedSentence {
        Sentence text
        Spacy parse
}

class ParsedText {
        Original text
        Sentences: List of ParsedSentence // expected in order of original text
}
```

CueCode

Runtime: Algorithm that selects top N most relevant endpoints for a given sentence

Output: list of EndpointRecommendation objects.

Notes:

- There are three kinds of prompts used for each endpoint, each with a separate embedding columns, which are accessed using SQL queries via the PgVector extension:
  - A single openapi_path.selection_prompt, whose embedding (field selection_prompt_embedding) is used in a cosine similarity search when determining which OpenAPI Paths to supply to the LLM as Tool Call specifications.
    - Note: We supply Paths as Tool Call specifications to avoid the difficult task of having the LLM generate structured data.
  - Many openapi_verb_http_equiv, whose embedding (field verb_lemma_embedding) is used in a cosine similarity search when determining the action being taken on an entity.
  - The prompt we give the LLM in the function call definition (not stored in vectorized format)

Runtime: Algorithm that selects top N most relevant endpoints for a given sentence

Steps of the algorithm:

- Split into sentences.
  - SpaCy sentencizer https://spacy.io/api/sentencizer
- For each sentence:
  - Use spacy to identify “noun chunks” and “lemmatized” verbs in each sentence.
    - https://spacy.io/usage/linguistic-features#lemmatization
    - https://spacy.io/usage/linguistic-features#noun-chunks
  - For each sentence:
    - Embed the noun chunk for the sentence with Ollama and save to a data structure for the current sentence.
      - https://github.com/ollama/ollama-python/tree/main?tab=readme-ov-file#embed
    - Embed the noun chunk for the sentence with Ollama and save to a data structure for the current sentence.
      - https://github.com/ollama/ollama-python/tree/main?tab=readme-ov-file#embed

Runtime: Algorithm that selects top N most relevant endpoints for a given sentence

- Use a parameterized SQL query to
  - Find up to M verb_lemma records matching the user intent, using cosine similarity between an embedding of the verb lemma - identified by spaCy - and the verb_lemma.verb_lemma_embedding column.
  - Find openapi_ver_http_equiv records that correspond to the top M verbs found.
  - Find openapi_entity records that match the openapi_entity.noun_prompt_embedding with an embedding of the noun chuck.
  - Match the openapi_entity records with openapi_ver_http_equiv records via a JOIN.
  - JOIN to openapi_endpoint, whose records are also returned based on a cosine similarity search on the selection_prompt_embedding field and the sentence.
- Get the top N results from the DB, which will include the names of the endpoints.
- Return list of EndpointRecommendation from subroutine.
- Resources:
  - https://www.machinelearningplus.com/nlp/cosine-similarity/
  - https://github.com/pgvector/pgvector?tab=readme-ov-file#querying

Other algorithms

- Build function call definition from Schema Object
- Validate OpenAPI specs for compatibility with CueCode
- Develop identifier symbol table to track relationships of entities with as-yet unknown identifiers.
- Develop Client libraries to update symbol table copy locally as entities are created by the target REST API.
- Provide few-shot prompt from OpenAPI examples section, as available.
- Provide function specs for pre-selected Function calls for generating structured JSON.
- Call JSON generation functions with correct symbol given current payload ordering state.
- Keep order of payloads correct during LLM generation using entity identifier symbol table.
- Type-checks on LLM function calls
- Consistency of relationship symbol table
- Log acceptance rate of payloads via client libraries.

CueCode

User stories

User Stories: Initiative core functionality

- (Epic: client library integration)
  - As a Developer, I
    - Need to send text to CueCode via client libraries
    - Need to receive structured JSON representing suggested API calls
    - Need to issue API requests in sequence, using entities that might not exist yet.
- (Epic: account creation)
- (Epic: login)
- (Epic: upload OpenAPI/Swagger specification for an API)
  - As a Developer, I
    - Need to upload an OpenAPI spec to CueCode
    - Need to receive feedback on when the configuration process is done.
    - Need to be presented with configuration options required to start generating payloads for the given OpenAPI spec.
    - Need to add natural language prompts to each endpoint defined in the OpenAPI specification per CueCode's instructions on how to get good results.
    - Want to see feedback on how to make my OpenAPI spec more effective when using CueCode.
- (Epic: entity detection)
  - As a Developer, I
    - Need CueCode to recognize entities described in the natural language input.
    - Want CueCode to perform lookups when entities do not appear in the natural language as they do in my system (e.g., first name, not unique identifier is used in text)
- (Epic: payload generation)
  - As a Developer, I
    - Need CueCode to return zero or more suggested API calls to make, given a text input.
    - Need CueCode to return results in the correct order in which my application would need to issue the API calls.
    - Need CueCode to validate that the generated payloads conform to my configured Swagger specification.
    - Want CueCode to return why it could not generate results, when none can be generated.

CueCode

User Stories: As A Developer

- As a developer, I would like to be able to return in the correct order in which my application would need to issue the API calls to avoid errors in execution.
- As a developer, I would like CueCode to return clear explanation of why no results were generated such as missing parameters, unrecognizable entities etc.
- As a developer, I would like to address any issues with input or configurations.
- As a developer. I need CueCode to return zero or more suggested API calls, so that I can integrate the recommended calls into my application.
- As a deve;oper, I need to be able to upload an OpenAPI spec so that the system can understand the API structure and generate payloads.
- As a developer, I need to be able to log into CueCode via client library to authenticate and interact with the system.
- As a developer, I need to be able to create an account through CueCode's client library.
- As a developer, I need to receive structured JSON representing suggested API calls to understand the recommended sequence of actions my application should take.

CueCode

References

References 1 of 7

About continuous integration with GitHub Actions. (n.d.). GitHub Docs. Retrieved October 22, 2024, from https://docs.github.com/en/actions/about-github-actions/about-continuous-integration-with-github-actions [1]

About Git. (n.d.). GitHub Docs. Retrieved October 22, 2024, from https://docs.github.com/en/get-started/using-git/about-git [2]

Against LLM maximalism · Explosion. (2023, May 18). https://explosion.ai/blog/explosion.ai [3]

Baker, S. (2024). *Paragonsean/ChatBotAsync* [Python]. https://github.com/paragonsean/ChatBotAsync [5]

Cloud Natural Language. (n.d.). Google Cloud. Retrieved September 26, 2024, from https://cloud.google.com/natural-language [6]

Dickson-Mwendia. (2024, August 1). *Tutorial: Register a Python web app with the Microsoft identity platform - Microsoft identity platform*. https://learn.microsoft.com/en-us/entra/identity-platform/tutorial-web-app-python-register-app

Evaluation | Genkit. (n.d.). Firebase. Retrieved September 14, 2024, from https://firebase.google.com/docs/genkit/evaluation [7]

Firebase Genkit. (n.d.). Retrieved September 14, 2024, from https://firebase.google.com/docs/genkit [8]

Function Calling. (n.d.). Retrieved September 14, 2024, from https://platform.openai.com/docs/guides/function-calling [9]

Jesse, A. (2017, April 18). *Grok the GIL: How to write fast and thread-safe Python | Opensource.com*. https://opensource.com/article/17/4/grok-gil [10]

CueCode

References 2 of 7

Llama-recipes/recipes/quickstart/Prompt_Engineering_with_Llama_3.ipynb at main · meta-llama/llama-recipes. (n.d.). GitHub. Retrieved November 12, 2024, from https://github.com/meta-llama/llama-recipes/blob/main/recipes/quickstart/Prompt_Engineering_with_Llama_3.ipynb [11]

Lorica, B. (2024, January 25). *Expanding AI Horizons: The Rise of Function Calling in LLMs*. Gradient Flow. https://gradientflow.com/expanding-ai-horizons-the-rise-of-function-calling-in-llms/ [12]

Mark Needham (Director). (2023, July 26). *Returning consistent/valid JSON with OpenAI/GPT* [Video recording]. https://www.youtube.com/watch?v=IJJkBaO15Po [13]

Merritt, R. (2023, November 15). *What Is Retrieval-Augmented Generation aka RAG?* NVIDIA Blog. https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/ [14]

Microsoft/prompt-engine. (2024). [TypeScript]. Microsoft. https://github.com/microsoft/prompt-engine (Original work published 2022) [15]

References 3 of 7

Natural Language Processing [NLP] Market Size | Growth, 2032. (n.d.). Retrieved September 14, 2024, from https://www.fortunebusinessinsights.com/industry-reports/natural-language-processing-nlp-market-101933 [16]

Ollama/ollama-python. (2024). [Python]. Ollama. https://github.com/ollama/ollama-python (Original work published 2023) [17]

OpenAI Platform. (n.d.-a). Retrieved September 10, 2024, from https://platform.openai.com [18]

OpenAI Platform. (n.d.-b). Retrieved October 24, 2024, from https://platform.openai.com [19]

Openai-cookbook/examples/Function_calling_with_an_OpenAPI_spec.ipynb at main · openai/openai-cookbook. (n.d.). GitHub. Retrieved November 20, 2024, from https://github.com/openai/openai-cookbook/blob/main/examples/Function_calling_with_an_OpenAPI_spec.ipynb [20]

References 4 of 7

OpenAPI Specification—Version 3.1.0 | Swagger. (n.d.). Retrieved September 10, 2024, from https://swagger.io/specification/ [21]

OpenAPITools/openapi-generator. (2024). [Java]. OpenAPI Tools. https://github.com/OpenAPITools/openapi-generator (Original work published 2018) [22]

Pgvector/pgvector. (2024). [C]. pgvector. https://github.com/pgvector/pgvector (Original work published 2021)

piembsystech. (2023, October 2). *Dynamic Binding in Python Language*. PiEmbSysTech. https://piembsystech.com/dynamic-binding-in-python-language/ [23]

Prabhakaran, S. (2018, October 22). Cosine Similarity - Understanding the math and how it works? (With python). *Machine Learning Plus*. https://www.machinelearningplus.com/nlp/cosine-similarity/ [24]

Rao, P. (2024, January 24). *Turbo-charge your spaCy NLP pipeline*. Medium. https://towardsdatascience.com/turbo-charge-your-spacy-nlp-pipeline-551435b664ad [25]

References 5 of 7

Sentencizer · spaCy API Documentation. (n.d.). Sentencizer. Retrieved November 20, 2024, from https://spacy.io/api/sentencizer [26]

SpaCy · Industrial-strength Natural Language Processing in Python. (n.d.). Retrieved September 26, 2024, from https://spacy.io/ [27]

spaCy Linguistic Features—Lemmatization. (n.d.). Linguistic Features. Retrieved November 20, 2024, from https://spacy.io/usage/linguistic-features#lemmatization [28]

spaCy Linguistic Features—Noun Chunks. (n.d.). Linguistic Features. Retrieved November 20, 2024, from https://spacy.io/usage/linguistic-features#noun-chunks [29]

Stanfordnlp/dspy. (2024). [Python]. Stanford NLP. https://github.com/stanfordnlp/dspy (Original work published 2023) [30]

Su, Y., Awadallah, A. H., Khabsa, M., Pantel, P., Gamon, M., & Encarnacion, M. (2017). Building Natural Language Interfaces to Web APIs. *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 177–186. https://doi.org/10.1145/3132847.3133009 [31]

References 6 of 7

Tool/function calling | LangChain. (n.d.). Retrieved September 14, 2024, from https://python.langchain.com/v0.1/docs/modules/model_io/chat/function_calling/ [32]

Tutorial: ChatGPT Over Your Data. (2023, February 6). LangChain Blog. https://blog.langchain.dev/tutorial-chatgpt-over-your-data/ [33]

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2023). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models* (arXiv:2201.11903). arXiv. http://arxiv.org/abs/2201.11903 [34]

What Is NLP (Natural Language Processing)? | IBM. (2021, September 23). https://www.ibm.com/topics/natural-language-processing [35]

Why Visual Studio Code? (n.d.). Retrieved October 22, 2024, from https://code.visualstudio.com/docs/editor/whyvscode [37]

References 7 of 7

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). *ReAct: Synergizing Reasoning and Acting in Language Models* (arXiv:2210.03629). arXiv. http://arxiv.org/abs/2210.03629

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

"26.1. SQL Dump." PostgreSQL Documentation, 11 May 2023, www.postgresql.org/docs/current/backup-dump.html.

CueCode