# **System Design Document**

# **Intelligent Document Extraction using Agentic AI**

## **Document Control**

| **Document Information** |  |
| --- | --- |
| Title | System Design Document |
| Project | TSP04 - Intelligent Document Extraction |
| Date | May 29, 2025 |
| Status | Draft |
| Version | 1.0 |
| Prepared by | Ayyanar Jeyakrishnan |

[System Design Document](#_heading=)

[Intelligent Document Extraction using Agentic AI](#_heading=)

[Document Control](#_heading=)

[1. Introduction](#_heading=)

[1.1 Purpose](#_heading=)

[1.2 Scope](#_heading=)

[1.3 Definitions, Acronyms, and Abbreviations](#_heading=)

[1.4 References](#_heading=)

[2. System Overview](#_heading=)

[2.1 System Context](#_heading=)

[2.2 System Functions](#_heading=)

[2.3 User Characteristics](#_heading=)

[2.4 Constraints and Assumptions](#_heading=)

[3. Architecture](#_heading=)

[3.1 Architectural Overview](#_heading=)

[3.2 Architectural Principles](#_heading=)

[3.3 Technology Stack](#_heading=)

[3.3.1 Infrastructure](#_heading=)

[3.3.2 Application Layer](#_heading=)

[3.3.3 Data Layer](#_heading=)

[3.3.4 AI/ML Layer](#_heading=)

[3.4 System Topology](#_heading=)

[4. System Components](#_heading=)

[4.1 Frontend Component](#_heading=)

[4.1.1 Purpose](#_heading=)

[4.1.2 Functions](#_heading=)

[4.1.3 Interfaces](#_heading=)

[4.2 Backend Component](#_heading=)

[4.2.1 Purpose](#_heading=)

[4.2.2 Functions](#_heading=)

[4.2.3 Interfaces](#_heading=)

[4.3 Data Layer Component](#_heading=)

[4.3.1 Purpose](#_heading=)

[4.3.2 Functions](#_heading=)

[4.3.3 Interfaces](#_heading=)

[4.4 LLM Engine Component](#_heading=)

[4.4.1 Purpose](#_heading=)

[4.4.2 Functions](#_heading=)

[4.4.3 Interfaces](#_heading=)

[4.5 Consumption Layer Component](#_heading=)

[4.5.1 Purpose](#_heading=)

[4.5.2 Functions](#_heading=)

[4.5.3 Interfaces](#_heading=)

[4.6 Batch Processing Component](#_heading=)

[4.6.1 Purpose](#_heading=)

[4.6.2 Functions](#_heading=)

[4.6.3 Interfaces](#_heading=)

[4.7 Ingestion API Component](#_heading=)

[4.7.1 Purpose](#_heading=)

[4.7.2 Functions](#_heading=)

[4.7.3 Interfaces](#_heading=)

[5. Data Architecture](#_heading=)

[5.1 Data Model](#_heading=)

[5.1.1 Document Collection](#_heading=)

[5.1.2 User Collection](#_heading=)

[5.1.3 Extraction Template Collection](#_heading=)

[5.2 Data Flow](#_heading=)

[5.3 Data Retention and Archiving](#_heading=)

[6. Interface Design](#_heading=)

[6.1 User Interfaces](#_heading=)

[6.1.1 Document Upload Interface](#_heading=)

[6.1.2 Extraction Review Interface](#_heading=)

[6.1.3 Dashboard Interface](#_heading=)

[6.2 API Interfaces](#_heading=)

[6.2.1 Ingestion API](#_heading=)

[6.2.2 Extraction API](#_heading=)

[6.2.3 Consumption API](#_heading=)

[6.3 Internal Interfaces](#_heading=)

[6.3.1 Backend to LLM Engine Interface](#_heading=)

[6.3.2 Backend to Batch Processing Interface](#_heading=)

[7. Non-Functional Requirements](#_heading=)

[7.1 Performance](#_heading=)

[7.2 Scalability](#_heading=)

[7.3 Availability](#_heading=)

[7.4 Reliability](#_heading=)

[7.5 Maintainability](#_heading=)

[7.6 Security](#_heading=)

[8. Security Architecture](#_heading=)

[8.1 Authentication and Authorization](#_heading=)

[8.2 Data Security](#_heading=)

[8.3 API Security](#_heading=)

[8.4 Audit and Compliance](#_heading=)

[9. Deployment Architecture](#_heading=)

[9.1 Deployment Model](#_heading=)

[9.2 Container Strategy](#_heading=)

[9.3 CI/CD Pipeline](#_heading=)

[9.4 Monitoring and Logging](#_heading=)

[10. Appendices](#_heading=)

[10.1 Glossary](#_heading=)

[10.2 References](#_heading=)

[10.3 Revision History](#_heading=)

## **1. Introduction**

### **1.1 Purpose**

This System Design Document (SDD) provides a comprehensive architectural overview of the Intelligent Document Extraction system using Agentic AI. It presents the high-level design and architecture of the system to be implemented for global operations.

### **1.2 Scope**

The document covers the system architecture, components, data models, interfaces, and deployment strategy for the Intelligent Document Extraction platform. It serves as a blueprint for the development teams and stakeholders.

### **1.3 Definitions, Acronyms, and Abbreviations**

| **Term/Acronym** | **Definition** |
| --- | --- |
| OCP | OpenShift Container Platform |
| LLM | Large Language Model |
| API | Application Programming Interface |
| UI | User Interface |
| HITL | Human-in-the-Loop |

### **1.4 References**

- TSP04 Architecture and Project Plan

- OpenShift Container Platform Documentation

- MongoDB Documentation

- Langchain Framework Documentation

## **2. System Overview**

### **2.1 System Context**

The Intelligent Document Extraction system is designed to provide automated extraction of structured information from unstructured documents using Agentic AI technologies. The system will be deployed g to support document processing operations across multiple usecases.

### **2.2 System Functions**

The primary functions of the system include:

1. Document ingestion from various sources

2. Intelligent extraction of data using LLM and Langgraph

3. Storage of extraction results with complete lineage

4. Human-in-the-loop review and correction

5. API-based consumption of extraction results by downstream systems

### **2.3 User Characteristics**

The system will be used by:

1. **Document Processors**: Users onboard to the Platform by providing ICMP document as source

2. **System Administrators**: Users who manage the system configuration and monitor performance

3. **API Consumers**: Downstream systems that consume extraction results via APIs

4. **Data Analysts**: Users who analyze extraction performance and quality metrics

### **2.4 Constraints and Assumptions**

**Constraints:**

- The system must be deployed on OpenShift Container Platform

- The system must use MongoDB as the data store

- The system must use Langchain as the LLM framework

- The system must be delivered within 4 months

**Assumptions:**

- Adequate infrastructure resources will be available in OCP

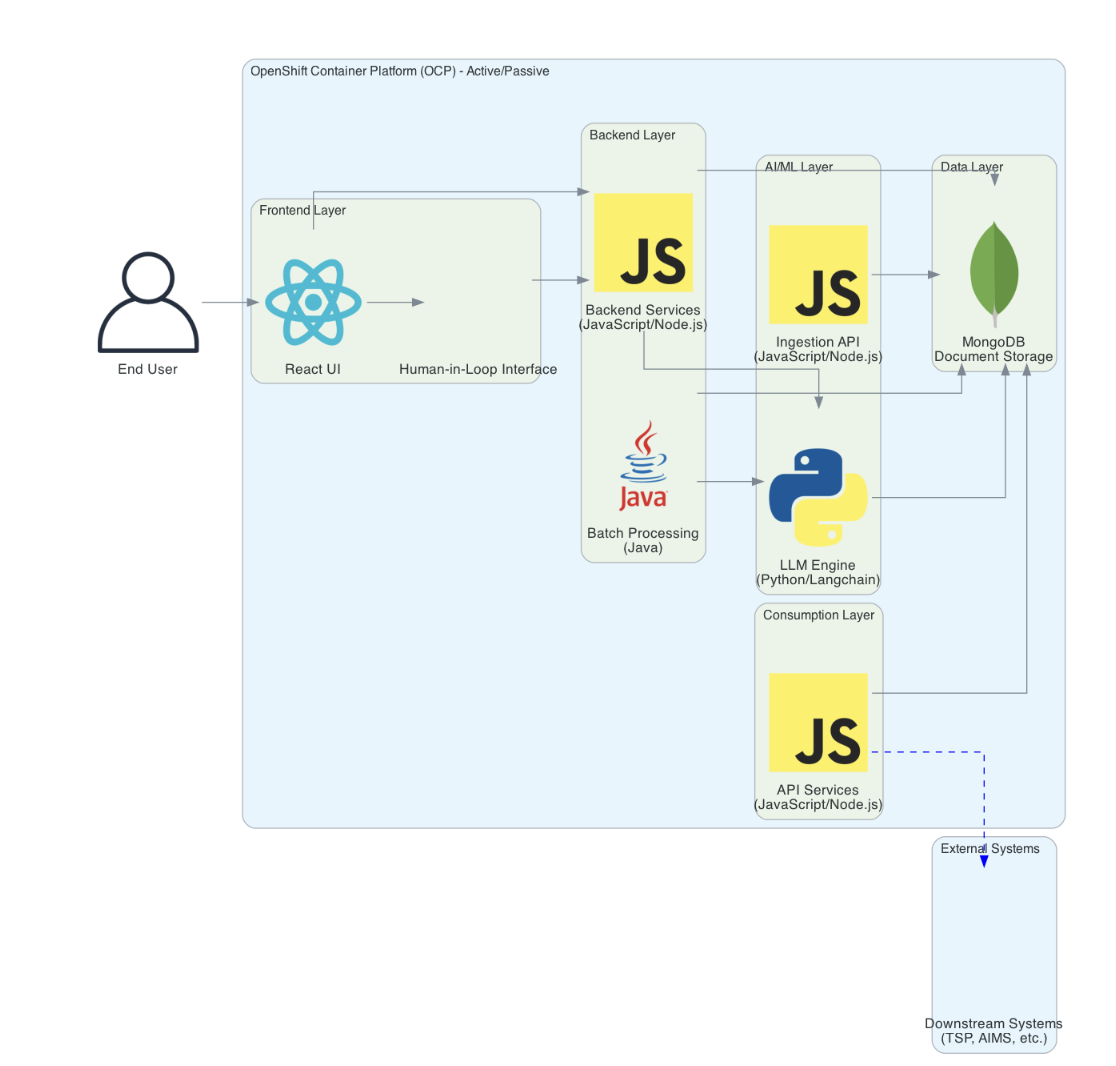
- Required LLM models will be accessible for document extraction

- Downstream systems will be able to consume REST APIs

## **3. Architecture**

### **3.1 Architectural Overview**

The system follows a modular, microservices-based architecture with clear separation of concerns. It is designed to be scalable, resilient, and extensible.



### **3.2 Architectural Principles**

1. **Separation of Concerns**: Clear boundaries between system components

2. **API-First Design**: All components interact through well-defined APIs

3. **Stateless Services**: Application services are stateless for scalability

4. **Data Lineage**: Complete tracking of document processing history

5. **Extensibility**: Modular design allowing for future enhancements

### **3.3 Technology Stack**

#### **3.3.1 Infrastructure**

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

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

- **Container Orchestration**: Kubernetes (via OCP)

- **CI/CD**: OpenShift Pipelines (Tekton)

#### **3.3.2 Application Layer**

1. **Frontend**

- **Framework**: React

- **State Management**: Redux

- **UI Components**: Material-UI

- **Build Tool**: Webpack

2. **Backend**

- **Runtime**: Node.js

- **Framework**: Express.js

- **API Documentation**: Swagger/OpenAPI

- **Authentication**: OAuth 2.0 / JWT

3. **Consumption Layer**

- **Runtime**: Node.js

- **Framework**: Express.js

- **API Gateway**: APIGEE

4. **Batch Processing**

- **Runtime**: Java

- **Framework**: Spring boot

- **Build Tool**: Maven

#### **3.3.3 Data Layer**

- **Database**: MongoDB

- **Caching**: Redis (optional)

- **Data Migration**: MongoDB Migration Tools

#### **3.3.4 AI/ML Layer**

1. **LLM Engine**

- **Language**: Python

- **Framework**: Langgraph

- **Models**: Compatible with various LLM providers

2. **Ingestion API**

- **Runtime**: Node.js

- **Framework**: Express.js

- **File Processing**: Multer

### **3.4 System Topology**

The system will be deployed in an active-passive configuration across two OCP clusters for high availability. Each component will be containerized and orchestrated by Kubernetes within OCP.

## **4. System Components**

### **4.1 Frontend Component**

#### **4.1.1 Purpose**

Provides the user interface for document upload, extraction review, and human-in-the-loop corrections.

#### **4.1.2 Functions**

- Document upload interface

- Extraction result visualization

- Human-in-the-loop correction interface

- User authentication and authorization

- Dashboard for extraction metrics

#### **4.1.3 Interfaces**

- **Inbound**: User interactions via web browser

- **Outbound**: REST API calls to Backend Component

### **4.2 Backend Component**

#### **4.2.1 Purpose**

Provides API services, business logic, and orchestration for the document extraction process.

#### **4.2.2 Functions**

- Document processing orchestration

- Business logic implementation

- Authentication and authorization

- Integration with LLM Engine

- Data validation and transformation

#### **4.2.3 Interfaces**

- **Inbound**: REST API calls from Frontend Component

- **Outbound**:

- Database operations to MongoDB

- API calls to LLM Engine

- Event publishing for batch processing

### **4.3 Data Layer Component**

#### **4.3.1 Purpose**

Stores extraction results, metadata, and document lineage in MongoDB.

#### **4.3.2 Functions**

- Document storage and retrieval

- Metadata management

- Lineage tracking

- Query optimization

- Data backup and recovery

#### **4.3.3 Interfaces**

- **Inbound**: Database operations from Backend Component

- **Outbound**: Data provision to Consumption Layer

### **4.4 LLM Engine Component**

#### **4.4.1 Purpose**

Performs document understanding, extraction, and processing using Langchain framework.

#### **4.4.2 Functions**

- Document text extraction

- Entity recognition

- Relationship extraction

- Document classification

- Confidence scoring

#### **4.4.3 Interfaces**

- **Inbound**: API calls from Backend Component

- **Outbound**: Extraction results to Backend Component

### **4.5 Consumption Layer Component**

#### **4.5.1 Purpose**

Provides API endpoints for downstream systems to consume extraction results.

#### **4.5.2 Functions**

- API endpoint provision

- Data filtering and transformation

- Rate limiting and throttling

- API versioning

- Usage metrics collection

#### **4.5.3 Interfaces**

- **Inbound**: API calls from downstream systems

- **Outbound**: Data retrieval from MongoDB

### **4.6 Batch Processing Component**

#### **4.6.1 Purpose**

Handles batch processing tasks and scheduled operations.

#### **4.6.2 Functions**

- Scheduled document processing

- Bulk extraction operations

- Data export and reporting

- System maintenance tasks

- Performance optimization

#### **4.6.3 Interfaces**

- **Inbound**: Scheduled triggers or event-based triggers

- **Outbound**:

- Database operations to MongoDB

- API calls to LLM Engine

### **4.7 Ingestion API Component**

#### **4.7.1 Purpose**

Provides lightweight API for document ingestion.

#### **4.7.2 Functions**

- Document reception and validation

- Initial metadata extraction

- Document storage in MongoDB

- Extraction job queuing

- Input validation

#### **4.7.3 Interfaces**

- **Inbound**: API calls from document sources

- **Outbound**: Document storage to MongoDB

## **5. Data Architecture**

### **5.1 Data Model**

The system uses MongoDB's document-based schema to store extraction data with complete lineage.

#### **5.1.1 Document Collection**

{  
 "\_id": "ObjectId",  
 "documentId": "string",  
 "fileName": "string",  
 "fileType": "string",  
 "uploadDate": "date",  
 "status": "string",  
 "metadata": {  
 "source": "string",  
 "uploadedBy": "string",  
 "tags": ["string"]  
 },  
 "content": {  
 "raw": "string",  
 "processed": "string"  
 },  
 "extractionResults": [  
 {  
 "extractionId": "string",  
 "extractionDate": "date",  
 "extractionInstructions": "string",  
 "rawResults": "object",  
 "processedResults": "object",  
 "confidence": "number",  
 "humanCorrections": [  
 {  
 "correctionId": "string",  
 "correctionDate": "date",  
 "correctedBy": "string",  
 "originalValue": "string",  
 "correctedValue": "string",  
 "fieldPath": "string"  
 }  
 ]  
 }  
 ]  
}

#### **5.1.2 User Collection**

{  
 "\_id": "ObjectId",  
 "username": "string",  
 "email": "string",  
 "role": "string",  
 "permissions": ["string"],  
 "lastLogin": "date"  
}

#### **5.1.3 Extraction Template Collection**

{  
 "\_id": "ObjectId",  
 "templateName": "string",  
 "description": "string",  
 "documentTypes": ["string"],  
 "extractionFields": [  
 {  
 "fieldName": "string",  
 "fieldType": "string",  
 "required": "boolean",  
 "extractionPrompt": "string"  
 }  
 ],  
 "createdBy": "string",  
 "createdDate": "date",  
 "version": "number"  
}

### **5.2 Data Flow**

1. **Ingestion Flow**:

- Documents are submitted through the Ingestion API

- Documents are stored in MongoDB with metadata

- Extraction jobs are queued for processing

2. **Extraction Flow**:

- LLM Engine retrieves documents from MongoDB

- Langchain-based extraction processes the documents

- Extraction results are stored back in MongoDB with full lineage

3. **Human-in-the-Loop Flow**:

- UI displays extraction results

- Human reviewers can correct/validate extractions

- Corrections are stored in MongoDB with lineage

4. **Consumption Flow**:

- Downstream systems access extraction results via Consumption Layer APIs

- Results can be filtered by extraction ID or other metadata

### **5.3 Data Retention and Archiving**

- Active documents will be retained in the primary MongoDB collection

- Processed documents older than a configurable period will be archived to a separate collection

- Document retention policies will be configurable per document type

- Archiving jobs will run as scheduled batch processes

## **6. Interface Design**

### **6.1 User Interfaces**

#### **6.1.1 Document Upload Interface**

- Onboarding ICMP for file upload

- Batch upload capability

- Document type selection

- Initial metadata input

- Upload progress indication

#### **6.1.2 Extraction Review Interface**

- Side-by-side view of original document and extracted data

- Highlighting of extracted entities in the original document

- Confidence score visualization (Optional)

- Correction capability for extracted fields

- Approval/rejection workflow

#### **6.1.3 Dashboard Interface**

- Extraction metrics and KPIs

- Processing queue status

- System health indicators

- User activity logs

- Performance trends

### **6.2 API Interfaces**

#### **6.2.1 Ingestion API**

POST /api/v1/documents  
- Request: multipart/form-data with document file and metadata  
- Response: Document ID and status  
  
GET /api/v1/documents/{documentId}  
- Response: Document metadata and status

#### **6.2.2 Extraction API**

POST /api/v1/documents/{documentId}/extract  
- Request: Extraction parameters and template ID  
- Response: Extraction job ID and status  
  
GET /api/v1/extractions/{extractionId}  
- Response: Extraction results and status

#### **6.2.3 Consumption API**

GET /api/v1/extractions  
- Query Parameters: filters, pagination, sorting  
- Response: List of extraction results matching criteria  
  
GET /api/v1/extractions/{extractionId}/results  
- Response: Detailed extraction results with lineage

### **6.3 Internal Interfaces**

#### **6.3.1 Backend to LLM Engine Interface**

POST /internal/extract  
- Request: Document content, extraction parameters  
- Response: Extraction results with confidence scores

#### **6.3.2 Backend to Batch Processing Interface**

POST /internal/batch/schedule  
- Request: Batch job parameters  
- Response: Batch job ID and status

## **7. Non-Functional Requirements**

### **7.1 Performance**

- API response times under 2 seconds for standard operations

- Document processing throughput of at least 100 documents per minute

- Support for concurrent processing of multiple documents

- Scalable to handle peak loads with minimal performance degradation

### **7.2 Scalability**

- Horizontal scaling of all components

- Auto-scaling based on load metrics

- Support for distributed processing

- Efficient resource utilization

### **7.3 Availability**

- 99.9% uptime for the overall system

- Active-passive deployment for high availability

- Graceful degradation under load

- Automated failover mechanisms

### **7.4 Reliability**

- Complete document lineage for auditability

- Data consistency across system components

- Transactional integrity for critical operations

- Comprehensive error handling and recovery

### **7.5 Maintainability**

- Modular architecture for component-level updates

- Comprehensive logging and monitoring

- Automated testing for all components

- Clear documentation for maintenance procedures

### **7.6 Security**

- Role-based access control

- Data encryption at rest and in transit

- Secure API authentication and authorization

- Audit logging for security events

- Compliance with relevant security standards

## **8. Security Architecture**

### **8.1 Authentication and Authorization**

- OAuth 2.0 / JWT for API authentication

- Role-based access control for UI and API access

- Fine-grained permissions for document operations

- Integration with enterprise identity providers (optional)

### **8.2 Data Security**

- Encryption of sensitive data at rest

- TLS/SSL for all data in transit

- Secure handling of document content

- Data masking for sensitive information in logs

### **8.3 API Security**

- Rate limiting and throttling

- Input validation and sanitization

- Protection against common API attacks

- API keys and secrets management

### **8.4 Audit and Compliance**

- Comprehensive audit logging

- User activity tracking

- Security event monitoring

- Compliance reporting capabilities

## **9. Deployment Architecture**

### **9.1 Deployment Model**

The system will be deployed in an active-passive configuration across two OCP clusters for high availability.

### **9.2 Container Strategy**

- Each component will be containerized using Docker

- Container images will be stored in a private container registry

- Container health checks and readiness probes will be implemented

- Resource limits and requests will be defined for each container

### **9.3 CI/CD Pipeline**

- Source code management in Git

- Automated builds triggered by code commits

- Automated testing (unit, integration, system)

- Deployment automation using OpenShift Pipelines

- Environment promotion workflow (Dev → Test → Staging → Production)

### **9.4 Monitoring and Logging**

- Centralized logging using ELK stack or similar

- Metrics collection using Prometheus

- Visualization using Grafana

- Alerting based on predefined thresholds

- Distributed tracing for request flows

## **10. Appendices**

### **10.1 Glossary**

| **Term** | **Definition** |
| --- | --- |
| Document Extraction | The process of identifying and extracting structured information from unstructured documents |
| Agentic AI | AI systems that can act autonomously to achieve specific goals |
| Lineage | The complete history and provenance of data throughout its lifecycle |
| Human-in-the-Loop | A model that requires human interaction for decision making or validation |

### **10.2 References**

1. OpenShift Container Platform Documentation

2. MongoDB Documentation

3. Langchain Framework Documentation

4. React Documentation

5. Node.js Documentation

6. TIDE Architecture and Project Plan

### **10.3 Revision History**

| **Version** | **Date** | **Description** | **Author** |
| --- | --- | --- | --- |
| 1.0 | May 29, 2025 | Initial draft | TIDE Team |