CueCode

Team Red CS410W project

Elevator Pitch

CueCode lets a Web application generate API calls from natural language with minutes of development time. “I booked an appointment for Patricia Davis for Thursday at 2pm” can become an API call to your appointment booking backend with little additional programming effort.

A good API specification and a few key questions are all the model needs to start generating API requests.

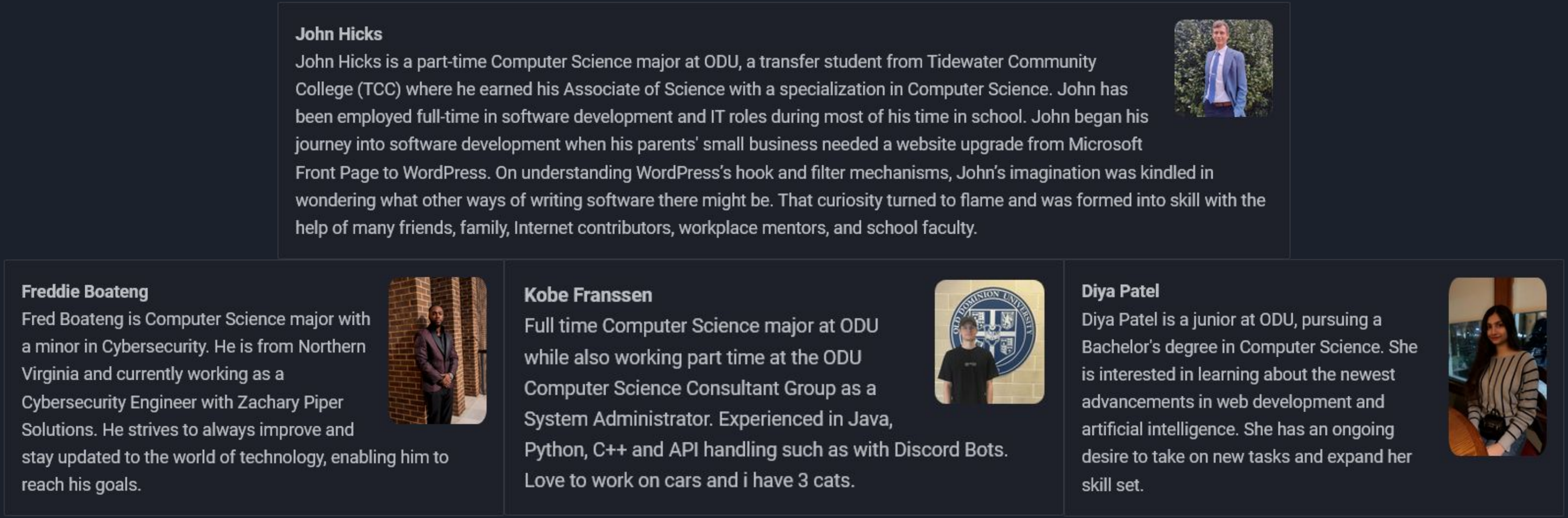
This allows rapid development of natural language processing features typical of those created during the Generative AI boom, without having to take humans or business rules out of the loop. CueCode can add AI features to your app without any backend code changes or specialized NLP or large language model (LLM) skills.

CueCode is easy to integrate with existing services, making a better experience for users and developers alike.

Table of Contents

- Team Bios
- Table of Contents
- Elevator Pitch
- The Societal Problem
- Solution
- Development Tools
- Major Functional Components
- References
- Appendix

Team Bios



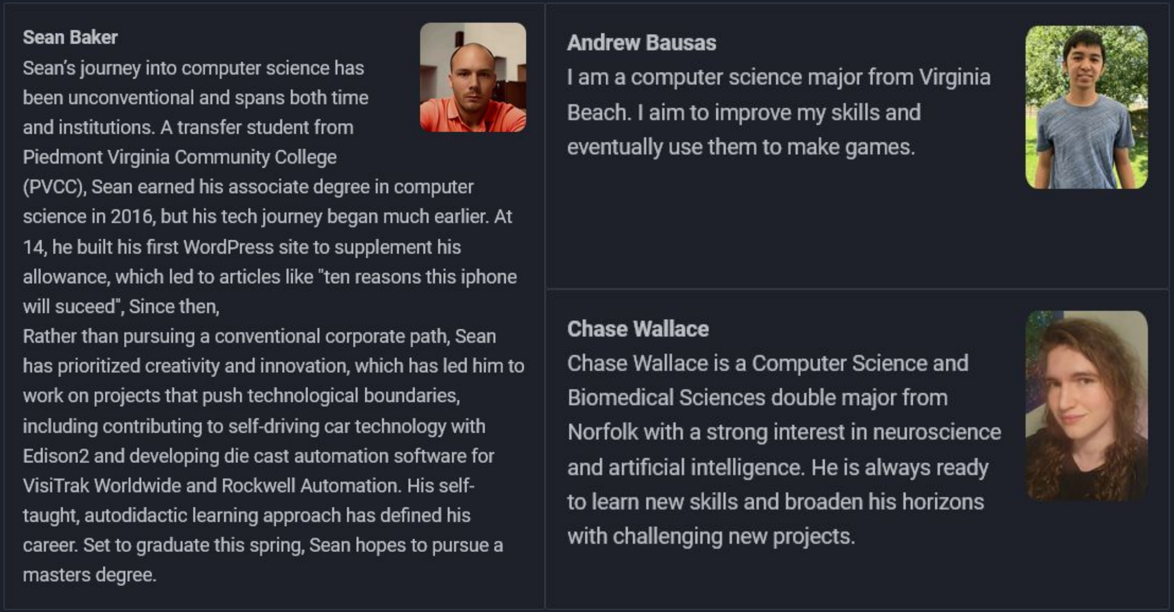
John Hicks
John Hicks is a part-time Computer Science major at ODU, a transfer student from Tidewater Community College (TCC) where he earned his Associate of Science with a specialization in Computer Science. John has been employed full-time in software development and IT roles during most of his time in school. John began his journey into software development when his parents' small business needed a website upgrade from Microsoft Front Page to WordPress. On understanding WordPress's hook and filter mechanisms, 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
Fred Boateng is 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.

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

Diya Patel
Diya Patel is 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.

Team Bios



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 his allowance, which led to articles like "ten reasons this iphone will suceed", 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 technology 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.

Andrew Bausas

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

Chase Wallace

Chase Wallace is 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.

The Societal Problem

- User interfaces don’t speak the user’s language, but users rely on apps to make things happen.
- Things happen in Web apps through Web APIs.
- Developers are motivated to add Natural Language Processing (NLP) features to their apps, but doing so is painstaking.
- We need a way to turn natural language into Web Application Programming Interface (API) calls.
    - For example, if a client service representative were to provide input to an application in natural language, “I called Patricia Davis and rescheduled her appointment from August 1st to August 16th.”, a Web API call like the following would be generated:
    - `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}}}}`

2.1 Problem Statement

No framework exists for making Natural Language to API-call generation simple for fullstack and Web developers.

Existing approaches:

- Microsoft created a paper describing their approach to using natural language to operate on the Microsoft Graph API [CITE MS]. But that is just for the Graph API.
- Zafin claims to have built a system that does uniquely well at identifying which API endpoint to call due to an embedding strategy for API calls [CITE Zafin]. The solution focused on chatbot integration more than data entry.
- LLM Function Calling is promising, but it requires LLM prompt engineering and backend programming to turn natural language to API calls, vs. the normal chatbot use-case.

=> Demand for API-call generation, but no simple, operationalized, and risk-aware tooling for it.

2.2 Problem Characteristics - use of API specs

- To solve the above problems, we must commoditize the process of turning natural language into API calls against a large number of existing existing Web APIs.
    - (Reach for the pre-made tool.)
- 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.

2.2 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 approaches that perform every LLM Function Call that the LLM requests; this is high-risk.

2.2 Problem Characteristics - NLP/LLM challenges

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

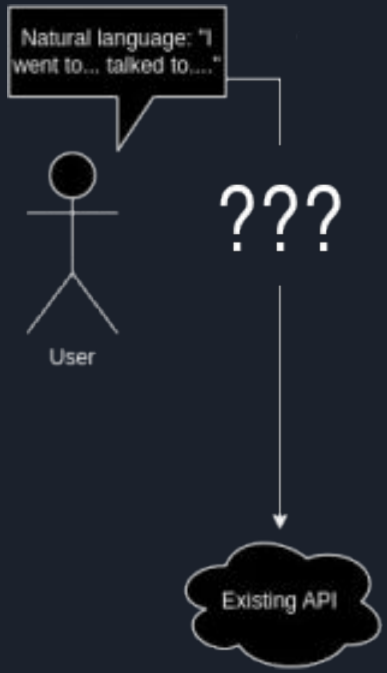
- Limiting Responses to fit an API Structure Is Difficult
- Lack of Understanding of Entity Relationships
- Absence of a Consistent Framework for Web Developers

2.3 Current Process Flow

A solution for generating API calls would ideally address all of these points.

- Design the interface between the customer's application and the API call generation code.
- Encode the Web API structure for validation and generation. Options:
  - In Langchain, build Python classes in [9]
  - OpenAI, use schema specification [CITE] and hope for the best.
- Tag entities and their relationships in the natural language input.
- For JSON, prompt the LLM to use a certain JSON format. Verify output is in JSON format (LangChain [9], Guidance AI [6])
- 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 [CITE].
- Once an API call is generated, confirm its structure (JSON or otherwise) conforms to the spec.
- Confirm that the sequence of data manipulations is consistent with the new/modified entities' relationships.
- 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.

No single application or framework on the market addresses all of these concerns..



3 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.

3.1 Solution Statement

What that means:

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

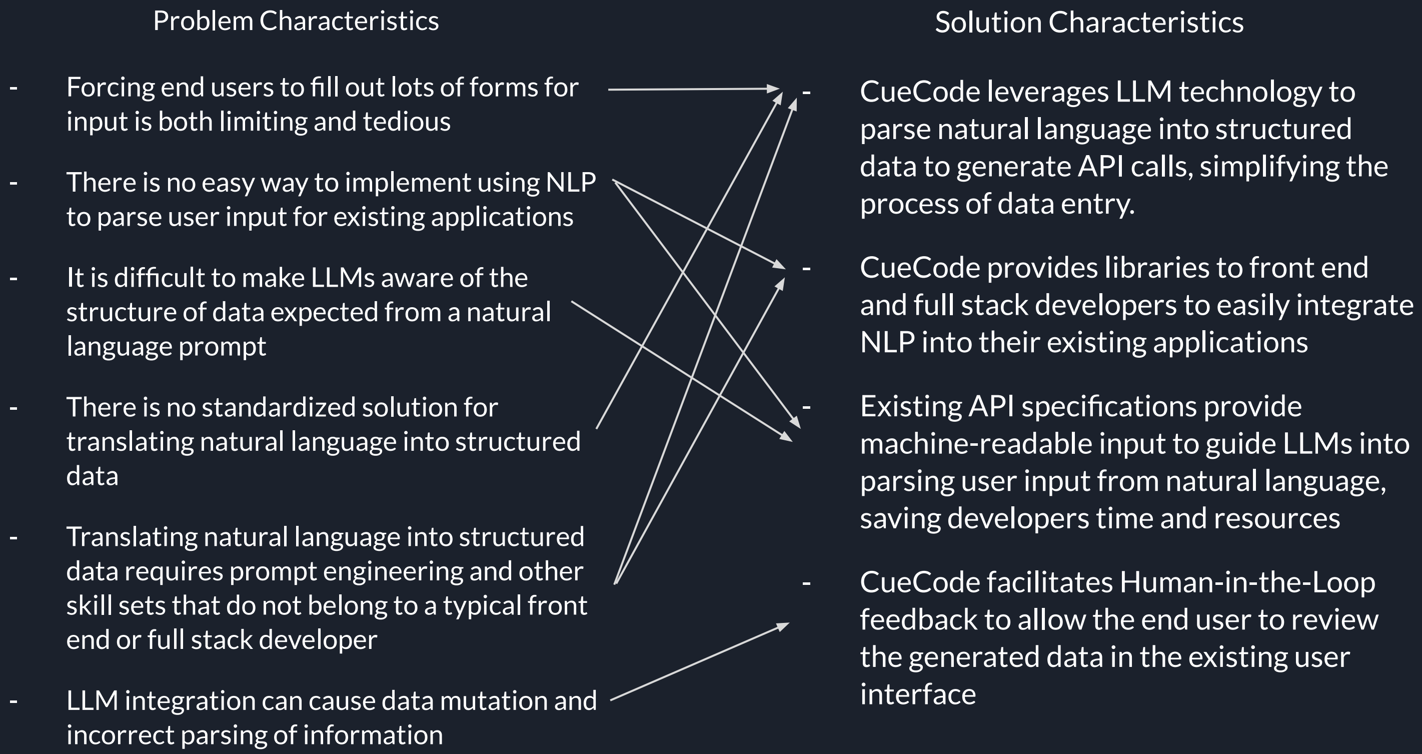
So, our client service representative can provide input to a booking application using CueCode in natural language, “I called Patricia Davis and rescheduled her appointment from August 1st to August 16th.” The application can then use CueCode’s libraries, which have been configured using documentation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

```
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.

3.2 Solution Characteristics



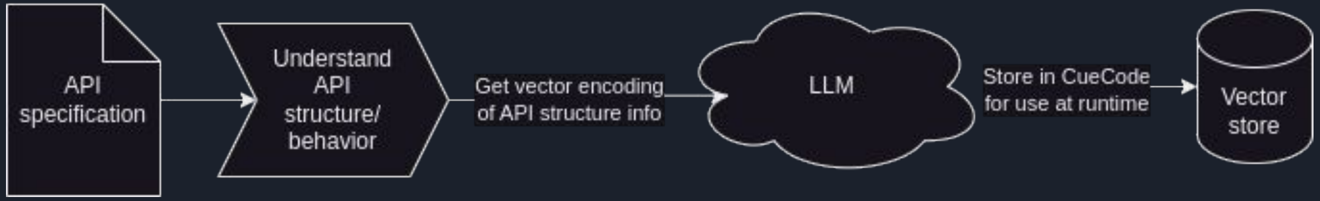
Problem Characteristics

- 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

3.3 Solution Process Flow (configuration)


API specification
Understand API structure/ behavior
Get vector encoding of API structure info
LLM
Store in CueCode for use at runtime
Vector store

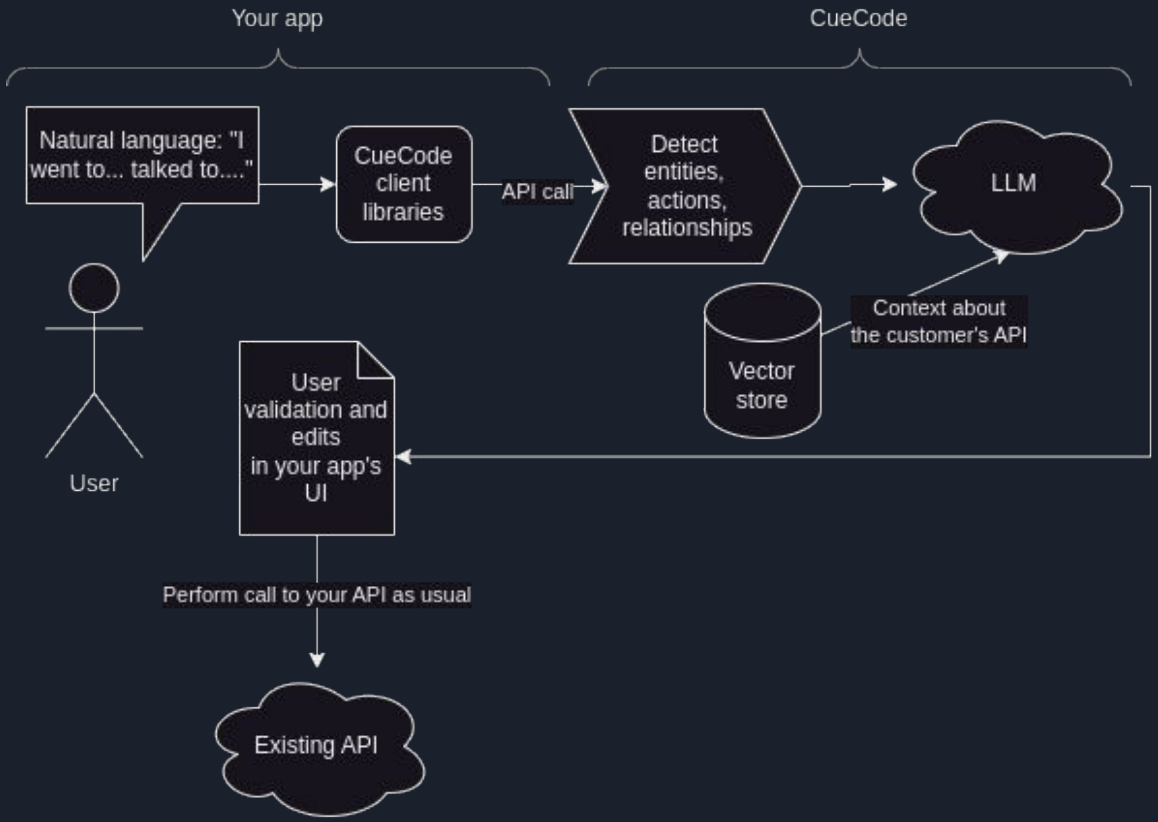
At configuration time:

- Developers ensure their API specification is accurate.
- Developer uploads their API specification to CueCode.
- 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.

3.3 Solution Process Flow (runtime)

Use CueCode in the developer's app:

- 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


Your app
CueCode
Natural language: "I went to... talked to...."
User
CueCode client libraries
API call
Detect entities, actions, relationships
LLM
Vector store
Context about the customer's API
User validation and edits in your app's UI
Perform call to your API as usual
Existing API

3.4 What it Will Do

- Will implement NLP capabilities to enable and understand natural language
- Will offer a user friendly interface (API) that developers can use
- Will provide a developer portal web application, where developers can upload API specifications
- Will enable quick iteration and prototyping by allowing developers to test and refine how their applications respond to the natural language inputs.
- 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 reduce the time and financial investment typically required for implementing NLP, making it affordable for smaller teams and startups
- Will use API specifications, enabling context-aware replies that complement the distinct functionality and data structure of each application.
- Will allow for real time analysis and response generation, enhancing user experience through immediate feedback and interactions.

3.5 What it Will Not Do

- 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 be perfect, misinterpretations could occur with certain slang, ambiguous phrasing or idioms.
- Will not be able to handle complex conversations.
- Will struggle with dialogues, conversations that require deeper understanding.
- 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.

3.6 Competition Matrix

✔- Full Implementation
✓ - Partial Implementation

Feature	CueCode	OpenAI Functions	Google Natural Language API	Spacy.io	LangChain	GenKit	Phone AI Alexa, Siri,...
Entity recognition	✔		✔	✔		✓	✔
Plug and Play	✔				✓	✓	✓
LLM suggests action	✔	✓			✓		✓
Retrieval Augmented Generation	✔	✔			✔	✔	
Requires no LLM Expertise	✔	✔	✓	✓			✔
Natural language to perform action	✓						

4 Development Tools

Version Control:

- **Git with GitHub**
  The industry standard for version control is GitHub With Git. Using branching, pull requests, and issue tracking, it promotes easy collaboration and guarantees that teams function well even on challenging projects. With GitHub's built-in capabilities, we can keep an eye on changes, work together with other team members, and protect our codebase with top-notch security measures.

Integrated Development Environment (IDE):

- **VS Code**
  VS Code is a top option for development across many languages and frameworks because of its wide ecosystem of extensions and high esteem for flexibility. Its Git connection and real-time collaboration tool make coding and team coordination easier and guarantee that our project stays structured and productive.

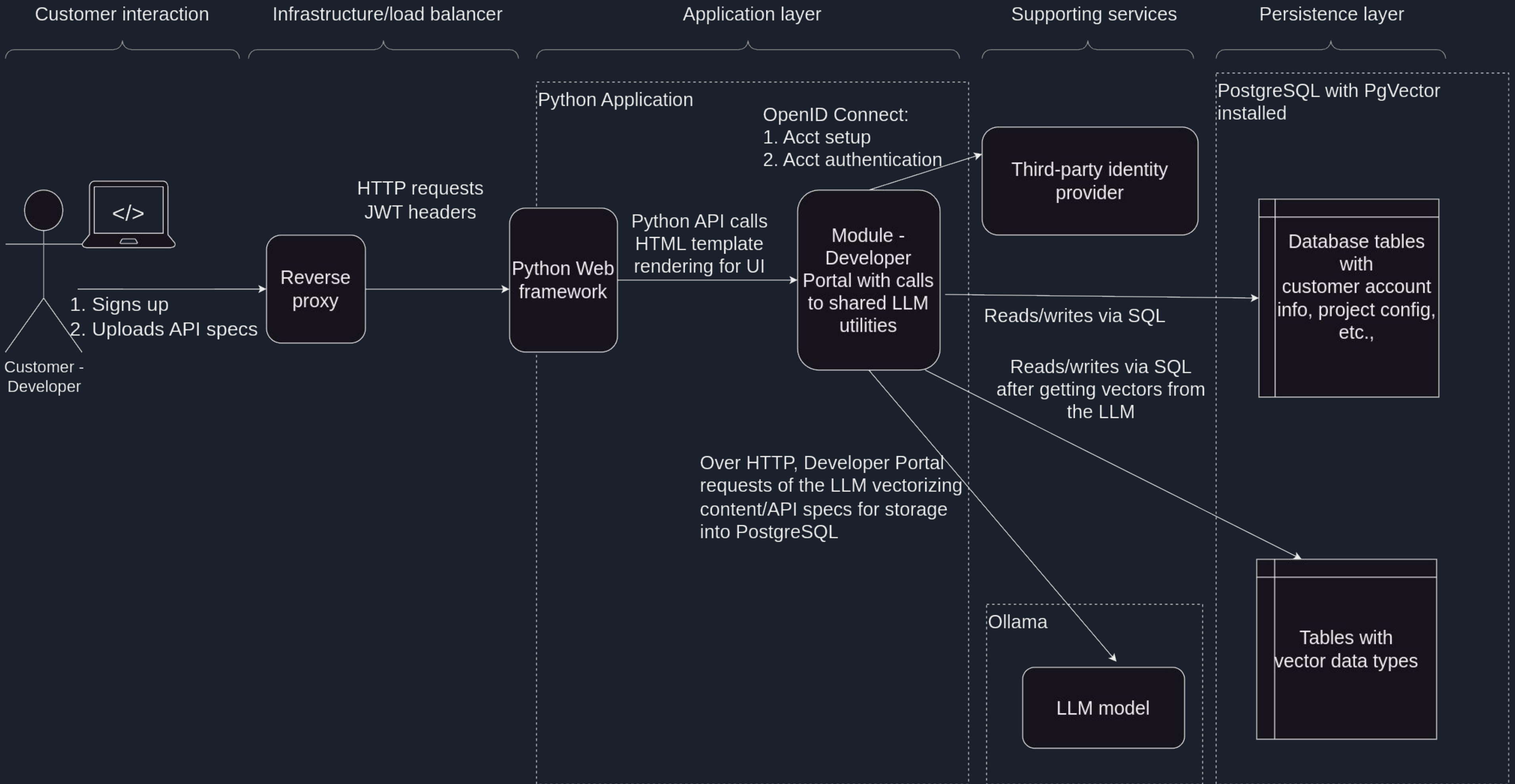
Continuous Integration (CI) & Continuous Deployment (CD):

- **GitHub Actions and Workflows**
  We manage our CI/CD pipelines with GitHub Actions, integrating deployment and testing into an easier process. Given the flexibility that GitHub Workflows offer in automating processes across the development lifecycle, we can confidently deploy, minimize manual intervention, and maintain code quality.

5 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 Call 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 persistence:
    - PgVector extension for storing vectors generated by the LLM
    - Normal PostgreSQL tables for customer accounts, configuration, generation monitoring and audit information
- Ollama:
    - A Web service and set of standardized LLM-call APIs that standardizes running various LLMs in one service
- Third-party identity service:
    - For developer portal
    - TBD on how/whether CueCode runtime API traffic would use the same identity provider for authentication.

5.1 Major Functional Components Diagram - Configuration



Customer interaction

Infrastructure/load balancer

Application layer

Supporting services

Persistence layer

Customer - Developer

</>

1. Signs up
2. Uploads API specs

Reverse proxy

HTTP requests
JWT headers

Python Application

Python Web framework

Python API calls
HTML template rendering for UI

Module - Developer Portal with calls to shared LLM utilities

OpenID Connect:
1. Acct setup
2. Acct authentication

Third-party identity provider

Reads/writes via SQL

Reads/writes via SQL after getting vectors from the LLM

Over HTTP, Developer Portal requests of the LLM vectorizing content/API specs for storage into PostgreSQL

Ollama

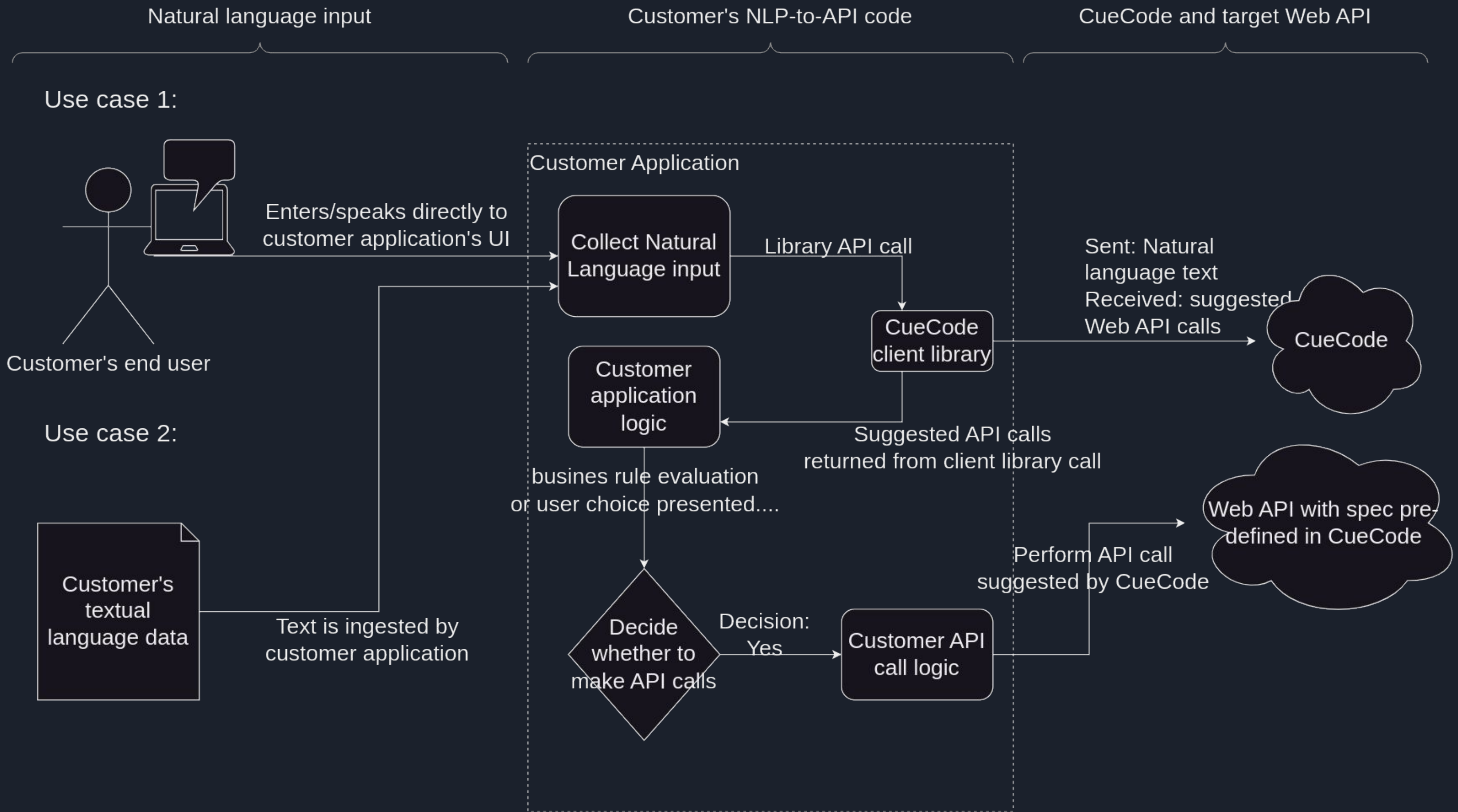
LLM model

PostgreSQL with PgVector installed

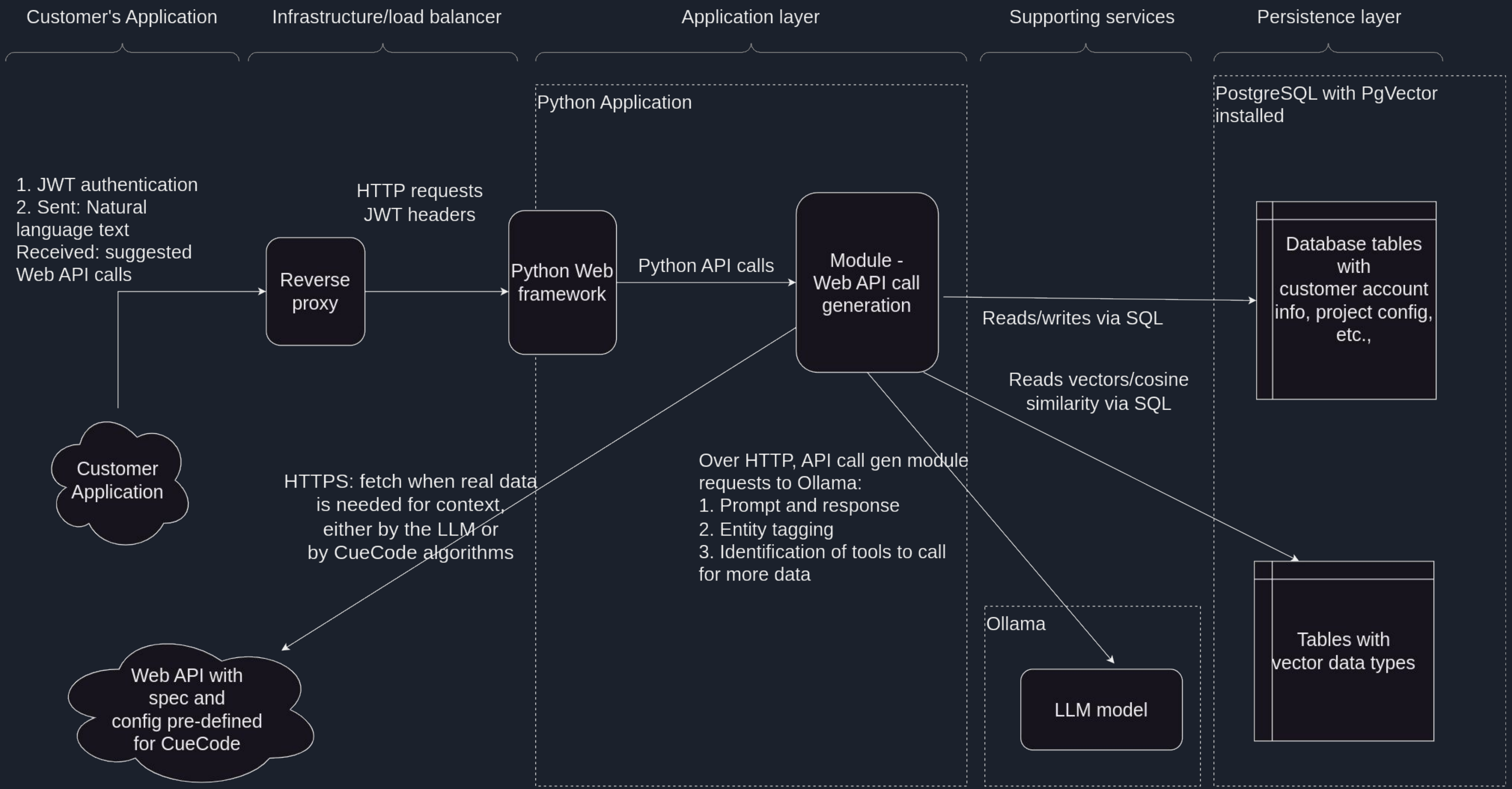
Database tables with customer account info, project config, etc.,

Tables with vector data types

5.1 Major Functional Components Diagram - Runtime - Customer Application


Natural language input
Customer's NLP-to-API code
CueCode and target Web API
Use case 1:
Customer's end user
Enters/speaks directly to customer application's UI
Use case 2:
Customer's textual language data
Text is ingested by customer application
Customer Application
Collect Natural Language input
Library API call
CueCode client library
Sent: Natural language text
Received: suggested Web API calls
CueCode
Customer application logic
Suggested API calls returned from client library call
busines rule evaluation or user choice presented....
Decide whether to make API calls
Decision: Yes
Customer API call logic
Perform API call suggested by CueCode
Web API with spec pre-defined in CueCode

5.1 Major Functional Components Diagram - Runtime - CueCode



Customer's Application

Infrastructure/load balancer

Application layer

Supporting services

Persistence layer

Python Application

PostgreSQL with PgVector installed

1. JWT authentication
2. Sent: Natural language text
Received: suggested Web API calls

HTTP requests
JWT headers

Reverse proxy

Python Web framework

Python API calls

Module - Web API call generation

Reads/writes via SQL

Database tables with customer account info, project config, etc.,

Reads vectors/cosine similarity via SQL

Customer Application

HTTPS: fetch when real data is needed for context, either by the LLM or by CueCode algorithms

Over HTTP, API call gen module requests to Ollama:
1. Prompt and response
2. Entity tagging
3. Identification of tools to call for more data

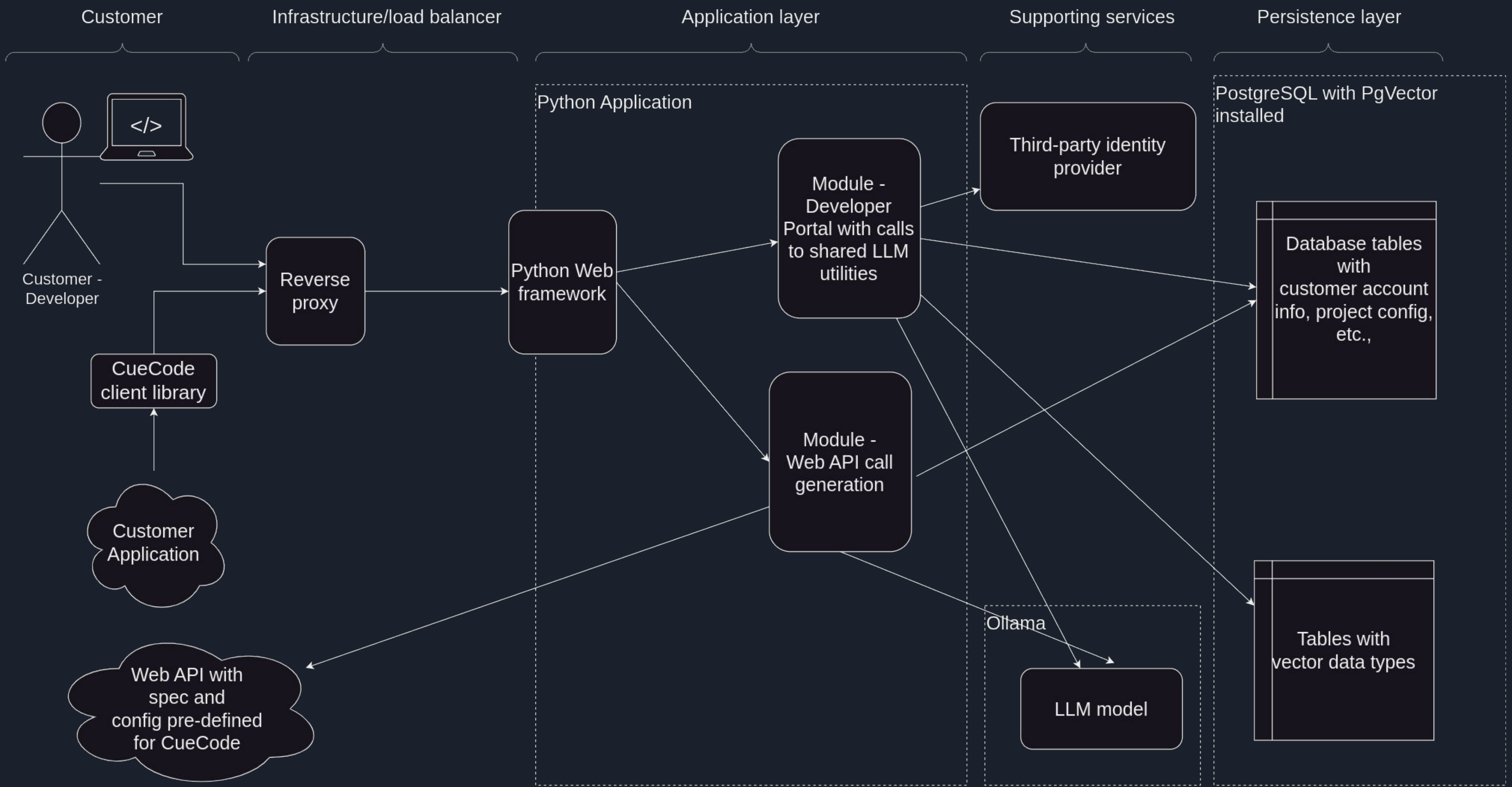
Ollama

LLM model

Tables with vector data types

Web API with spec and config pre-defined for CueCode

5.1 Major Functional Components Diagram - Overview


Customer
Infrastructure/load balancer
Application layer
Supporting services
Persistence layer
</>
Customer - Developer
CueCode client library
Customer Application
Web API with spec and config pre-defined for CueCode
Reverse proxy
Python Application
Python Web framework
Module - Developer Portal with calls to shared LLM utilities
Module - Web API call generation
Third-party identity provider
Ollama
LLM model
PostgreSQL with PgVector installed
Database tables with customer account info, project config, etc.,
Tables with vector data types

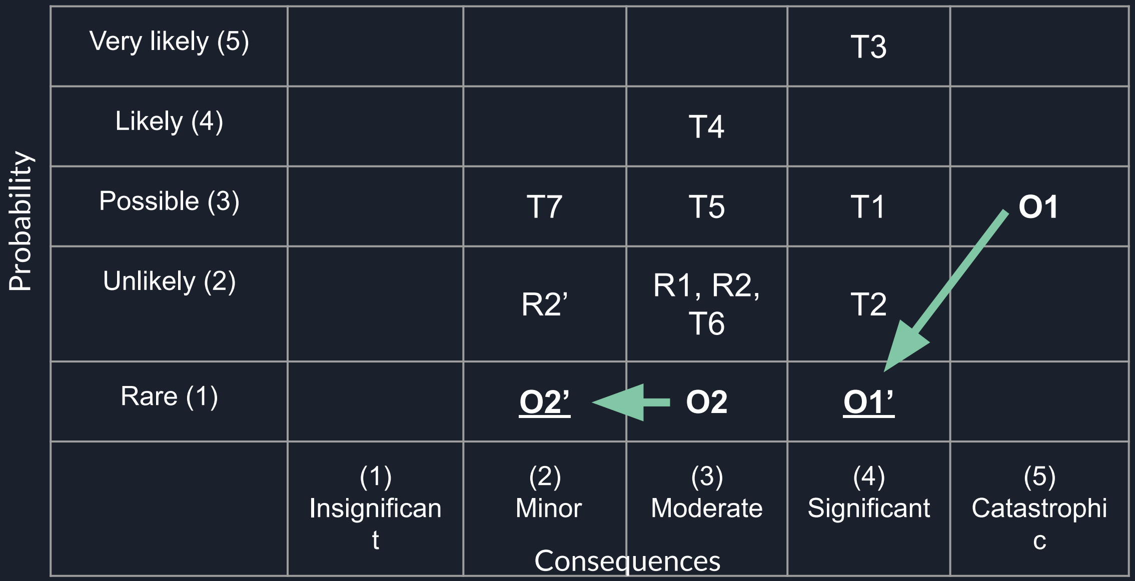
6 Risks - Customer, Operational, Regulatory

O1 - Unable to procure GPU Hardware for development.

- **Mitigation approach:** Control
- **Mitigation:**
  - Ask for GPU time from the CS department
  - Personal contacts and networking

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

6 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.

Probability

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

Consequences

6 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 for user-specific data

Probability

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

Consequences

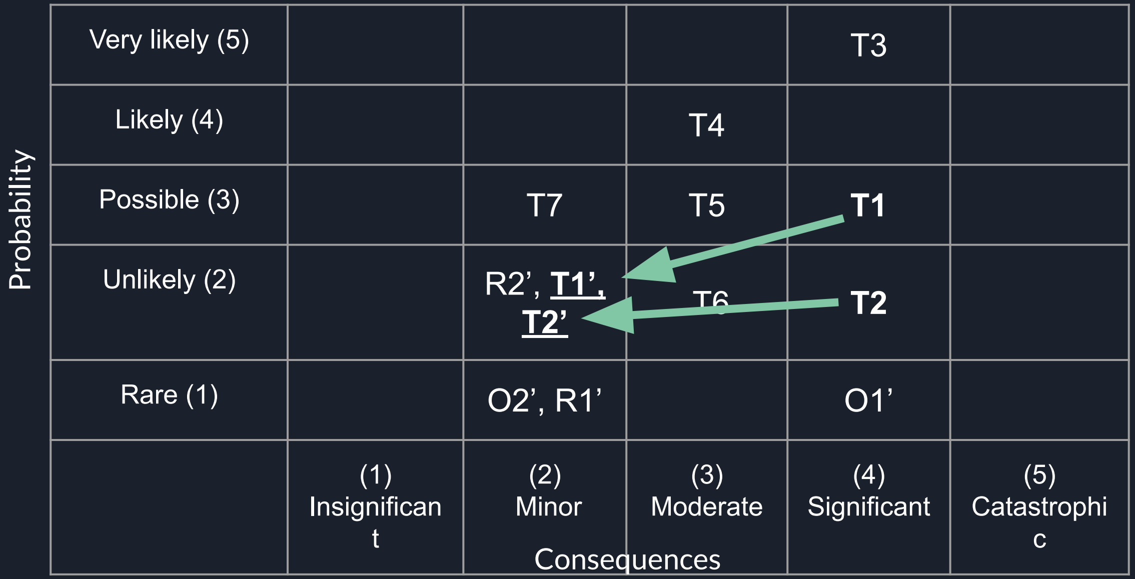
6 Risks - Technical

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

- **Mitigation approach:** Control
- **Mitigation:** Require that developers include a few examples in their OpenAPI specs.

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

- **Mitigation approach:** Continue Monitoring.
- **Mitigation:** Pivot to change value propositions and require backend development from the customer to publish API request bodies to CueCode for its consumption and storage.



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

Consequences

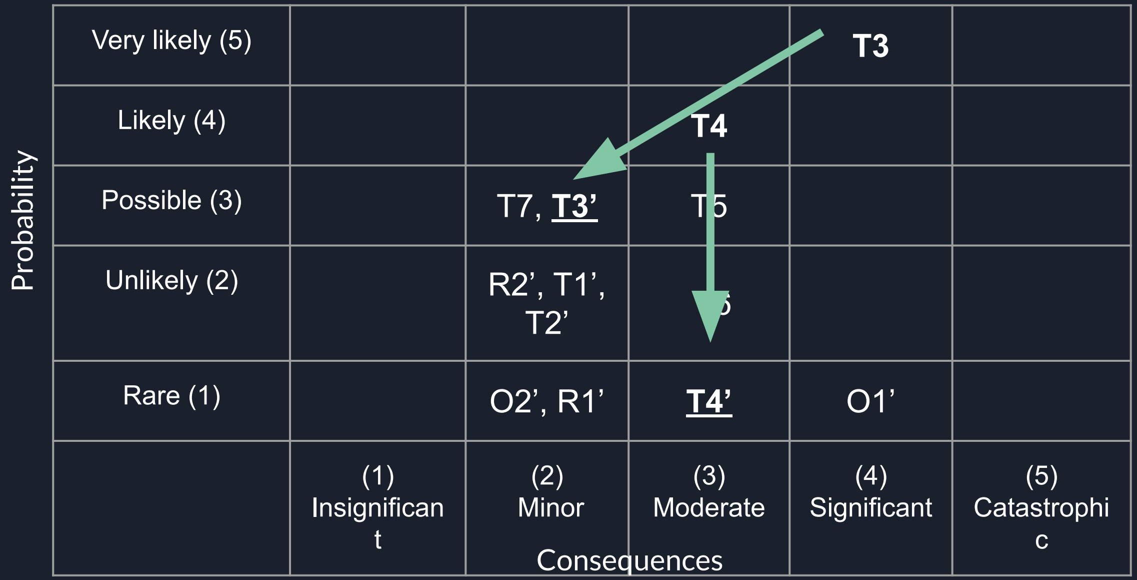
6 Risks - Technical

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:** Partition customer data; Give customers the ability to partition their customers' data in CueCode's data storage; use strong encryption when transferring data; and enforce stringent access limits.



Probability

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

Consequences

6 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.

Probability

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

Consequences

6 Risks - Technical

T6 - The performance of an API model declines with complexity.

- **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

6 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.

Probability

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

Consequences

6 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

7 References

[1]

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

[2]

E. at 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]

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.

[4]

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

7 References

[5]

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

[6]

guidance-ai/guidance. (Sep. 25, 2024). Jupyter Notebook. guidance-ai. Accessed: Sep. 25, 2024. [Online]. Available: https://github.com/guidance-ai/guidance

[7]

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

[8]

OpenAPITools/openapi-generator. (Sep. 10, 2024). Java. OpenAPI Tools. Accessed: Sep. 10, 2024. [Online]. Available: https://github.com/OpenAPITools/openapi-generator

7 References

[9]

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

[10]

“What Is NLP (Natural Language Processing)? | IBM.” Accessed: Sep. 10, 2024. [Online]. Available: https://www.ibm.com/topics/natural-language-processing

[11]

“Cloud Natural Language,” Google Cloud. Accessed: Sep. 26, 2024. [Online]. Available: https://cloud.google.com/natural-language

7 References

[12]

“Projects · spaCy Usage Documentation,” Projects, 2016. https://spacy.io/usage/projects (accessed Oct. 03, 2024).

[13]

“Firebase Genkit,” Firebase. https://firebase.google.com/docs/genkit

[14]

“Github Docs,” Github, Inc. https://docs.github.com/en/get-started/using-git/about-git

[15]

“VS code for educators and students,” Visual Studio Code.

https://code.visualstudio.com/docs/getstarted/educators-and-students

7 References

8 Appendix

8.1 Real World Product vs Prototype Table

Not in scope for Feasibility iteration 3.

That said, we will implement CueCode for OpenAPI specs but not GraphQL specs.