**Intelligent Document Extraction**

*using Agentic AI*

**Architecture and Project Plan**

May 29, 2025

**TSP04 Team**

**High-Level Architecture**

**Overview**

The Intelligent Document Extraction platform leverages Agentic AI to provide document extraction capabilities for global operations. The architecture follows a modular approach with clear separation of concerns between frontend, backend, data layer, and AI processing components.

**System Architecture Diagram**

┌───────────────────────────────────────────────────────────────────────────┐  
│ OCP │  
│ ┌─────────────────┐ ┌─────────────────┐ ┌─────────────────────────┐ │  
│ │ Frontend │ │ Backend │ │ Consumption Layer │ │  
│ │ (React) │◄──┤ (JavaScript) │◄──┤ (JavaScript) │ │  
│ └─────────────────┘ └─────────────────┘ └─────────────────────────┘ │  
│ ▲ ▲ ▲ │  
│ │ │ │ │  
│ │ │ │ │  
│ ▼ ▼ ▼ │  
│ ┌─────────────────┐ ┌─────────────────┐ ┌─────────────────────────┐ │  
│ │ Human-in-Loop │ │ Data Layer │ │ Batch Processing │ │  
│ │ Interface │ │ (MongoDB) │ │ (Java) │ │  
│ └─────────────────┘ └─────────────────┘ └─────────────────────────┘ │  
│ ▲ ▲ │  
│ │ │ │  
└────────────────────────────────┼────────────────────────┼────────────────┘  
 │ │  
 ┌────────┴────────┐ ┌──────────┴───────────┐  
 │ LLM Engine │ │ Ingestion API │  
 │ (Python) │ │ (JavaScript) │  
 │ Langchain │ └──────────────────────┘  
 └─────────────────┘

**Technology Stack**

**Infrastructure**

- **Hosting Environment**: OpenShift Container Platform (OCP)

- **Deployment Model**: Active-Passive configuration for high availability

**Application Layer**

1. **Frontend**

- **Technology**: React

- **Purpose**: User interface for document ingestion via API from ICMP, extraction review, and human-in-the-loop corrections

2. **Backend**

- **Technology**: JavaScript (Node.js)

- **Purpose**: API services, business logic, and orchestration

3. **Consumption Layer**

- **Technology**: JavaScript (Node.js)

- **Purpose**: API endpoints for downstream systems to consume extraction results

4. **Batch Processing**

- **Technology**: Java

- **Purpose**: Handle batch processing tasks and scheduled operations

**Data Layer**

- **Database**: MongoDB

- **Purpose**: Store extraction results, metadata, and document lineage

- **Data Structure**: Document-based schema that maintains complete lineage:

- Original document

- Extraction instructions

- Raw extraction results

- Processed extraction results

- Human corrections (if any)

**AI/ML Layer**

1. **LLM Engine**

- **Technology**: Python with Langchain framework

- **Purpose**: Document understanding, extraction, and processing

2. **Ingestion API**

- **Technology**: JavaScript (Node.js)

- **Purpose**: Lightweight API for document ingestion

**Key Components and Workflows**

**Document Ingestion Flow**

1. Documents are submitted through the Ingestion API

2. Documents are stored in MongoDB with metadata

3. Extraction jobs are queued for processing

**Extraction Flow**

1. LLM Engine retrieves documents from MongoDB

2. Langchain-based extraction processes the documents

3. Extraction results are stored back in MongoDB with full lineage

**Human-in-the-Loop Flow**

1. UI displays extraction results

2. Human reviewers can correct/validate extractions

3. Corrections are stored in MongoDB with lineage

**Consumption Flow**

1. Downstream systems access extraction results via Consumption Layer APIs

2. Results can be filtered by extraction ID or other metadata

3. Optional adapters can be built for specific downstream systems (e.g., AIMS)

**System Characteristics**

1. **Scalability**

- Containerized deployment in OCP allows horizontal scaling

- Lightweight JavaScript services for faster processing

- Document-based storage for flexible schema evolution

2. **Reliability**

- Active-Passive deployment for failover

- Complete document lineage for auditability

- Human-in-the-loop for quality assurance

3. **Extensibility**

- API-first design for integration with downstream systems

- Modular architecture allowing component upgrades

- Separation of concerns between extraction and consumption

**Project Plan**

**Team Structure (12 Members)**

**Scrum Team 1: Core Platform & Frontend (4 members)**

- 1 Tech Lead/Scrum Master

- 2 React Developers

- 1 UX/UI Designer

**Scrum Team 2: Backend & Data (4 members)**

- 1 Backend Lead/Scrum Master

- 1 Node.js Developer

- 1 MongoDB Specialist

- 1 Java Developer (Batch Processing)

**Scrum Team 3: AI & Integration (4 members)**

- 1 AI Lead/Scrum Master

- 1 Python/Langchain Developer

- 1 API/Integration Developer

- 1 QA/Test Automation Engineer

**4-Month Delivery Timeline**

**Month 1: Foundation & MVP**

**Sprint 1: Architecture & Setup (2 weeks)**

- Set up OCP environment (active-passive)

- Establish CI/CD pipelines

- Create MongoDB schema design

- Define API contracts between components

- Set up development environments

**Sprint 2: Core Components (2 weeks)**

- Implement basic document ingestion API

- Create initial MongoDB data layer

- Set up basic LLM extraction pipeline

- Develop minimal UI for document ingestion

**Month 2: Core Functionality**

**Sprint 3: Extraction Engine (2 weeks)**

- Implement Langgraph-based extraction as Modular functionality for scalability

- Develop document lineage tracking

- Create basic extraction API endpoints

- Set up batch processing framework

**Sprint 4: Frontend & Integration (2 weeks)**

- Develop document viewer UI

- Implement extraction result display

- Create human-in-the-loop correction interface

- Integrate frontend with backend APIs

**Month 3: Advanced Features & Optimization**

**Sprint 5: Advanced Extraction (2 weeks)**

- Enhance LLM extraction capabilities

- Implement extraction templates

- Add metadata extraction features

- Optimize extraction performance

**Sprint 6: Consumption Layer (2 weeks)**

- Develop consumption layer APIs

- Implement filtering and query capabilities

- Create documentation for API consumers

- Add security and access controls

**Month 4: Refinement & Production Readiness**

**Sprint 7: Quality & Performance (2 weeks)**

- Comprehensive testing (load, performance, security)

- Optimize database queries and indexing

- Implement caching strategies

- Enhance error handling and logging

**Sprint 8: Production Deployment (2 weeks)**

- Final integration testing

- Production environment setup

- Documentation and knowledge transfer

- User acceptance testing and feedback incorporation

**Key Deliverables by Phase**

**Phase 1 (Month 1)**

- OCP environment setup with CI/CD

- Basic document ingestion flow

- Initial extraction pipeline

- Minimal UI for document upload

**Phase 2 (Month 2)**

- Complete extraction engine with Langchain

- Document lineage tracking in MongoDB

- Human-in-the-loop interface

- Integration between frontend and backend

**Phase 3 (Month 3)**

- Advanced extraction capabilities

- Consumption layer APIs

- API documentation

- Security implementation

**Phase 4 (Month 4)**

- Performance optimizations

- Production deployment

- Complete documentation

- Training and knowledge transfer

**Risk Management**

1. **Technical Risks**

- LLM performance issues: Mitigate with thorough testing and fallback mechanisms

- Scalability challenges: Address through load testing and architecture reviews

- Integration complexities: Manage with clear API contracts and early integration testing

2. **Project Risks**

- Timeline constraints: Prioritize features and maintain MVP focus

- Resource availability: Cross-train team members on critical components

- Requirement changes: Implement agile change management process

3. **Operational Risks**

- Production deployment issues: Plan for blue-green deployment strategy

- Data security concerns: Implement comprehensive security testing

- Performance in production: Establish monitoring and alerting

**Success Criteria**

1. Platform successfully deployed to OCP in active-passive configuration

2. Document extraction accuracy meets or exceeds 85% for target document types

3. API response times under 2 seconds for standard operations

4. System can handle the required document processing volume

5. Downstream systems can successfully consume extraction results

6. Human-in-the-loop corrections improve extraction quality over time