

prithviraj

Intelligent Document Classifier and Router - Product Requirements Document

Try HackMD

Intelligent Document Classifier and Router - Product Requirements Document

1. Executive Summary

1.1 Product Vision

Eliminate the productivity drain of manual document management by intelligently classifying, routing, and organizing business documents with AI-powered automation and context-aware workflows.

1.2 Business Objectives

Productivity: Save 1 full workday per week per knowledge worker

Speed: 70% reduction in document routing time (3 days → 8 hours)

Accuracy: 95% correct document classification rate

Searchability: 80% reduction in time spent searching for documents

1.3 Target Market

Knowledge-intensive businesses (legal, consulting, finance)

Companies with document-heavy workflows

Organizations with compliance requirements

Remote/hybrid teams needing document collaboration

2. Product Overview

2.1 Core Value Proposition

AI-powered document intelligence that automatically understands, categorizes, and routes business documents to the right people at the right time with relevant context and actionable insights.

2.2 Key Features

Intelligent Classification: AI-powered document type and content analysis

Smart Routing: Context-aware assignment to teams and individuals

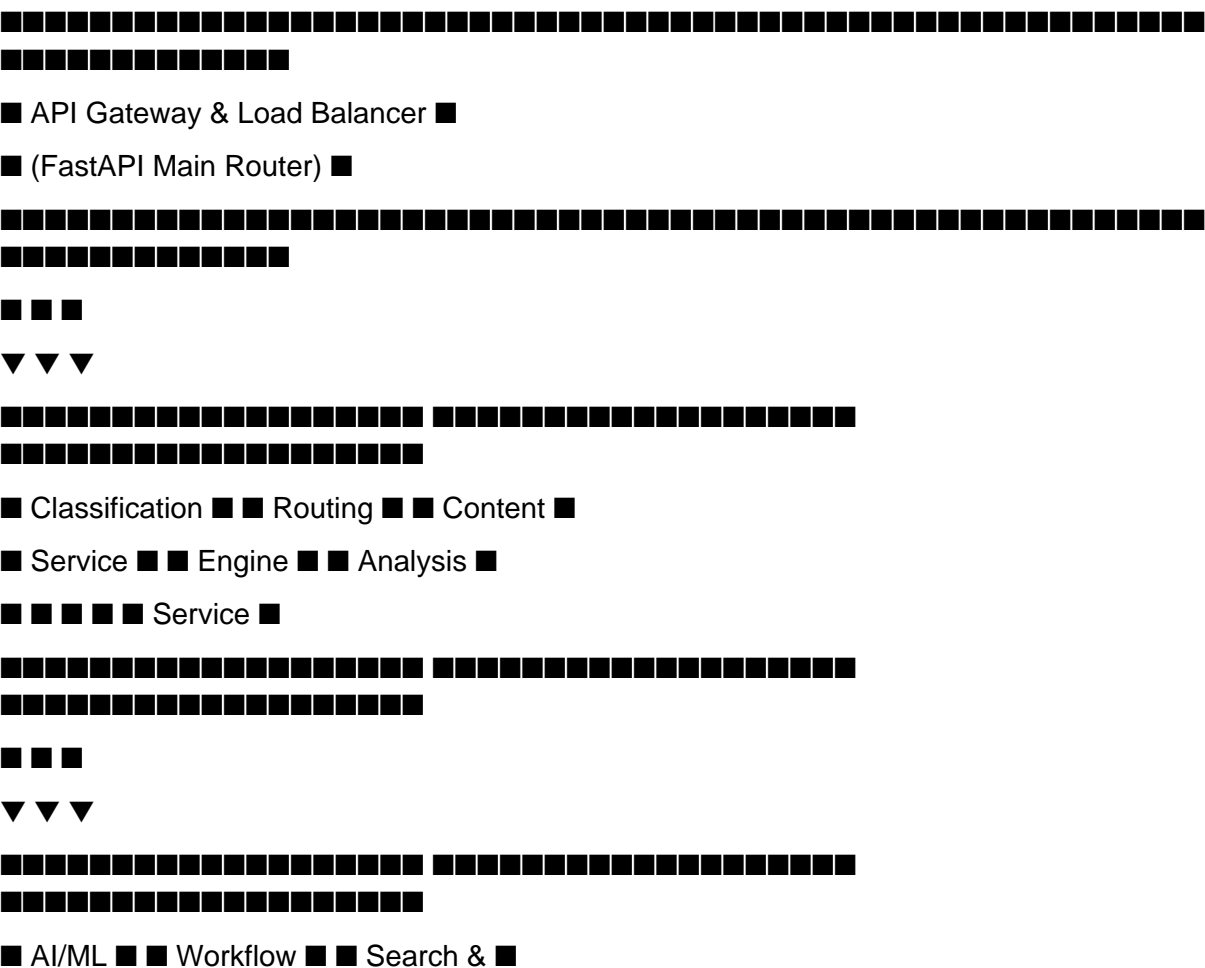
Content Extraction: Key information extraction and tagging

Workflow Integration: Seamless integration with existing business tools

Knowledge Discovery: AI-powered insights and document relationships

3. Microservice Architecture

3.1 High-Level Architecture Diagram



- ■ ■ ■ Real-time processing
- ■ ■ Content Analysis
- ■ ■ ■ Text extraction
- ■ ■ ■ OCR processing
- ■ ■ ■ Metadata extraction
- ■ ■ AI Classification Engine
- ■ ■ ■ Document type detection
- ■ ■ ■ Content categorization
- ■ ■ ■ Confidence scoring
- ■ ■ Learning Module
- ■ ■ User feedback integration
- ■ ■ Model fine-tuning
- ■ ■ Performance optimization

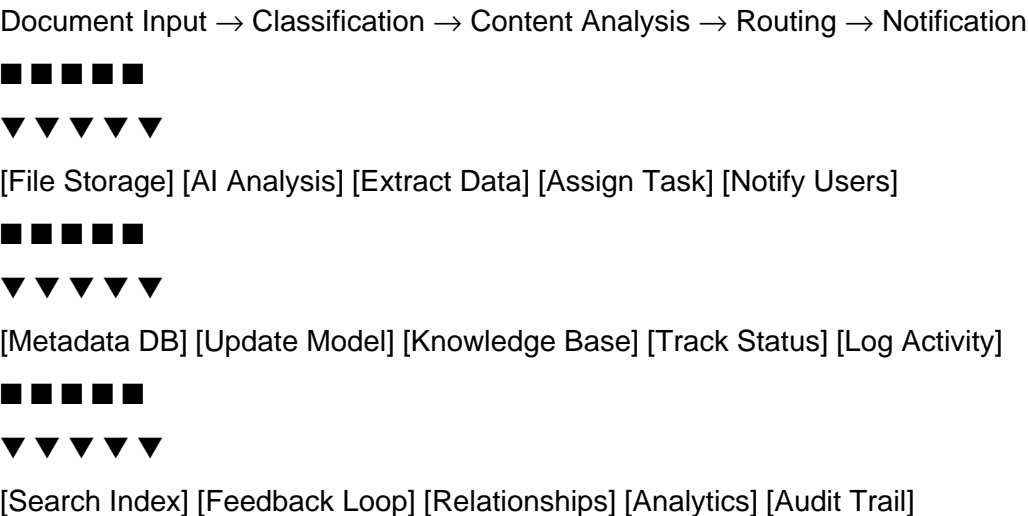
Routing Engine

- ■ ■ Rule Engine
- ■ ■ ■ Business rule evaluation
- ■ ■ ■ Priority assignment
- ■ ■ ■ Escalation logic
- ■ ■ Assignment Logic
- ■ ■ ■ Team/individual routing
- ■ ■ ■ Workload balancing
- ■ ■ ■ Expertise matching
- ■ ■ Context Analysis
- ■ ■ ■ Project association
- ■ ■ ■ Customer relationship
- ■ ■ ■ Historical patterns
- ■ ■ Decision Tracking
- ■ ■ Routing decisions log
- ■ ■ Performance metrics
- ■ ■ Optimization insights

Content Analysis Service

- Information Extraction
 - ■■■ Entity recognition (names, dates, amounts)
 - ■■■ Key phrase extraction
 - ■■■ Summary generation
- Relationship Mapping
 - ■■■ Document connections
 - ■■■ Reference detection
 - ■■■ Dependency analysis
- Compliance Scanning
 - ■■■ Sensitive data detection
 - ■■■ Regulatory compliance
 - ■■■ Risk assessment
- Insight Generation
 - Actionable insights
 - Trend analysis
 - Recommendation engine

3.3 Data Flow Diagram



4. Feature Specifications

4.1 Document Classification Features

4.1.1 Multi-Format Support

Document Types: PDF, Word, Excel, PowerPoint, images, text files

Email Integration: Direct email attachment processing

Drag & Drop: Browser-based file upload interface

Batch Processing: Multiple file handling

API Upload: Programmatic document submission

4.1.2 AI-Powered Classification

Document Types: Contracts, invoices, reports, correspondence, legal docs

Content Categories: Financial, legal, technical, marketing, HR

Industry-Specific: Healthcare, finance, legal, manufacturing classifications

Custom Categories: User-defined classification schemes

Confidence Levels: AI certainty scores for classifications

4.1.3 Content Understanding

Entity Extraction: People, organizations, dates, amounts, locations

Key Information: Critical data points and summaries

Language Detection: Multi-language document support

Sentiment Analysis: Document tone and sentiment scoring

Topic Modeling: Automatic subject identification

4.2 Intelligent Routing Features

4.2.1 Smart Assignment

Role-Based Routing: Route based on user roles and responsibilities

Expertise Matching: Match documents to subject matter experts

Workload Balancing: Distribute work evenly across team members

Priority-Based: High-priority documents to appropriate resources

Time Zone Awareness: Route to available team members

4.2.2 Business Rule Engine

Conditional Routing: If-then-else routing logic

Multi-Criteria: Route based on multiple document attributes

Escalation Rules: Automatic escalation for unhandled documents

Exception Handling: Special handling for edge cases

Approval Workflows: Multi-step approval processes

4.2.3 Context-Aware Processing

Project Association: Link documents to relevant projects

Customer Context: Associate with customer relationships

Historical Patterns: Learn from past routing decisions

Deadline Awareness: Consider time-sensitive requirements

Compliance Requirements: Route for regulatory compliance

4.3 Workflow Integration

4.3.1 Communication Integration

Slack Integration: Channel notifications and bot interactions

Microsoft Teams: Team collaboration and notifications

Email Notifications: Customizable email alerts

Mobile Push: Real-time mobile notifications

In-App Messaging: Internal messaging system

4.3.2 Task Management

Jira Integration: Automatic ticket creation and updates

Asana Integration: Task assignment and tracking

Trello Integration: Card creation and board management

Custom Webhooks: Integration with any REST API

Calendar Integration: Schedule document review sessions

4.4 Search & Discovery

4.4.1 Intelligent Search

Full-Text Search: Content-based document search

Semantic Search: Meaning-based search capabilities

Filter Options: Date, type, assignee, status filters

Faceted Search: Multiple filter combinations

Auto-Complete: Search suggestion and completion

4.4.2 Knowledge Discovery

Related Documents: Find similar or related content

Trend Analysis: Identify patterns in document types

Usage Analytics: Track document access and usage

Recommendation Engine: Suggest relevant documents

Knowledge Graphs: Visual relationship mapping

5. Technical Requirements

5.1 Performance Requirements

Processing Speed: <10 seconds per document classification

Throughput: 500+ documents per hour

Search Response: <1 second for search queries

Uptime: 99.5% service availability

Concurrent Users: Support 200+ simultaneous users

5.2 AI/ML Requirements

Classification Accuracy: >95% for standard document types

Model Training: Continuous learning from user feedback

Multi-Language: Support for 10+ major languages

Custom Models: Industry-specific model training

Confidence Thresholds: Configurable confidence levels

5.3 Integration Requirements

API Standards: RESTful APIs with comprehensive documentation

Real-time Events: WebSocket support for live updates

Webhook Support: Outbound event notifications

SSO Integration: SAML, OAuth 2.0 authentication

Data Export: Multiple format export capabilities

6. User Experience Design

6.1 Web Dashboard

Document Pipeline: Visual workflow status

Quick Actions: Common tasks and shortcuts

Analytics Dashboard: Usage and performance metrics

Configuration Panel: Rule and workflow management

6.2 Mobile Interface

Document Capture: Camera-based document upload

Quick Classification: Mobile document processing

Notification Center: All alerts and updates

Offline Capability: Basic functionality without internet

6.3 Admin Console

User Management: Role and permission administration

Rule Configuration: Business rule setup and management

Model Training: AI model management and tuning

Analytics & Reporting: Comprehensive usage analytics

7. Security & Compliance

7.1 Data Security

Encryption: End-to-end encryption for all documents

Access Control: Granular permission management

Audit Logging: Complete activity tracking

Data Residency: Configurable data storage locations

Backup & Recovery: Automated backup systems

7.2 Compliance Features

GDPR Compliance: Data protection and privacy controls

HIPAA Ready: Healthcare data handling capabilities

SOX Compliance: Financial document audit trails

Industry Standards: ISO 27001, SOC 2 compliance

Data Retention: Configurable retention policies

8. Success Metrics

8.1 Operational Metrics

Classification Accuracy: Percentage of correct classifications

Processing Time: Average time per document

User Adoption: Active users and feature usage

Search Success Rate: Successful search completion

8.2 Business Impact Metrics

Time Savings: Reduction in document handling time

Productivity Gains: Documents processed per user

Error Reduction: Decrease in misrouted documents

User Satisfaction: Net Promoter Score (NPS)

9. Implementation Roadmap

9.1 Phase 1 - Core Classification (2 Days POC)

Basic document upload and classification

Simple routing rules

OpenAI integration for content analysis

Basic notification system

9.2 Phase 2 - Enhanced Intelligence (Week 1-2)

Advanced classification models

Complex routing engine

Integration with Slack/Teams

Search capabilities

9.3 Phase 3 - Enterprise Features (Week 3-4)

Multi-tenant architecture

Advanced analytics

Custom model training

Compliance features

9.4 Phase 4 - AI Enhancement (Month 2)

Machine learning optimization

Predictive routing

Advanced insights

Knowledge graph features

10. Risk Assessment

10.1 Technical Risks

AI Accuracy: Variable classification performance

Scalability: High volume processing challenges

Integration Complexity: Multiple system integrations

Data Privacy: Sensitive document handling

10.2 Mitigation Strategies

Human-in-the-Loop: Manual review for low confidence

Microservice Architecture: Independent scaling

API-First Design: Standardized integrations

Privacy by Design: Built-in data protection

11. Competitive Advantages

11.1 Technical Differentiators

AI-First Approach: Native AI capabilities vs. bolt-on features

Real-time Processing: Immediate classification and routing

Context Awareness: Understanding document relationships

Learning System: Continuous improvement from usage

11.2 Business Differentiators

Rapid Deployment: 2-day POC to production in weeks

Cost Effective: 80% lower than enterprise solutions

User-Friendly: Intuitive interface requiring minimal training

Flexible Integration: Works with existing tool ecosystem

12. Conclusion

The Intelligent Document Classifier and Router addresses a critical productivity challenge in modern businesses. By leveraging AI and microservice architecture, it provides immediate value while scaling to enterprise requirements, positioning WNS as a leader in the intelligent document processing market.

Technical Design Document: Intelligent Document Classifier & Router

1. System Architecture Overview

API Gateway

Classification Service

Routing Engine

Content Analysis

AI/ML Service

Workflow Integration

Search & Discovery

Shared Data Layer

Document Storage

Metadata DB

Permissions DB

2. Microservice Specifications

Service Technology Core Responsibilities Key Interfaces

API Gateway FastAPI, JWT Request routing, authentication, rate limiting REST endpoints, WebSocket

Classification Python, spaCy Document type detection, content categorization File upload API, confidence scoring

Routing Engine Python, Rule Engine Context-aware assignment, workload balancing JSON rule definitions, priority queues

Content Analysis Pandas, NumPy Entity extraction, relationship mapping Dataframe processing, metadata enrichment

AI/ML Service scikit-learn Model training/inference, feedback integration Model versioning endpoint

Workflow Integration Python, Webhooks Jira/Slack connectivity, notification delivery Custom adapter pattern

Search & Discovery SQLAlchemy, Full-Text Semantic search, knowledge graphs Query parser, ranking algorithms

3. Data Flow Sequence

4. Database Schema Design

Documents Table

Column Type Description

doc_id UUID Unique document identifier

original_name VARCHAR(255) Original filename

storage_path TEXT S3 path/object reference

doc_type VARCHAR(50) Contract/Invoice/Report etc.

confidence FLOAT Classification confidence score

Metadata Table

Column Type Description

meta_id SERIAL Auto-increment ID

doc_id UUID Foreign key to documents

key_entities JSONB Extracted names/dates/amounts

related_docs UUID[] Linked document references

risk_score FLOAT Compliance risk assessment

Routing Rules Table

Column Type Description

rule_id SERIAL Auto-increment ID

condition JSONB IF-THEN logic definition

assignee VARCHAR(100) User/team assignment target

priority INT 1-5 urgency scale

5. CI/CD Pipeline Design

6. Non-Functional Requirements

Category Requirement Implementation Strategy

Performance <10s/doc processing Async processing, message queues

Scalability 500 docs/hour throughput Kubernetes auto-scaling

Security End-to-end encryption TLS 1.3, AES-256 at rest

Reliability 99.5% uptime Health checks, circuit breakers

Maintainability 85% test coverage Pytest, mutation testing

Compliance GDPR/HIPAA readiness Audit logs, data anonymization hooks

8. Failure Mode Analysis

Failure Point Mitigation Strategy Recovery Process

OCR failure Fallback to metadata-based routing Human review queue

Classification low confidence Secondary model verification User feedback loop

Routing rule conflict Priority-based arbitration Dead-letter queue monitoring

Service downtime Kubernetes liveness probes Auto-container restart

Data corruption Cryptographic hash verification S3 version rollback

9. Monitoring & Observability

Key Metrics Dashboard

Metric Source Alert Threshold

Docs processed/min Prometheus <10

Classification accuracy Model feedback <90%

Avg. routing time Application logs >30s

Error rate ELK stack >5%

Container memory cAdvisor >80% utilization

10. Security Controls

Layer Controls

Network VPC isolation, Security Groups

Application JWT authentication, Input validation

Data AES-256 encryption, RBAC permissions

Infrastructure Immutable containers, Signed images

Compliance Audit trails, Data retention policies

This design enables implementation of core classification/routing functionality within 2 days using specified technologies, while establishing foundations for enterprise-grade document intelligence capabilities.

Initial Folder Structure & Docker Setup Guide

Tech Stack: Python, FastAPI, Docker Compose, PostgreSQL, SQLAlchemy, RabbitMQ, GitHub Actions

Root Directory Structure

intelligent_document_router/

■■■■ .github/

■ ■■■■ workflows/

■ ■■■■ ci-cd.yml # GitHub Actions pipeline

■■■■ docker-compose.yml # Main compose file

■■■■ .env # Environment variables

■■■■ docs/ # Architecture diagrams

■■■■ libs/ # Shared Python libraries

■ ■■■■ database/ # SQLAlchemy base models

- ■ ■ ■ ■ utils/ # Common utilities
- ■ ■ ■ ■ microservices/
- ■ ■ ■ ■ api_gateway/ # Entry point
- ■ ■ ■ ■ Dockerfile
- ■ ■ ■ ■ app/
- ■ ■ ■ ■ main.py # FastAPI routes
- ■ ■ ■ ■ dependencies.py
- ■ ■ ■ ■ requirements.txt
- ■
- ■ ■ ■ ■ classification/ # Doc classification
- ■ ■ ■ ■ Dockerfile
- ■ ■ ■ ■ app/
- ■ ■ ■ ■ classifier.py # spaCy model
- ■ ■ ■ ■ schemas.py
- ■ ■ ■ ■ requirements.txt
- ■
- ■ ■ ■ ■ routing_engine/ # Assignment logic
- ■ ■ ■ ■ Dockerfile
- ■ ■ ■ ■ rules/ # Business rules
- ■ ■ ■ ■ app/
- ■ ■ ■ ■ requirements.txt
- ■
- ■ ■ ■ ■ content_analysis/ # Entity extraction
- ■ ■ ■ ■ Dockerfile
- ■ ■ ■ ■ app/
- ■ ■ ■ ■ requirements.txt
- ■
- ■ ■ ■ ■ workflow_integration/ # Jira/Slack
- ■ ■ ■ ■ Dockerfile
- ■ ■ ■ ■ adapters/ # Jira/Slack clients
- ■ ■ ■ ■ requirements.txt
-

- infrastructure/
- ■■■ db/ # PostgreSQL setup
- ■ ■■■ Dockerfile
- ■ ■■■ init.sql # DB schema
- ■■■ rabbitmq/ # Message broker
- ■■■ storage/ # MinIO (S3-compatible)
-
- tests/ # Pytest suites
- unit/
- integration/

Key Files Implementation

1. docker-compose.yml

version: '3.8'

services:

api_gateway:

build: ./microservices/api_gateway

ports:

- "8000:8000"

env_file: .env

depends_on:

- rabbitmq
- db

classification:

build: ./microservices/classification

env_file: .env

routing_engine:

build: ./microservices/routing_engine

env_file: .env

content_analysis:

build: ./microservices/content_analysis

env_file: .env

workflow_integration:

build: ./microservices/workflow_integration

env_file: .env

db:

build: ./infrastructure/db

env_file: .env

volumes:

- pgdata:/var/lib/postgresql/data

rabbitmq:

image: rabbitmq:3-management

ports:

- "5672:5672"

- "15672:15672"

minio:

image: minio/minio

command: server /data

volumes:

- storage:/data

ports:

- "9000:9000"

volumes:

pgdata:

storage:

2. Sample Dockerfile (for classification service)

FROM python:3.9-slim

WORKDIR /app

COPY ./requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

Download spaCy model during build

RUN python -m spacy download en_core_web_sm

COPY ./app ./app

CMD ["uvicorn", "app.classifier:app", "--host", "0.0.0.0", "--port", "8001"]

3. .env File

PostgreSQL

POSTGRES_HOST=db

POSTGRES_USER=admin

POSTGRES_PASSWORD=secret

POSTGRES_DB=document_db

RabbitMQ

RABBITMQ_HOST=rabbitmq

RABBITMQ_QUEUE=document_queue

MinIO Storage

MINIO_ENDPOINT=minio:9000

MINIO_ACCESS_KEY=minioadmin

MINIO_SECRET_KEY=minioadmin

4. Database Initialization (infrastructure/db/init.sql)

CREATE TABLE documents (

id UUID PRIMARY KEY,

filename VARCHAR(255) NOT NULL,

filepath VARCHAR(255) NOT NULL,

```
doc_type VARCHAR(50),  
confidence FLOAT  
);
```

```
CREATE TABLE routing_rules (  
id SERIAL PRIMARY KEY,  
condition JSONB NOT NULL,  
assignee VARCHAR(100) NOT NULL  
);
```

Development Workflow

Initialize Services:

```
docker-compose up --build -d
```

Access Services:

API Gateway: <http://localhost:8000/docs>

RabbitMQ Console: <http://localhost:15672> (guest/guest)

MinIO Console: <http://localhost:9000> (minioadmin/minioadmin)

CI/CD Pipeline (.github/workflows/ci-cd.yml):

name: CI/CD Pipeline

on: [push]

jobs:

build-test:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Build and test

run: |

```
docker-compose -f docker-compose.test.yml up --build --abort-on-container-exit
```

Microservice Communication Pattern:

```
# api_gateway/app/main.py
```

```
from fastapi import FastAPI, UploadFile
```

```
import httpx
```

```
app = FastAPI()
```

```
@app.post("/upload")
```

```
async def upload(file: UploadFile):
```

```
# 1. Save to MinIO
```

```
# 2. Send message to RabbitMQ
```

```
async with httpx.AsyncClient() as client:
```

```
await client.post("http://classification:8001/process", json=file_metadata)
```

Key Design Decisions

Decoupled Communication:

RabbitMQ for async inter-service messaging

HTTP for synchronous requests (service discovery via Docker DNS)

Shared Libraries:

Central libs/database with SQLAlchemy base models

Common validation schemas in libs/schemas.py

Stateless Services:

Each microservice connects directly to DB/MinIO

No inter-service dependencies during initialization

DevOps Integration:

Pre-configured Jenkinsfile in each service directory

GitHub Actions for automated testing

Docker Hub auto-builds on Git tags

POC-Scalable Architecture:

Local MinIO instead of AWS S3

SQLite option for lightweight testing

Mock AI models (spaCy) instead of cloud APIs

Next Steps for Implementation

Generate boilerplate code:

```
# For each microservice  
mkdir -p microservice/{service}/app  
touch microservice/{service}/Dockerfile  
touch microservice/{service}/requirements.txt
```

Start core services:

```
docker-compose up db rabbitmq minio -d
```

Implement classification service first:

Document upload endpoint

spaCy text processing

DB storage logic

This structure allows parallel development by new hires, with each team owning one microservice while learning the full stack.

Intelligent Document Classifier and Router - Implementation Todo

Project Overview

AI-powered document intelligence that automatically understands, categorizes, and routes business documents to the right people at the right time with relevant context and actionable insights.

Phase 1: Core Classification (2 Days POC) - PRIORITY**Day 1: Core Services Setup**

1.1 Project Structure Creation

Create root directory structure

Set up microservices folders (api_gateway, classification, routing_engine, content_analysis, workflow_integration)

Create shared libraries folder (libs)

Set up infrastructure folder (db, rabbitmq, storage)

Create tests folder structure

1.2 Docker & Infrastructure Setup

Create docker-compose.yml with all services

Set up PostgreSQL database with init.sql

Configure RabbitMQ message broker

Set up MinIO (S3-compatible storage)

Create .env file with all environment variables

Test infrastructure startup

1.3 Database Schema Implementation

Create documents table (UUID, filename, filepath, doc_type, confidence)

Create metadata table (meta_id, doc_id, key_entities, related_docs, risk_score)

Create routing_rules table (rule_id, condition, assignee, priority)

Set up SQLAlchemy models in libs/database

Test database connectivity

1.4 API Gateway Service

Create FastAPI main application

Implement JWT authentication

Set up request routing and rate limiting

Create document upload endpoint

Implement health check endpoints

Add API documentation (Swagger/OpenAPI)

1.5 Classification Service MVP

Set up spaCy for text processing

Implement document type detection (PDF, DOC, TXT, images)

Create basic classification logic (contracts, invoices, reports)

Add confidence scoring

Implement file storage to MinIO

Create classification API endpoints

Day 2: Integration & Deployment

2.1 Content Analysis Service

Implement text extraction from various formats

Add OCR processing for images

Create entity extraction (names, dates, amounts)

Implement key phrase extraction

Add summary generation

Test with sample documents

2.2 Routing Engine

Create rule engine for business logic

Implement basic routing rules (role-based, expertise matching)

Add workload balancing logic

Create priority assignment system

Implement routing decision tracking

Test routing with sample scenarios

2.3 Workflow Integration

Set up Slack integration (notifications)

Implement Jira integration (ticket creation)

Create email notification system

Add webhook support for external systems

Test notification delivery

2.4 Message Queue Integration

Implement RabbitMQ producers/consumers

Set up async document processing pipeline

Create error handling and retry logic

Add dead letter queue for failed messages

Test message flow between services

2.5 Basic Search & Discovery

Implement full-text search

Add basic filtering (date, type, assignee)

Create document relationship mapping

Add simple analytics dashboard

Test search functionality

2.6 POC Testing & Validation

End-to-end testing with sample documents

Performance testing (<10 seconds per document)

Load testing (500+ documents per hour)

User acceptance testing

Document POC results and findings

Phase 2: Enhanced Intelligence (Week 1-2)

Week 1: Advanced Features

3.1 Advanced Classification Models

Implement multi-language support

Add industry-specific classifications

Create custom category training

Implement model fine-tuning

Add confidence threshold configuration

3.2 Complex Routing Engine

Implement conditional routing (if-then-else)

Add multi-criteria routing logic

Create escalation rules

Implement approval workflows

Add exception handling

3.3 Enhanced Content Analysis

Add sentiment analysis

Implement topic modeling

Create compliance scanning

Add risk assessment scoring

Implement relationship mapping

Week 2: Integration & Optimization

3.4 Advanced Integrations

Microsoft Teams integration

Asana/Trello integration

Calendar integration

Mobile push notifications

Custom webhook framework

3.5 Search & Discovery Enhancement

Implement semantic search

Add faceted search capabilities

Create knowledge graphs

Implement recommendation engine

Add trend analysis

Phase 3: Enterprise Features (Week 3-4)

Week 3: Multi-tenancy & Security

4.1 Multi-tenant Architecture

Implement tenant isolation

Add tenant-specific configurations

Create tenant management UI

Implement resource quotas

Add tenant analytics

4.2 Advanced Security

Implement end-to-end encryption

Add granular access control

Create audit logging system

Implement data residency controls

Add backup and recovery

Week 4: Compliance & Analytics

4.3 Compliance Features

GDPR compliance implementation

HIPAA readiness features

SOX compliance audit trails

Data retention policies

Compliance reporting

4.4 Advanced Analytics

Comprehensive usage analytics

Performance monitoring dashboard

Predictive analytics

Business intelligence reports

Custom reporting engine

Phase 4: AI Enhancement (Month 2)

Month 2: Machine Learning & Optimization

5.1 Machine Learning Optimization

Implement continuous learning

- Add model versioning
- Create A/B testing framework
- Implement model performance monitoring
- Add automated model retraining

5.2 Predictive Features

- Predictive routing
- Document priority prediction
- Workload forecasting
- Risk prediction
- Trend forecasting

5.3 Advanced Insights

- Document relationship discovery
- Knowledge graph expansion
- Intelligent recommendations
- Process optimization suggestions
- Business intelligence insights
- Technical Implementation Details

Current Status: ■ READY TO START

Product Requirements Document (plan1.md)

- Technical Architecture Design
- Implementation Roadmap
- Project Structure Creation
- Docker Setup
- Database Schema
- First Microservice

Next Immediate Actions:

- Create the complete project folder structure
- Set up docker-compose.yml with all services

Implement database schema and models

Create API Gateway service

Build Classification Service MVP

Success Criteria:

Document classification accuracy >95%

Processing time <10 seconds per document

Throughput 500+ documents per hour

99.5% service availability

Support 200+ concurrent users

Risk Mitigation:

Human-in-the-loop for low confidence classifications

Fallback mechanisms for service failures

Comprehensive error handling and logging

Performance monitoring and alerting

Security and compliance by design

Last Updated: [Current Date]

Current Phase: Phase 1 - Core Classification

Next Milestone: Day 1 - Core Services Setup

#_____

Intelligent Document Classifier and Router - Architecture & Folder Structure Guide

This guide explains the folder structure, code organization, and extension points for the Intelligent Document Classifier and Router, based on the requirements in plan1.md.

■ Root Directory Structure

intelligent_document_router/

■■■■ .github/workflows/ # CI/CD pipeline definitions (GitHub Actions)

■■■■ docs/ # Architecture diagrams, technical docs, PRDs

■■■■ libs/ # Shared Python libraries (models, utils)

- ■■■■ database/ # SQLAlchemy models, DB connection logic
- ■■■■ utils/ # Common utilities (logging, helpers)
- microservices/ # All microservices (each in its own folder)
- ■■■■ api_gateway/ # FastAPI entry point, auth, routing
- ■■■■ classification/ # AI-powered document classification
- ■■■■ routing_engine/ # Smart document routing logic
- ■■■■ content_analysis/ # Entity extraction, content analysis
- ■■■■ workflow_integration/ # Slack/Jira/email integrations
- infrastructure/ # Infrastructure as code (DB, MQ, storage)
- ■■■■ db/ # PostgreSQL Dockerfile, init.sql
- ■■■■ rabbitmq/ # RabbitMQ config (if needed)
- ■■■■ storage/ # MinIO/S3 config (if needed)
- tests/ # Unit and integration tests
- ■■■■ unit/ # Unit tests for each service
- ■■■■ integration/ # Integration/E2E tests
- docker-compose.yml # Main compose file for all services
- .env # Environment variables for all services
- README.md # Project overview and quickstart
- todo.md # Implementation progress tracker
- SETUP_COMPLETE.md # Setup summary and next steps
- Folder-by-Folder Explanation

.github/workflows/

Purpose: CI/CD pipeline definitions (e.g., build, test, deploy)

Add: YAML files for GitHub Actions, e.g., ci-cd.yml

docs/

Purpose: Architecture diagrams, PRDs, technical documentation

Add: Markdown docs, Mermaid diagrams, onboarding guides

libs/

Purpose: Shared code used by multiple microservices

database/ : SQLAlchemy models, DB session/connection logic

utils/ : Logging, error handling, helper functions

Add: New shared models, utility functions, or adapters

microservices/

Purpose: All business logic, each service in its own folder

api_gateway/ : FastAPI app, authentication, main API endpoints

app/: Main FastAPI app, routers, middleware, dependencies

routers/: Add new API endpoints (e.g., /documents, /auth)

middleware/: Custom middleware (logging, rate limiting)

dependencies.py: Dependency injection (auth, DB session)

main.py: FastAPI app entry point

adapters/: (Optional) HTTP clients for other services

rules/: (Optional) Business rule definitions

requirements.txt, Dockerfile: Service dependencies and build

classification/ : Document type detection, spaCy/NLP logic

routing_engine/ : Rule engine, workload balancing, assignment

content_analysis/ : Entity extraction, key phrase, summary

workflow_integration/ : Slack, Jira, email, webhooks

Add: New microservices for additional features

infrastructure/

Purpose: Infrastructure as code for DB, MQ, storage

db/ : PostgreSQL Dockerfile, schema init.sql

rabbitmq/ : RabbitMQ config (if custom setup needed)

storage/ : MinIO/S3 config (if custom setup needed)

Add: Custom Dockerfiles, config scripts

tests/

Purpose: All automated tests

unit/ : Unit tests for each microservice

integration/ : Integration/E2E tests across services

Add: New test modules as you add features

■■■ Where to Add What Code

New API endpoint?

Add a new file in microservices/api_gateway/app/routers/ and register it in main.py.

New database model?

Add to libs/database/models.py and run migrations/init.

New microservice?

Create a new folder in microservices/, add app/, Dockerfile, requirements.txt.

Shared logic?

Add to libs/ (e.g., libs/utils/ for helpers, libs/database/ for DB logic).

Infrastructure change?

Update infrastructure/ and docker-compose.yml.

New test?

Add to tests/unit/ or tests/integration/ as appropriate.

■ Onboarding Guide for New Developers

Read README.md and ARCHITECTURE_GUIDE.md for project overview.

Clone the repo and copy .env.example to .env.

Start infrastructure:

```
docker-compose up db rabbitmq minio redis -d
```

Run the API Gateway locally:

```
cd microservices/api_gateway
```

```
pip install -r requirements.txt
```

```
uvicorn app.main:app --reload
```

Explore the codebase:

API endpoints: microservices/api_gateway/app/routers/

DB models: libs/database/models.py

Shared logic: libs/

Add new features/tests as described above

Run tests:

```
pytest
```

Check progress in todo.md and update as you complete tasks.

■ Extending the System

Add new document types: Update classification logic in classification/ and models in libs/database/models.py.

Add new routing rules: Update routing_engine/ and libs/database/models.py.

Integrate new tools (Slack, Jira, etc): Add adapters in workflow_integration/.

Improve search/analytics: Extend content_analysis/ and add endpoints to API Gateway.

■ Reference: plan1.md

All folder and service responsibilities are mapped directly from the PRD and technical design in plan1.md.

For detailed feature specs, see plan1.md and README.md.

For any questions, see the docs/ folder or contact the project maintainers.

■■■ Example: Creating an API Endpoint in Each Microservice

Below are minimal examples for adding a simple /ping endpoint to each microservice. This helps new developers quickly understand where to add code and how to register routes.

1. API Gateway (microservices/api_gateway)

File: microservices/api_gateway/app/routers/ping.py

```
from fastapi import APIRouter  
router = APIRouter()
```

```
@router.get("/ping")
```

```
def ping():
```

```
    return {"message": "pong from API Gateway"}
```

Register in app/main.py:

```
from app.routers import ping  
app.include_router(ping.router, prefix="/ping", tags=["ping"])
```

2. Classification Service (microservices/classification)

File: microservices/classification/app/main.py

```
from fastapi import FastAPI  
app = FastAPI()
```

```
@app.get("/ping")
```

```
def ping():
```

```
    return {"message": "pong from Classification Service"}
```

3. Routing Engine (microservices/routing_engine)

File: microservices/routing_engine/app/main.py

```
from fastapi import FastAPI  
app = FastAPI()
```

```
@app.get("/ping")
```

```
def ping():
```

```
return {"message": "pong from Routing Engine"}
```

4. Content Analysis (microservices/content_analysis)

File: microservices/content_analysis/app/main.py

```
from fastapi import FastAPI  
app = FastAPI()
```

```
@app.get("/ping")
```

```
def ping():
```

```
return {"message": "pong from Content Analysis Service"}
```

5. Workflow Integration (microservices/workflow_integration)

File: microservices/workflow_integration/app/main.py

```
from fastapi import FastAPI  
app = FastAPI()
```

```
@app.get("/ping")
```

```
def ping():
```

```
return {"message": "pong from Workflow Integration Service"}
```

How to test:

Start the relevant service (e.g., with uvicorn app.main:app --reload or via Docker Compose)

Visit <http://localhost:/ping> in your browser or use curl

Tip:

For more complex APIs, create a routers/ directory in each service and organize endpoints as in the API Gateway example.

Always register new routers in your FastAPI main.py using `app.include_router(...)`.

Last changed by