Cellyn MVP - Technical Response & Development Strategy

---

1. Project Understanding

Cellyn aims to deliver an intelligent Excel Add-in that empowers finance professionals to interact with spreadsheets using natural language. The system will recognize spreadsheet structures, financial contexts, and execute automation workflows directly inside Excel, securely and efficiently.

The MVP will focus on implementing:

- A **Context Engine** that parses Excel data (tables, formulas, pivots, color, layout).
- A **Custom Orchestration Framework** (replacing LangChain) for dynamic AI routing.
- A **Memory Layer** for contextual persistence across user actions.
- An **Action Layer** for executing Excel commands (insert, calculate, format).
- A **Macro-Intelligent Workflow Engine** for recording and replaying automation sequences.
- An **Error Handling Subsystem** to ensure resilience and transparency.

The architecture will emphasize modularity, data privacy, and extensibility for future enterprise deployments.

---

2. Technical Approach & Development Strategy

2.1 Architectural Overview

The MVP will follow a **modular multi-agent architecture** consisting of the following layers:

Layer	Description	Core Technology
Frontend Layer	React-based Excel Add-in UI built with Office.js, providing chat and workflow panes.	React, Office.js, TypeScript
Backend Layer	Python API server hosting context processing, orchestration, and workflow logic.	FastAPI, Uvicorn, Pydantic

Layer	Description	Core Technology
AI Layer	LLM interface for natural-language interpretation and structured task planning.	OpenAI GPT-4/5 API or Anthropic Claude
Orchestrator Layer	Custom controller for intent classification, routing, and validation (LangChain-free).	Python custom agents + async routing
Context Engine	Extracts spreadsheet metadata, structure, and dependencies.	Office.js + custom Excel parser
Memory Layer	Stores user intents, prior actions, and contextual embeddings.	SQLite / Redis + sentence-transformer embeddings
Workflow Engine	Manages reusable automation chains and triggers.	Python backend + JSON schema
Error Handling & Logging	Monitors API performance, retries failed tasks, and logs securely.	Python logging + Sentry (optional)
Security Layer	Implements JWT authentication, HTTPS, and anonymization pipeline.	PyJWT, AES-256 encryption

---

2.2 Development Strategy by Component

(a) Context Engine

- Parse Excel sheets, formulas, headers, colors, and tables using `Office.js` APIs.
- Convert workbook data into structured JSON for the backend AI model.
- Maintain formula dependency mapping (cell references, relationships).
- Cache parsed metadata for reuse in subsequent commands.

Deliverables:

- Excel context extraction module
- JSON schema for workbook structure
- Sample test report with financial model Excel files

(b) Custom Orchestrator (LangChain Replacement)

- Implement lightweight orchestration logic in Python using async tasks.
- Handle multi-agent coordination (Parser Agent, Action Agent, Memory Agent).
- Validate model outputs and ensure secure Excel command translation.
- Route requests dynamically depending on user intent type (formula, insight, automation).

Deliverables:

- Modular orchestrator class with pluggable agents

- Task router API endpoint `/orchestrate`
- Validation layer for Excel commands

(c) Action Layer

- Implement secure Excel operations via Office.js API calls:
  - Insert rows, columns, tables
  - Write formulas and data
  - Apply conditional formatting and charts
- Enable controlled, human-in-the-loop execution (preview → confirm → apply).

Deliverables:

- Action execution API
- Excel operation templates
- Confirmation UI flow

(d) Memory Layer

- Context persistence per workbook session.
- Embedding-based recall for previously used headers, formulas, and task patterns.
- Integration with Redis or local SQLite for low-latency lookups.

Deliverables:

- Memory API (store, recall, update)
- Context retrieval integration with orchestrator

(e) Workflow Automation Engine

- Record user actions and AI steps as a JSON workflow chain.
- Allow users to replay, edit, and schedule workflows.
- Include metadata: creator, timestamp, triggers, step dependencies.

Deliverables:

- Workflow schema definition
- CRUD API for workflows
- Excel macro recording integration

(f) Error Handling & Logging

- Centralized logging with structured JSON logs.
- Retry mechanism for failed API or AI calls.
- Error categorization (API, Model, Excel, Network).

Deliverables:

- Error handler middleware
- Retry + notification subsystem
- Log dashboard (optional in future release)

---

3. Available Resources & Tools

Category	Tools / Resources
Programming Languages	Python 3.11, TypeScript (for Office Add-in)
Frameworks	FastAPI, React, Office.js
Databases	SQLite (MVP), optional Redis for caching
AI/ML Libraries	OpenAI SDK, HuggingFace Transformers, Sentence-Transformers
Testing	Pytest, Jest
CI/CD	GitHub Actions (already configured)
Version Control	Git + GitHub
Documentation	MkDocs / Markdown / docx exports
Deployment	Docker-based one-click deployment for testing
Security	JWT Auth, HTTPS (TLS), AES-256 encryption for data at rest

---

4. Development Timeline (1-Month MVP Sprint)

The goal is to deliver a working MVP with core capabilities (context engine, orchestration, and Excel actions) within four weeks.

Week	Milestones	Deliverables
Week 1	Project setup, architecture finalization, environment configuration	Final architecture diagram, local dev setup verified
Week 2	Build Context Engine and integrate Excel parser	JSON schema output, context extraction API

Week	Milestones	Deliverables
Week 3	Implement Orchestrator + Action Layer	AI → Excel command execution (create table, formula, highlight)
Week 4	Add Memory + Error Handling + Testing	Context persistence, retry mechanism, integration tests
End of Week 4	MVP Demonstration & Feedback	Working demo with provided sample Excel models

Total Duration: 4 weeks (≈160 hours)
Total Estimate: $5,500 USD (Discounted Fixed Package)

---

5. Risk Mitigation & Quality Strategy

Risk	Mitigation Plan
Model Output Variability	Implement structured prompt templates and validation filters.
Excel API Rate/Latency Issues	Use batch operations and async execution.
Data Security	Redact sensitive cell values before LLM calls; encrypt all traffic.
Cross-Platform Issues (Windows/Mac/Web)	Test with Office 365 and Desktop versions during QA phase.
Timeline Compression	Prioritize context engine and orchestration for MVP; workflow automation can extend post-MVP.

Testing Approach:

- Unit testing for all backend modules
- Integration testing with provided Excel files
- Manual validation for finance models (M&A, LBO, DCF)
- Continuous logging and bug tracking

---

6. Next Steps

Upon client confirmation:

1. Finalize scope of MVP (confirm features for Month-1 sprint).
2. Set up shared Git repository access.
3. Begin development sprint (daily sync updates).
4. MVP delivery in 4 weeks → client review and feedback iteration.

Conclusion

This proposal outlines a complete development strategy for the **Cellyn MVP**, focusing on flexibility, modularity, and future scalability without reliance on LangChain or proprietary frameworks.