SE 4485: Software Engineering Projects

Spring 2025

Architecture Documentation

|  |  |
| --- | --- |
| Group Number | 11 |
| Project Title | Design-of-experiments Interactive Oracle LLM App |
| Sponsoring Company | The Fellows Consulting Group (FCG) |
| Sponsor(s) | Tom Hill |
| Students | 11. Alberto Escobar  2. Dara Moheimani  3. Lee Rafael Filomeno  4. Brandon Bailey  5. Blanca Berrios Henriquez  6. Saishrey Bhandare |

# ABSTRACT

This document describes the architectural design of an Ontology Generation system which will be implemented using Llama 3.2 and the Ollama API. The system combines a WordPress frontend with a Python backend to enable users to generate domain specific ontologies with logically validated relationships. The overall architecture is a hybrid of a client-server model with layered components. The layered approach allows for interface concerns to be separated from processing logic. The hybrid architecture supports the system’s functional requirements while maintaining performance and reliability standards. This Comprehensive architecture documentation provides a foundation for implementing an efficient, reliable, and user-friendly ontology generation system.

# TABLE OF CONTENTS

# Introduction………………………………………………………………………………………….

# Architectural Style(s) Used………………………………………………………………………….

# Architectural Model…………………………………………………………………………………

# Technology, Software, and Hardware Used………………………………………………………...

# Rationale for Architectural Style and Model………………………………………………………..

# Traceability from Requirements to Architecture……………………………………………………

# Evidence of Configuration Management……………………………………………………………

# Engineering Standards and Constraints……………………………………………………………..

# Additional References……………………………………………………………………………….

# LIST OF FIGURES

# Figure 1: Architecture Model

1. Figure 2: Deployment Diagram

# LIST OF TABLES

# Table 1: Technology Stack Table

1. Table 2: Hardware Requirements Table

# Table 3: Requirements to Architecture Mapping Table

1. Table 4: CM tool change log

# INTRODUCTION

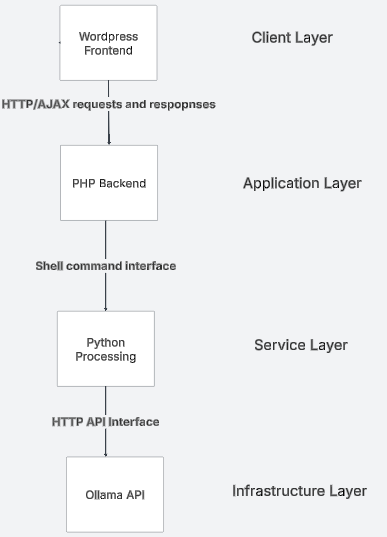
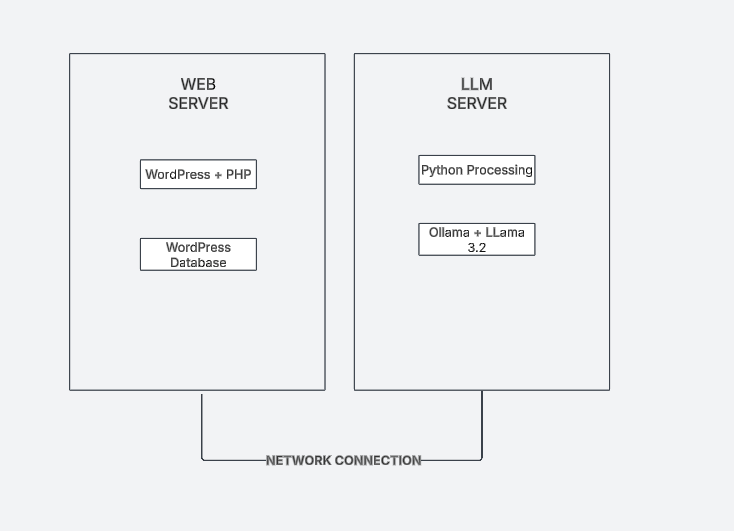
# The document outlines the design choices, overall system structure, and technology choices for the Ontology Generation LLM. The system enables users to generate domain ontologies with logical relationships using Llama 3.2 and a WordPress interface. The purpose of the document is to define architectural components and their interactions, explain the rationale behind these architectural choices, identify the technology stack, and map system requirements to architectural requirements. The document mainly covers high-level architecture and does not go too far into specifics besides what is necessary to understand the design choices that were made. It is organized into sections that cover architectural styles that were used, a component model, the technology stack, design rationale, and requirements traceability.

# ARCHITECTURAL STYLE(S) USED

The Ontology Generation LLM employs a combination of Client- Server Architecture and Layered Architecture styles. The Client-Server architecture separates the WordPress frontend (client) where users input domain keywords from the processing logic in the Python backend (server). The separation of concerns leads to many benefits such as load distribution, where the resource intensive LLM processing is isolated from the UI.

The Layered Architecture complements the approach by organizing the system’s functionality into different layers with clear responsibilities. The WordPress frontend layer handles user interaction and initial input validation. The PHP layer manages communication, security, and process coordination. The Python service layer implements the core logic for ontology generation. It also enforces logical relationship rules by ensuring ontologies follow natural subject verb-object order. Finally, the final layer connects Ollama to the rest of the system. The hybrid approach taken allows for comprehensive error handling at various layers, and it also allows the frontend to remain responsive as the ontology is generated.

# ARCHITECTURAL MODEL The system can be broken down into 4 different subsystems: WordPress Frontend, PHP backend, Python processing, and the Ollama API. The Frontend is responsible for the UI for the domain input, displaying results, and handling AJAX requests. The Backend is responsible for short code registration, admin configuration, process control (executing the Python Script), and security. The Python Processing layer is responsible for ontology generation, JSON parsing and validation, relationship validation, and error handling and fallbacks. The final layer is responsible for hosting the model itself and text generation. The model's overall architecture and deployment diagrams are included below. In the architecture model all relationships are bidirectional to indicate the flow of a JSON response back to the Client Layer.

  
**Figure 1: Architecture Model**  


**Figure 2: Deployment Diagram**

# TECHNOLOGY, SOFTWARE, AND HARDWARE USED

The Ontology Generation LLM project employs a diverse technology stack to achieve its objectives. WordPress serves as the foundation for the user interface, providing content management, plugin infrastructure, and admin capabilities. PHP powers all the server-side components of the WordPress plugin: this includes handling HTTP requests, security, and process management. Python 3 was selected for the ontology generation script due to its text processing capabilities and its API interaction libraries. The system’s core intelligence comes from Llama 3.2 with the help of local API service provided by Ollama. Throughout the system JSON serves as the standard data interchange format which allows for a structured yet flexible approach to communication between systems.

Hardware requirements will vary based on the deployment scenario, and as such they are subject to change. The web server hosting WordPress needs at minimum 4GB RAM, a dual-core CPU, and 20GB storage, though 8GB RAM, a quad-core CPU, and 40GB storage is recommended for optimal performance. The server running the LLM requires more substantial resources: at minimum 16GB RAM, a quad-core CPU, and 10GB storage, with a recommended configuration of 32GB RAM, an 8-core CPU, an NVIDIA GPU with at least 8GB VRAM, and 20GB storage. If the components are hosted separately, a reliable network connection between the WordPress server and Ollama server is essential.

|  |  |  |  |
| --- | --- | --- | --- |
| **Layer** | **Component** | **Technology** | **Purpose** |
| Presentation | WordPress Frontend | WordPress, jQuery, HTML/CSS | User interactions and display |
| Application | PHP Backend | PHP 7.4+ | Requests handling, security |
| Service | Python Processing | Python 3, Requests library | Process domain input, validate relationships |
| Infrastructure | Ollama Api | Ollama, Llama 3.2 | Natural language processing |

**Table 1: Technology Stack Table**

|  |  |  |
| --- | --- | --- |
| **Component** | **Minimum Requirements** | **Recommended Requirements** |
| Web Server | 4 GB RAM, dual-core CPU, 20 GB storage | 8 GB RAM, quad-core CPU, 40 GB storage |
| LLM Server | 16 GB Ram, quad-core CPU, 10 GB storage | 32GB RAM, 8-core CPU, NVIDIA GPU with 8GB+ VRAM, 20 GB storage |
| Network | 10 MBps connection | 100 Mbps connection |

**Table 2: Hardware Requirements Table**

RATIONALE FOR YOUR ARCHITECTURAL STYLE AND MODEL

The selection of a hybrid Client-Server and Layered Architecture styles was driven by several key factors. First and most important, this approach enables a clear separation of concerns, allowing each component to focus on its own specialized functionality. The fronted focuses exclusively on UI, the PHP handles WordPress integration and security, the Python script implements the ontology generation logic, and the Ollama API provides LLM capabilities.

Another key concern in this architectural design was the performance optimization given the given requirements. The 500 ms validation requirement is easily met by the frontend layer without impacting other operations. The 15-second LLM processing requirement is also addressed through the dedicated Python component: the script has optimized prompts and context management.

Other major concerns were error handling and reliability. With each layer implementing its own domain specific error handling errors are more reliably caught and reported. The fallback ontology mechanism also ensures that users always receive some form of usable output. Comprehensive error logging also supports debugging and continuous improvement of the system. One final concern that was addressed by the architecture was that of scalability. With the separation of concerns deployment across different servers becomes feasible, and resource-intensive LLM processing can be further scaled independently from the UI.

TRACEABILITY FROM REQUIREMENTS TO ARCHITECTURE

|  |  |  |
| --- | --- | --- |
| Requirement | Architectural Components | Implementation Details |
| Display Homepage | WordPress Frontend | WordPress template system with UTD styling patterns |
| Input Domain Keyword | WordPress Frontend | Form interface |
| Validate Input | WordPress Frontend, PHP Backend | Client-side validation (jQuery) for length/char check; Server-side validation (sanitization) |
| Generate Ontology | Python Processing, Ollama API | optimized prompts and context window management |
| Display Ontology Graph | WordPress Frontend | JavaScript returns styled relationship visualization |
| Display Error Message | All subsystems | Error classes in CSS, PHP, exception handling in Python, AJAX error callbacks |
| Performance: Input validation < 500ms | WordPress Frontend | Client-side validation is lightweight and operated independently |
| Performance: LLM processing < 15s | Python Processing, Ollama API | optimized prompts and context window management |
| Performance: UI interactions < 100ms | WordPress Frontend | Asynchronous jQuery event handling |
| Performance: Graph rendering < 2s | WordPress Frontend | Optimized HTML/CSS rendering and efficient DOM generation |
| Performance: Homepage load < 3s | WordPress Frontend | Lightweight template design |
| Reliability: 99.9% uptime | All subsystems | Multi-layer error handling, fallback ontology generation, and input validation |
| Usability: Learn in < 5 minutes | WordPress Frontend | Intuitive interface |

**Table 3: Requirements to Architecture Mapping Table**

EVIDENCE THE DOCUMENT HAS BEEN PLACED UNDER CONFIGURATION MANAGEMENT

|  |  |  |
| --- | --- | --- |
| Version | Author | Change |
| 1 | Alberto Escobar | Made initial version of the document |

**Table 4:** CM tool change log

ENGINEERING STANDARDS AND MULTIPLE CONSTRAINTS

* IEEE Std 1471-2000: Software Architecture
* ISO/IEC/IEEE Std 42030:2019: Software, Systems and Enterprise

Architecture Evaluation Framework

ADDITIONAL REFERENCES

* Lattanze, A.J., 2008. *Architecting Software Intensive Systems: A Practitioner’s Guide.*

CRC Press

* Bass, L., Clements, P. and Kazman, R., 2003. *Software Architecture in Practice.*

Addison-Wesley