**Document 2: Software Specification Document (SPEC)**

**OllamaUI – Software Specification Document  
Document Identifier: SPEC-OllamaUI-001  
Revision: 1.3  
Date: 2025-02-23**

**1. Introduction**

**1.1 Purpose**

**This document describes the technical design, architecture, and module interactions for OllamaUI. It outlines how the requirements defined in the SRS will be implemented and is intended for developers and technical stakeholders.**

**1.2 Scope**

**The specification covers:**

* **The overall system architecture.**
* **Detailed design for each module (both MVP and full features).**
* **State management and dependency injection.**
* **Error handling, logging, testing strategy, deployment, and monitoring.**
* **Future-proofing for enhancements and security integration.**

**2. System Architecture and Design Overview**

**2.1 Architectural Pattern**

**OllamaUI is built using a Clean/Modular Layered Architecture comprising:**

* **Presentation Layer:**
  + **Flutter UI components and navigation (unified UI across platforms).**
* **Business Logic Layer:**
  + **Riverpod providers and controllers handling application workflows.**
* **Data Layer:**
  + **Repositories and service classes for API interactions, engine commands, and local storage.**
* **Platform/Service Layer:**
  + **OS-specific functionality (system tray via system\_tray, file picking via file\_picker, background tasks).**

**2.2 Component Diagram Overview**

* **Engine Management Module:**
  + **EngineService and EngineRepository manage engine detection, installation, and status updates.**
* **Chat & Engine Interaction Module:**
  + **Chat screen, text input widget, and integration with the Engine module.**
* **Model Management Module:**
  + **ModelRepository handles fetching and managing models.**
* **Future Feature Modules:**
  + **Query History, Performance Monitoring, Downloads & Updates, Navigation, and Multimodal Input Handling modules.**
* **Nice-to-Have Modules:**
  + **Search Integration, Advanced Model Comparison, Diagnostics & Recommendations.**

**3. Detailed Module Design**

**3.1 Core Modules (MVP)**

**Engine Management & Status**

* **Responsibilities:**
  + **Check for engine installation at startup; if absent or outdated, trigger automated installation/update.**
  + **Run engine in the background and report status.**
* **Key Classes/Interfaces:**
  + **EngineService, EngineRepository**
  + **UI elements: system tray icon, status indicators.**
* **Integration:**
  + **Invoked at application startup via Riverpod providers.**

**Chat & Engine Interaction**

* **Responsibilities:**
  + **Allow users to type queries and view responses.**
  + **Integrate multimodal input (attachments processed to text).**
* **Key Components:**
  + **Chat screen widget, input field, send button.**
  + **Integration with Engine Management for processing queries.**

**Basic Model Management**

* **Responsibilities:**
  + **List models, load a selected model, and delete models.**
* **Key Components:**
  + **ModelRepository, model list and detail screens.**
* **Interactions:**
  + **User selects a model from the list; corresponding actions are handled by business logic.**

**3.2 Full Feature Modules (Future Enhancements)**

* **Query History & Favorites:**
  + **Log user queries, store timestamps, and mark favorites.**
* **Performance Monitoring & Alerts:**
  + **Display real-time metrics and trigger alerts.**
* **Downloads & Updates:**
  + **Manage model download processes with visual feedback.**
* **Navigation and Global UI Components:**
  + **A unified navigation bar for accessing different modules.**
* **Multimodal Input Handling:**
  + **Provide a dedicated interface for file/image input with drag-and-drop support.**

**3.3 Nice-to-Have Modules**

* **Search Integration:**
  + **Integrate an external search API for querying models/content.**
* **Advanced Model Comparison:**
  + **Enable detailed side-by-side comparisons of models.**
* **Diagnostics & Recommendations:**
  + **Run local diagnostics and provide model recommendations based on system performance.**

**4. State Management and Dependency Injection**

* **State Management:**
  + **Implemented with Riverpod, ensuring modular and testable state logic.**
* **Dependency Injection:**
  + **All services (network, storage, engine) are provided via Riverpod providers, allowing for easy swapping (e.g., future secure implementations).**

**5. Error Handling, Logging, and Testing**

* **Error Handling:**
  + **Global error handlers for both UI and background operations.**
* **Logging:**
  + **Use the logger package to capture logs across modules.**
* **Testing Strategy:**
  + **Comprehensive unit tests (using Flutter’s test framework and mockito).**
  + **Integration tests for module interactions.**
  + **UI/Acceptance tests using Flutter’s integration\_test package.**
  + **Performance tests and CI/CD integration via GitHub Actions (covering Windows, macOS, and Linux).**

**6. Deployment, Release Management, and Monitoring**

* **Deployment:**
  + **Fully Automated Deployment using GitHub Actions, with installer generation (e.g., Inno Setup, DMG packaging).**
* **Monitoring:**
  + **Integrated logging and error reporting using Sentry (or similar) for real-time production monitoring.**

**7. Maintenance, Extensibility, and Future-Proofing**

* **Documentation:**
  + **Module-specific documentation maintained alongside code with a central index.**
* **Code Quality:**
  + **Regular code reviews and refactoring, supplemented by automated tools (Dart Analyzer, linters).**
* **Extensibility:**
  + **Clear interface definitions and dependency injection ensure that new features (security, advanced input, etc.) can be added with minimal disruption.**

**8. Legal and Licensing Considerations**

* **License Management:**
  + **Automated License Management tools will be integrated in the CI/CD pipeline to continuously scan dependencies and ensure compliance.**

**9. Future Security Integration**

* **Security-Ready Design:**
  + **Abstractions in network communication and data storage allow for easy integration of encryption and additional security features in the future.**

**10. Conclusion**

**This Software Specification Document outlines the technical blueprint for OllamaUI. The design leverages a Clean/Modular Layered Architecture, Riverpod for state management, and robust automated deployment and monitoring processes. The system is designed to deliver an MVP with core engine management, chat interaction, and basic model management, while providing a clear roadmap for full feature integration and future enhancements.**