perplexity

continue

Continuing from CLI Tools section...

```
# cli/ingest.py (continued)

    with open(file, 'rb') as f:
        try:
            result = service.upload_document(f, metadata, replace_existing=replace)
            click.echo(click.style("✓ Success!", fg="green"))
            click.echo(f"  Document ID: {result['doc_id']}")
            click.echo(f"  Chunks ingested: {result['chunks_ingested']}")
            click.echo(f"  File hash: {result['file_hash'][:16]}...")
        except Exception as e:
            click.echo(click.style(f"✗ Error: {str(e)}", fg="red"))
            raise

if __name__ == '__main__':
    ingest()
```

Usage:

```
python cli/ingest.py \
  --domain hr \
  --file ./docs/leave_policy.pdf \
  --doc-id HR-POLICY-2025-001 \
  --title "Employee Leave Policy 2025" \
  --doc-type policy \
  --uploader-id admin@company.com \
  --replace
```

14.3 Query CLI

```
# cli/query.py

import click
import json
from core.services.document_service import DocumentService

@click.command()
@click.option('--domain', required=True, help='Domain ID')
@click.option('--query', required=True, help='Query text')
@click.option('--strategy', default=None, help='Retrieval strategy (hybrid, vector_simila
```

```
@click.option('--top-k', default=5, help='Number of results')
@click.option('--filter-doc-type', multiple=True, help='Filter by doc type')
@click.option('--include-deprecated', is_flag=True, help='Include deprecated docs')
@click.option('--output', type=click.Choice(['text', 'json']), default='text')
def query(domain, query, strategy, top_k, filter_doc_type, include_deprecated, output):
    """Query the RAG system."""
    click.echo(f"Querying domain '{domain}': {query}")

    service = DocumentService(domain)

    # Build filters
    filters = {}
    if filter_doc_type:
        filters['doc_type'] = list(filter_doc_type)

    try:
        results = service.query(
            query_text=query,
            strategy=strategy,
            metadata_filters=filters if filters else None,
            top_k=top_k,
            include_deprecated=include_deprecated
        )

        if output == 'json':
            click.echo(json.dumps(results, indent=2, default=str))
        else:
            click.echo(click.style(f"\n✓ Found {len(results)} results\n", fg="green"))
            for i, result in enumerate(results, 1):
                click.echo(click.style(f"Result {i}:", fg="cyan", bold=True))
                click.echo(f"  Chunk ID: {result['chunk_id']}")
                click.echo(f"  Score: {result['score']:.4f}")
                click.echo(f"  Doc Type: {result['metadata'].get('doc_type')}")
                click.echo(f"  Text: {result['chunk_text'][:200]}...")
                if 'dense_score' in result:
                    click.echo(f"  Dense: {result['dense_score']:.4f}, Sparse: {result['s
                click.echo()

    except Exception as e:
        click.echo(click.style(f"✗ Error: {str(e)}", fg="red"))
        raise

if __name__ == '__main__':
    query()
```

Usage:

```
python cli/query.py \
  --domain hr \
  --query "How many vacation days?" \
  --strategy hybrid \
  --top-k 5 \
  --filter-doc-type policy
```

14.4 Management CLI

```
# cli/manage.py

import click
from datetime import datetime
from core.services.document_service import DocumentService

@click.group()
def cli():
    """Document management commands."""
    pass

@cli.command()
@click.option('--domain', required=True, help='Domain ID')
@click.option('--doc-id', required=True, help='Document ID')
@click.option('--reason', required=True, help='Deprecation reason')
def deprecate(domain, doc_id, reason):
    """Deprecate a document."""
    click.echo(f"Deprecating document: {doc_id}")

    service = DocumentService(domain)

    try:
        service.deprecate_document(doc_id, reason)
        click.echo(click.style("✓ Document deprecated successfully", fg="green"))
    except Exception as e:
        click.echo(click.style(f"✗ Error: {str(e)}", fg="red"))
        raise

@cli.command()
@click.option('--domain', required=True, help='Domain ID')
@click.option('--include-deprecated', is_flag=True)
def list_docs(domain, include_deprecated):
    """List all documents in domain."""
    click.echo(f"Listing documents in domain: {domain}")

    service = DocumentService(domain)

    try:
        docs = service.list_documents(include_deprecated=include_deprecated)

        click.echo(click.style(f"\n✓ Found {len(docs)} documents\n", fg="green"))

        for doc in docs:
            status = "DEPRECATED" if doc.get('deprecated') else "ACTIVE"
            color = "red" if doc.get('deprecated') else "green"

            click.echo(click.style(f"[{status}]", fg=color) + f" {doc['doc_id']}")
            click.echo(f"  Title: {doc.get('title')}")
            click.echo(f"  Type: {doc.get('doc_type')}")
            click.echo(f"  Chunks: {doc.get('chunk_count')}")
            click.echo(f"  Uploaded: {doc.get('upload_timestamp')}")
            click.echo()

    except Exception as e:
```

```
        click.echo(click.style(f"✗ Error: {str(e)}", fg="red"))
        raise

@cli.command()
@click.option('--domain', required=True, help='Domain ID')
@click.option('--doc-id', required=True, help='Document ID')
def info(domain, doc_id):
    """Get detailed info about a document."""
    service = DocumentService(domain)

    try:
        info = service.get_document_info(doc_id)

        click.echo(click.style(f"\nDocument Information: {doc_id}\n", fg="cyan", bold=Tru
        click.echo(f"Title: {info.get('title')}")
        click.echo(f"Domain: {info.get('domain')}")
        click.echo(f"Type: {info.get('doc_type')}")
        click.echo(f"Author: {info.get('author')}")
        click.echo(f"Version: {info.get('version')}")
        click.echo(f"Authority: {info.get('authority_level')}")
        click.echo(f"Status: {info.get('review_status')}")
        click.echo(f"Chunks: {info.get('chunk_count')}")
        click.echo(f"Uploaded: {info.get('upload_timestamp')}")
        click.echo(f"Deprecated: {info.get('deprecated')}")
        if info.get('deprecated'):
            click.echo(f"Deprecated Date: {info.get('deprecated_date')}")
            click.echo(f"Reason: {info.get('deprecation_reason')}")

    except Exception as e:
        click.echo(click.style(f"✗ Error: {str(e)}", fg="red"))
        raise

if __name__ == '__main__':
    cli()
```

Usage:

```
# Deprecate document
python cli/manage.py deprecate \
  --domain hr \
  --doc-id HR-POLICY-2025-001 \
  --reason "Superseded by 2026 policy"

# List documents
python cli/manage.py list-docs --domain hr

# Get document info
python cli/manage.py info --domain hr --doc-id HR-POLICY-2025-001
```

14.5 Evaluation CLI

```
# cli/evaluate.py

import click
import yaml
from tabulate import tabulate
from tests.golden_qa.evaluate import evaluate_golden_qa
from core.services.document_service import DocumentService

@click.command()
@click.option('--domain', required=True, help='Domain ID')
@click.option('--qa-file', required=True, type=click.Path(exists=True), help='Golden QA Y
@click.option('--strategy', default=None, help='Retrieval strategy to test')
def evaluate(domain, qa_file, strategy):
    """Evaluate retrieval quality using Golden QA sets."""
    click.echo(f"Evaluating domain '{domain}' with QA set: {qa_file}")

    service = DocumentService(domain)

    # Load QA data
    with open(qa_file) as f:
        qa_data = yaml.safe_load(f)

    click.echo(f"Loaded {len(qa_data['questions'])} questions\n")

    # Evaluate each question
    results = []
    total_recall = 0
    total_mrr = 0
    passed_count = 0

    for qa in qa_data['questions']:
        query_results = service.query(
            query_text=qa['question'],
            strategy=strategy,
            top_k=10
        )

        # Compute metrics
        recall_at_5 = compute_recall(
            retrieved=[r['chunk_id'] for r in query_results[:5]],
            expected=qa['expected_chunks']
        )

        mrr = compute_mrr(
            retrieved=[r['chunk_id'] for r in query_results],
            expected=qa['expected_chunks']
        )

        passed = (
            recall_at_5 >= qa.get('min_recall_at_5', 0.8) and
            mrr >= qa.get('min_mrr', 0.7)
        )

        total_recall += recall_at_5
```

```
        total_mrr += mrr
        if passed:
            passed_count += 1

        status = click.style("✓ PASS", fg="green") if passed else click.style("✗ FAIL",

        results.append([
            qa['id'],
            qa['question'][:50] + "...",
            f"{recall_at_5:.3f}",
            f"{mrr:.3f}",
            status
        ])

    # Display results table
    headers = ["ID", "Question", "Recall@5", "MRR", "Status"]
    click.echo(tabulate(results, headers=headers, tablefmt="grid"))

    # Summary statistics
    num_questions = len(qa_data['questions'])
    avg_recall = total_recall / num_questions
    avg_mrr = total_mrr / num_questions
    pass_rate = (passed_count / num_questions) * 100

    click.echo(click.style("\n=== Summary ===", bold=True))
    click.echo(f"Total Questions: {num_questions}")
    click.echo(f"Passed: {passed_count}")
    click.echo(f"Pass Rate: {pass_rate:.1f}%")
    click.echo(f"Avg Recall@5: {avg_recall:.3f}")
    click.echo(f"Avg MRR: {avg_mrr:.3f}")

    if pass_rate >= 80:
        click.echo(click.style("\n✓ Evaluation PASSED (≥80% pass rate)", fg="green", bol
    else:
        click.echo(click.style("\n✗ Evaluation FAILED (<80% pass rate)", fg="red", bold=

def compute_recall(retrieved, expected):
    """Recall = (retrieved ∩ expected) / expected"""
    if not expected:
        return 1.0
    intersection = set(retrieved) & set(expected)
    return len(intersection) / len(expected)

def compute_mrr(retrieved, expected):
    """Mean Reciprocal Rank."""
    for i, chunk_id in enumerate(retrieved, start=1):
        if chunk_id in expected:
            return 1.0 / i
    return 0.0

if __name__ == '__main__':
    evaluate()
```

Usage:

```
python cli/evaluate.py \
  --domain hr \
  --qa-file tests/golden_qa/hr_domain_qa.yaml \
  --strategy hybrid
```

15. Migration Plan

15.1 Phase 1 to Phase 2 Migration

Pre-Migration Checklist

- [ ] Backup all vector stores
- [ ] Backup all configuration files
- [ ] Document current system state (domains, document counts)
- [ ] Test Phase 2 code in isolated environment
- [ ] Plan rollback procedure

Step-by-Step Migration

Step 1: Backup (30 minutes)

```
#!/bin/bash
# backup.sh

DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="./backups/phase1_$DATE"

echo "Creating backup: $BACKUP_DIR"
mkdir -p $BACKUP_DIR

# Backup vector stores
cp -r data/chromadb $BACKUP_DIR/chromadb
cp -r data/pinecone $BACKUP_DIR/pinecone 2>/dev/null || true

# Backup configs
cp -r configs $BACKUP_DIR/configs

# Backup code
git rev-parse HEAD > $BACKUP_DIR/git_commit.txt

echo "Backup complete: $BACKUP_DIR"
```

Step 2: Deploy Phase 2 Code (1 hour)

```
#!/bin/bash
# deploy_phase2.sh
```

```
echo "Deploying Phase 2 code..."

# Checkout Phase 2 branch
git checkout phase-2

# Install new dependencies
pip install -r requirements.txt

# Verify installation
python -c "from core.services.document_service import DocumentService; print('✓ Service
python -c "from core.retrievals.hybrid_retrieval import HybridRetrieval; print('✓ Hybrid
python -c "from core.metadata_models import ChunkMetadata; print('✓ Metadata models OK')

echo "Phase 2 deployment complete"
```

Step 3: Update Configurations (30 minutes)

```
#!/bin/bash
# update_configs.sh

echo "Updating configuration files..."

# Backup old configs
cp configs/global_config.yaml configs/global_config.yaml.phase1

# Deploy new configs
cp configs/phase2/global_config.yaml configs/global_config.yaml

# Update domain configs
for domain in configs/domains/*.yaml; do
    echo "Updating $domain"
    # Add new retrieval strategies section
    # Add new metadata settings
    # (Use sed/yq or manual editing)
done

echo "Configuration update complete"
```

Step 4: Metadata Migration (2-4 hours depending on data volume)

```
#!/usr/bin/env python
# migrate_metadata.py

"""
Migrate Phase 1 metadata to Phase 2 schema.
Adds missing fields with sensible defaults.
"""

import logging
from datetime import datetime
from core.services.document_service import DocumentService

logging.basicConfig(level=logging.INFO)
```

```
logger = logging.getLogger("MetadataMigration")

def migrate_domain(domain_id: str):
    """Migrate metadata for one domain."""
    logger.info(f"Migrating domain: {domain_id}")

    service = DocumentService(domain_id)
    vector_store = service.pipeline.vector_store

    # Get all documents
    all_docs = vector_store.get_all_documents()
    logger.info(f"Found {len(all_docs)} chunks to migrate")

    migrated_count = 0

    for doc in all_docs:
        chunk_id = doc['id']
        metadata = doc['metadata']

        # Add missing Phase 2 fields
        updates = {}

        if 'deprecated' not in metadata:
            updates['deprecated'] = False
        if 'deprecated_date' not in metadata:
            updates['deprecated_date'] = None
        if 'deprecation_reason' not in metadata:
            updates['deprecation_reason'] = None
        if 'embedding_model_name' not in metadata:
            updates['embedding_model_name'] = 'unknown'
        if 'embedding_dimension' not in metadata:
            updates['embedding_dimension'] = 384  # Default
        if 'chunking_strategy' not in metadata:
            updates['chunking_strategy'] = 'recursive'
        if 'processing_timestamp' not in metadata:
            updates['processing_timestamp'] = datetime.utcnow()
        if 'authority_level' not in metadata:
            updates['authority_level'] = 'draft'
        if 'review_status' not in metadata:
            updates['review_status'] = 'pending'

        # Apply updates if any
        if updates:
            vector_store.update_chunk_metadata(chunk_id, updates)
            migrated_count += 1

            if migrated_count % 100 == 0:
                logger.info(f"Migrated {migrated_count}/{len(all_docs)} chunks")

    logger.info(f"Migration complete for {domain_id}: {migrated_count} chunks updated")

def main():
    """Migrate all domains."""
    domains = ['hr', 'finance', 'legal', 'engineering']  # Update with your domains

    for domain in domains:
```

```
        try:
            migrate_domain(domain)
        except Exception as e:
            logger.error(f"Migration failed for {domain}: {e}")
            raise

    logger.info("All domains migrated successfully")

if __name__ == '__main__':
    main()
```

Run migration:

```
python migrate_metadata.py
```

Step 5: Build BM25 Indices (1-2 hours)

```
#!/usr/bin/env python
# build_bm25_indices.py

"""
Build BM25 indices for hybrid retrieval.
Must be run after Phase 2 deployment.
"""

import logging
import pickle
from core.services.document_service import DocumentService
from core.retrievals.bm25_retrieval import BM25Retrieval

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger("BM25IndexBuilder")

def build_index_for_domain(domain_id: str):
    """Build BM25 index for one domain."""
    logger.info(f"Building BM25 index for: {domain_id}")

    service = DocumentService(domain_id)
    vector_store = service.pipeline.vector_store

    # Get all documents
    all_docs = vector_store.get_all_documents()

    corpus = [doc['chunk_text'] for doc in all_docs]
    doc_ids = [doc['chunk_id'] for doc in all_docs]

    logger.info(f"Building index with {len(corpus)} documents")

    # Build BM25 index
    bm25_index = BM25Retrieval(corpus, doc_ids)

    # Save index
    index_path = f"./data/bm25_indices/{domain_id}_bm25.pkl"
```

```
    with open(index_path, 'wb') as f:
        pickle.dump(bm25_index, f)

    logger.info(f"Index saved: {index_path}")

def main():
    """Build indices for all domains."""
    import os
    os.makedirs("./data/bm25_indices", exist_ok=True)

    domains = ['hr', 'finance', 'legal', 'engineering']

    for domain in domains:
        try:
            build_index_for_domain(domain)
        except Exception as e:
            logger.error(f"Index build failed for {domain}: {e}")
            raise

    logger.info("All BM25 indices built successfully")

if __name__ == '__main__':
    main()
```

Run index builder:

```
python build_bm25_indices.py
```

Step 6: Validation & Testing (2 hours)

```
#!/bin/bash
# validate_migration.sh

echo "Validating Phase 2 migration..."

# Run unit tests
pytest tests/unit/ -v

# Run integration tests
pytest tests/integration/ -v

# Run Golden QA evaluation
python cli/evaluate.py --domain hr --qa-file tests/golden_qa/hr_domain_qa.yaml

# Test sample queries
python cli/query.py --domain hr --query "vacation policy" --strategy hybrid

# Verify metadata
python -c "
from core.services.document_service import DocumentService
service = DocumentService('hr')
docs = service.list_documents()
print(f'Found {len(docs)} documents')
```

```
for doc in docs[:3]:
    print(f'  {doc[\"doc_id\"]}: deprecated={doc.get(\"deprecated\")}')
"

echo "Validation complete"
```

Step 7: Rollback Procedure (if needed)

```
#!/bin/bash
# rollback.sh

echo "Rolling back to Phase 1..."

# Find latest backup
LATEST_BACKUP=$(ls -td backups/phase1_* | head -1)
echo "Restoring from: $LATEST_BACKUP"

# Restore vector stores
rm -rf data/chromadb
cp -r $LATEST_BACKUP/chromadb data/chromadb

# Restore configs
rm -rf configs
cp -r $LATEST_BACKUP/configs configs

# Restore code
git checkout $(cat $LATEST_BACKUP/git_commit.txt)

# Reinstall dependencies
pip install -r requirements.txt

echo "Rollback complete. System restored to Phase 1."
```

15.2 Migration Timeline

Phase	Duration	Critical Path
Pre-Migration	1 day	Testing in staging, backup verification
Backup	30 min	Vector stores, configs, code state
Code Deployment	1 hour	Git checkout, dependency install
Config Update	30 min	YAML updates, validation
Metadata Migration	2-4 hours	Depends on data volume
BM25 Index Build	1-2 hours	Depends on corpus size
Testing & Validation	2 hours	All test suites, Golden QA
Monitoring	24 hours	Watch for issues post-deployment
Total	**~2-3 days**	Including validation period

15.3 Post-Migration Checklist

- [ ] All unit tests passing
- [ ] All integration tests passing
- [ ] Golden QA evaluation meets targets (≥80% pass rate)
- [ ] Sample queries returning expected results
- [ ] All domains accessible via UI and CLI
- [ ] Deprecation workflow tested and functional
- [ ] Hybrid retrieval operational (both dense and sparse)
- [ ] Metadata migration complete (all chunks have Phase 2 fields)
- [ ] BM25 indices built and loadable
- [ ] Logs show no errors or warnings
- [ ] Performance metrics within acceptable range
- [ ] Rollback procedure documented and tested

16. Code Organization

16.1 Directory Structure (Phase 2)

```
multi-domain-rag/
├── configs/
│   ├── global_config.yaml
│   ├── domains/
│   │   ├── hr_domain.yaml
│   │   ├── finance_domain.yaml
│   │   ├── legal_domain.yaml
│   │   └── engineering_domain.yaml
│   └── templates/
│       ├── semantic_template.yaml
│       └── recursive_template.yaml
│
├── core/
│   ├── __init__.py
│   ├── services/                    # NEW: Service Layer
│   │   ├── __init__.py
│   │   └── document_service.py
│   ├── pipeline/
│   │   ├── __init__.py
│   │   └── document_pipeline.py
│   ├── factories/
│   │   ├── __init__.py
│   │   ├── chunking_factory.py
│   │   ├── embedding_factory.py
│   │   ├── retrieval_factory.py     # Enhanced with hybrid
│   │   └── vector_store_factory.py
│   ├── retrievals/                  # NEW: Retrieval Strategies
│   │   ├── __init__.py
```

```
│   │   ├── vector_similarity_retrieval.py
│   │   ├── bm25_retrieval.py        # NEW
│   │   └── hybrid_retrieval.py      # NEW
│   ├── chunking/
│   │   ├── __init__.py
│   │   ├── chunking_interface.py
│   │   ├── recursive_chunker.py
│   │   └── semantic_chunker.py
│   ├── embeddings/
│   │   ├── __init__.py
│   │   ├── embedding_interface.py
│   │   ├── sentence_transformer_embeddings.py
│   │   └── gemini_embeddings.py
│   ├── vectorstores/
│   │   ├── __init__.py
│   │   ├── vector_store_interface.py
│   │   ├── chromadb_store.py
│   │   └── pinecone_store.py
│   ├── utils/                       # NEW: Utility Functions
│   │   ├── __init__.py
│   │   ├── validation.py            # NEW
│   │   └── hashing.py               # NEW
│   ├── metadata_models.py           # Enhanced with Phase 2 fields
│   ├── config_manager.py
│   └── domain_config.py
│
├── utils/
│   └── fileparsers/
│       ├── __init__.py
│       ├── pdf_processor.py         # Enhanced
│       ├── docx_processor.py        # Enhanced
│       └── txt_processor.py         # Enhanced
│
├── cli/                             # NEW: CLI Tools
│   ├── __init__.py
│   ├── ingest.py
│   ├── query.py
│   ├── manage.py
│   └── evaluate.py
│
├── tests/
│   ├── unit/
│   │   ├── test_services/           # NEW
│   │   │   └── test_document_service.py
│   │   ├── test_pipeline/
│   │   │   └── test_document_pipeline.py
│   │   ├── test_factories/
│   │   │   ├── test_chunking_factory.py
│   │   │   ├── test_embedding_factory.py
│   │   │   ├── test_retrieval_factory.py
│   │   │   └── test_vector_store_factory.py
│   │   ├── test_retrievals/         # NEW
│   │   │   ├── test_bm25_retrieval.py
│   │   │   └── test_hybrid_retrieval.py
│   │   └── test_utils/              # NEW
│   │       ├── test_validation.py
```

```
│   │       └── test_hashing.py
│   ├── integration/
│   │   ├── test_end_to_end_ingestion.py
│   │   ├── test_end_to_end_retrieval.py
│   │   └── test_multi_strategy_retrieval.py
│   ├── golden_qa/                    # NEW
│   │   ├── hr_domain_qa.yaml
│   │   ├── finance_domain_qa.yaml
│   │   └── evaluate.py
│   └── fixtures/
│       ├── sample.pdf
│       ├── sample.docx
│       └── sample.txt
│
├── data/
│   ├── chromadb/
│   ├── pinecone/
│   ├── bm25_indices/                 # NEW
│   └── uploads/
│
├── logs/
│   └── rag_system.log
│
├── backups/                          # NEW
│   └── phase1_YYYYMMDD_HHMMSS/
│
├── app.py                            # Refactored: thin UI layer only
├── requirements.txt
├── .env.example
├── .gitignore
├── README.md
└── ARCHITECTURE.md                   # NEW: Architecture documentation
```

16.2 Import Structure Rules

UI Layer (`app.py`):

```
# ✅ ALLOWED
from core.services.document_service import DocumentService

# ❌ FORBIDDEN
from core.pipeline.document_pipeline import DocumentPipeline  # NO
from core.factories.embedding_factory import EmbeddingFactory  # NO
from core.vectorstores.chromadb_store import ChromaDBStore  # NO
```

Service Layer (`document_service.py`):

```
# ✅ ALLOWED
from core.pipeline.document_pipeline import DocumentPipeline
from core.utils.validation import validate_file_type
from core.metadata_models import ChunkMetadata
```

```
# ✗ FORBIDDEN
from core.factories import *  # Should use via pipeline only
```

Pipeline Layer (`document_pipeline.py`):

```
# ✓ ALLOWED
from core.factories.chunking_factory import ChunkingFactory
from core.factories.embedding_factory import EmbeddingFactory
from core.factories.vector_store_factory import VectorStoreFactory
from core.factories.retrieval_factory import RetrievalFactory
```

17. Developer Guidelines

17.1 UI Development Rules (CRITICAL)

Golden Rule: NEVER add business logic to UI layer

Checklist for UI Code Review

- [ ] Handler function ≤ 20 lines
- [ ] Only calls service layer methods (no pipeline/factory imports)
- [ ] No conditional business logic (no if/else for file types, etc.)
- [ ] No data transformations
- [ ] No validation logic (done in service layer)
- [ ] Only display formatting and user input capture

Examples

✓ CORRECT UI Handler:

```
def upload_handler(file, metadata):
    """UI handler - routing only."""
    try:
        result = DocumentService.upload_document(file, metadata)
        return f"✓ Success: {result['chunks_ingested']} chunks ingested"
    except ValidationError as e:
        return f"✗ Validation error: {str(e)}"
    except Exception as e:
        return f"✗ Error: {str(e)}"
```

✗ WRONG UI Handler:

```
def upload_handler(file, metadata):
    """WRONG - business logic in UI."""
    # ✗ File validation in UI
    if not file.name.endswith(('.pdf', '.docx', '.txt')):
```

```
        return "Invalid file type"

    # ✗ Metadata validation in UI
    if not metadata.get('doc_id'):
        return "Missing doc_id"

    # ✗ Direct pipeline call
    pipeline = DocumentPipeline(config)
    result = pipeline.process_document(file, metadata)

    return f"Success: {result}"
```

17.2 Service Layer Development Rules

Responsibilities:

- **ALL** input validation
- **ALL** business rule enforcement
- **ALL** orchestration logic
- Error transformation (technical → user-friendly)
- Logging at business event level

Guidelines:

- Every public method must validate inputs
- Use custom exceptions (ValidationError, DocumentNotFoundError)
- Log at INFO level for business events
- Delegate processing to pipeline
- Never call factories directly (pipeline uses factories)

Template:

```
def service_method(self, param1, param2):
    """
    Service method template.

    1. Validate inputs
    2. Log business event
    3. Delegate to pipeline
    4. Handle errors
    5. Return result
    """
    # Step 1: Validate
    if not param1:
        raise ValidationError("param1 required")

    # Step 2: Log
    logger.info(f"Business event: param1={param1}, param2={param2}")
```

```
    # Step 3: Delegate
    try:
        result = self.pipeline.do_something(param1, param2)
    except Exception as e:
        # Step 4: Handle errors
        logger.exception(f"Pipeline failed: {e}")
        raise ProcessingError(f"Failed to process: {str(e)}")

    # Step 5: Return
    logger.info(f"Business event complete: {result}")
    return result
```

17.3 Pipeline Development Rules

Responsibilities:

- Orchestrate workflows (chunk → embed → store)
- Use factories to create components
- Attach metadata to chunks
- Execute retrieval strategies
- No validation (done by service layer)

Guidelines:

- All components created via factories
- Use config exclusively for decisions
- Log at DEBUG/INFO for technical events
- Return structured results (dicts with keys)

17.4 Factory Development Rules

Responsibilities:

- Instantiate implementations based on config
- Handle provider-specific logic
- Read environment variables for secrets
- Validate config structure

Guidelines:

- Registry pattern: `_available_implementations` dict
- Clear error messages for unknown providers
- No business logic
- Stateless (no instance variables except registry)

Template:

```
class MyFactory:
    """Factory for creating X instances."""

    _available_implementations = {
        "impl1": Implementation1,
        "impl2": Implementation2,
    }

    @staticmethod
    def create_instance(config):
        """Create instance from config."""
        provider = config.get("provider")

        impl_cls = MyFactory._available_implementations.get(provider)
        if not impl_cls:
            raise ValueError(
                f"Unknown provider '{provider}'. "
                f"Available: {list(MyFactory._available_implementations.keys())}"
            )

        # Extract config params
        param1 = config.get("param1")
        param2 = config.get("param2")

        # Instantiate
        return impl_cls(param1=param1, param2=param2)
```

17.5 Testing Requirements

Coverage Targets:

- Service layer: 90%
- Pipeline layer: 85%
- Factories: 90%
- Utilities: 95%
- UI layer: Not required (thin routing only)

Test Structure:

- One test file per module
- Test class per class under test
- Test method per public method

Naming Convention:

```
def test_<method_name>_<scenario>_<expected_behavior>():
    """Test that method behaves correctly when scenario occurs."""
    pass

# Examples:
```

```
def test_upload_document_with_invalid_file_type_raises_validation_error():
def test_query_with_deprecated_false_filters_deprecated_documents():
def test_hybrid_retrieval_with_alpha_07_combines_scores_correctly():
```

Mock External Dependencies:

- Mock vector stores in service tests
- Mock pipeline in service tests
- Mock factories in pipeline tests
- Use real implementations in integration tests

18. Security & Compliance

18.1 File Upload Security

Validation:

- Strict file type checking via `allowed_file_types`
- File size limits via `max_file_size_mb`
- File name sanitization

Future Enhancements:

- Malware scanning integration
- File content type verification (not just extension)
- Sandbox execution for parsers

18.2 Data Privacy

PII Handling:

- Detect PII in metadata (future)
- Optional PII redaction (future)
- Compliance with GDPR/CCPA

Data Retention:

- Configurable retention policies
- Automatic archival of old documents
- Secure deletion workflows

Audit Trail:

- Log all uploads with uploader_id
- Log all queries with user context

- Log all deprecations with reason

18.3 Access Control (Future)

Authentication:

- SSO integration
- User authentication required

Authorization:

- Role-based access control (RBAC)
- Domain-level permissions
- Document-level permissions

19. Monitoring & Observability

19.1 Structured Logging

Log Format:

```
{
  "timestamp": "2025-11-24T13:00:00Z",
  "level": "INFO",
  "logger": "DocumentService",
  "event": "document_upload",
  "doc_id": "HR-POLICY-2025-001",
  "domain": "hr",
  "uploader_id": "admin@company.com",
  "chunks_ingested": 42,
  "file_hash": "a3b2c1d4e5f6...",
  "status": "success"
}
```

Implementation:

```
import logging
import json

class StructuredLogger:
    def __init__(self, name):
        self.logger = logging.getLogger(name)

    def log_event(self, level, event, **kwargs):
        log_data = {
            "event": event,
            **kwargs
        }
        self.logger.log(level, json.dumps(log_data))
```

```
# Usage
logger = StructuredLogger("DocumentService")
logger.log_event(logging.INFO, "document_upload",
                 doc_id="HR-001", chunks_ingested=42)
```

19.2 Metrics to Track

System Metrics:

- Total documents ingested
- Total queries processed
- Average query latency (P50, P95, P99)
- Error rate by component
- Active domains

Quality Metrics:

- Recall@K per domain (from Golden QA)
- Mean Reciprocal Rank (MRR)
- User satisfaction scores (future: thumbs up/down)
- Answer accuracy

Business Metrics:

- Documents per domain
- Queries per domain
- User adoption rate
- Most queried topics

19.3 Alerting (Future)

Error Alerts:

- Pipeline failure rate > 5%
- Vector store connection failures
- API key expiration

Performance Alerts:

- Query latency P95 > 1s
- Ingestion throughput < 1 doc/min

Quality Alerts:

- Recall@10 drops below 0.7

- MRR drops below 0.6

20. Future Enhancements (Phase 3+)

20.1 Multi-Modal Support

Features:

- Image extraction and OCR
- Table parsing and structure preservation
- Chart and diagram understanding
- Audio transcription

20.2 Advanced Retrieval

Features:

- Query expansion with LLMs
- Re-ranking with cross-encoders
- Contextual embeddings (per domain)
- Learned sparse representations

20.3 User Experience

Features:

- Conversational interface with history
- Citation highlighting in source docs
- Feedback loops (thumbs up/down)
- Query suggestions
- Related questions

20.4 Enterprise Features

Features:

- SSO integration (SAML, OAuth)
- Role-based access control (RBAC)
- Multi-tenancy support
- API rate limiting
- Audit logs export

20.5 Analytics & Insights

Features:

- Usage dashboards (per domain, per user)
- Quality metrics visualization
- A/B testing framework
- Query analytics (top queries, failed queries)
- Document popularity tracking

21. Appendices

Appendix A: Complete Code Templates

See sections above for:

- Complete `metadata_models.py` (Section 4.1)
- Complete `document_service.py` (Section 6.1)
- Complete `document_pipeline.py` (Section 7.1)
- Complete `bm25_retrieval.py` (Section 8.3)
- Complete `hybrid_retrieval.py` (Section 8.4)
- Complete validation utilities (Section 10.1)

Appendix B: Configuration Examples

See sections above for:

- Enhanced global config (Section 12.1)
- Domain-specific config (Section 12.2)

Appendix C: Testing Examples

See sections above for:

- Unit test examples (Section 13.1)
- Integration test examples (Section 13.2)
- Golden QA format (Section 13.3)

Appendix D: CLI Usage Examples

See section 14 for complete CLI documentation and usage examples.

Appendix E: Migration Scripts

See section 15 for complete migration procedures and scripts.

Document Sign-Off

Phase 2 Complete When:

- [ ] All code modules enhanced per specifications
- [ ] Service layer operational with zero UI business logic
- [ ] Hybrid retrieval functional and tested
- [ ] Enhanced metadata tracking in place
- [ ] All factories enhanced
- [ ] File processing with validation working
- [ ] 80%+ test coverage achieved
- [ ] Golden QA sets show ≥15% improvement
- [ ] CLI tools operational
- [ ] Migration from Phase 1 successful
- [ ] Documentation complete
- [ ] Stakeholder approval received

Document Version: 2.0 Final
Last Updated: November 24, 2025
Next Review: December 8, 2025

END OF PHASE 2 IMPLEMENTATION GUIDE

This comprehensive Phase 2 documentation incorporates:
✅ Critical "Zero Business Logic in UI" principle throughout
✅ Complete service layer specification
✅ Hybrid retrieval implementation details
✅ Enhanced metadata schema with all Phase 2 fields
✅ Migration plan from Phase 1 to Phase 2
✅ CLI tools for all operations
✅ Testing strategy with Golden QA sets
✅ Developer guidelines enforcing architectural principles
✅ Code organization with clear layer boundaries
✅ Complete working code examples for all components

The document is production-ready and can guide your team through Phase 2 implementation with clear separation of concerns and architectural excellence.