MES/SCADA RAG System - Vibe Coding Guide 🚀

The ultimate development philosophy and standards for building production-ready industrial RAG systems

🎯 THE VIBE

We build systems that work in the real world, not demos that impress in meetings.

- **Architecture First**: No code without clear design. Think twice, code once.
- **Domain-Driven**: Start from business logic, not cool technology
- **Contract-First**: API endpoints defined before implementation
- **Production-Ready**: Every line of code should be ready for 24/7 industrial environments
- **Documentation-Driven**: Future developers (including yourself) will thank you

🏗️ SYSTEM OVERVIEW

📊 Manufacturing Hierarchy (ISA-95 inspired, customer flexible)
↓
📄 Document Management (PDF, DOCX, XLSX processing)
↓
🧠 RAG Engine (Local embeddings + Vector search)
↓
💬 Chat Interface (Conversational AI for operators)

Stack: FastAPI + PostgreSQL + Qdrant + Ollama + Vanilla JS + Docker Compose

🎨 CODE STYLE - NON-NEGOTIABLE

Python Standards

python

```python
# ✅ Good - Type hints mandatory
def process_document(
    file_path: str,
    metadata: Dict[str, Any],
    node_ids: List[int]
) -> Optional[Document]:
    """Process document with proper error handling."""
    pass

# ❌ Bad - No types, unclear purpose
def process(file, data, ids):
    pass
```

Naming Conventions

```python
# Variables & functions: snake_case
user_documents = get_user_documents()

# Classes: PascalCase
class DocumentProcessor:
    pass

# Constants: UPPER_SNAKE_CASE
MAX_FILE_SIZE = 50 * 1024 * 1024  # 50MB

# Private methods: leading underscore
def _internal_helper():
    pass
```

Error Handling - Industrial Grade

```python
# ✅ Custom exceptions with context
class DocumentProcessingError(Exception):
    def __init__(self, message: str, file_path: str, original_error: Exception = None):
        self.file_path = file_path
        self.original_error = original_error
        super().__init__(message)

# ✅ Structured logging
logger.error(
    "Document processing failed",
    document_id=document_id,
    error=str(e),
    operation="process_document"
)
```

📁 PROJECT STRUCTURE - SACRED GEOMETRY

```
src/
├── api/                # FastAPI routes
│   └── v1/
│       ├── hierarchy.py    # Manufacturing hierarchy endpoints
│       ├── documents.py    # Document management
│       ├── search.py       # RAG search endpoints
│       └── auth.py         # Authentication
├── core/               # Business logic (domain layer)
│   ├── hierarchy/
│   ├── documents/
│   ├── search/
│   └── auth/
├── models/             # SQLAlchemy models
├── schemas/            # Pydantic models
├── services/           # External service integrations
├── utils/              # Utilities & helpers
└── config.py           # Configuration management
```

🗄️ DATABASE DESIGN PRINCIPLES

Core Tables

```sql
-- Flexible hierarchy (not strict ISA-95)
hierarchy_nodes:
- id, name, code, description
- parent_id (self-reference)
- level_type (varchar, not enum - customer flexibility)
- custom_attributes (JSON for extensions)

-- Document management
documents:
- id, filename, file_path, content_hash
- metadata (JSON), created_at, updated_at
- processing_status, content_extracted

-- Many-to-many relationships
document_node_associations:
- document_id, node_id, relationship_type
```

Database Operations - Always Safe

python

```python
# ✅ Dependency injection + proper error handling
def get_document_by_id(db: Session, document_id: int) -> Optional[Document]:
    try:
        return db.query(Document).filter(Document.id == document_id).first()
    except SQLAlchemyError as e:
        logger.error(f"Database error fetching document {document_id}: {e}")
        raise DatabaseError(f"Failed to fetch document: {e}")

# ✅ Context manager for sessions
@contextmanager
def get_db_session() -> Session:
    db = SessionLocal()
    try:
        yield db
        db.commit()
    except Exception:
        db.rollback()
        raise
    finally:
        db.close()
```

🤖 RAG PIPELINE - THE MAGIC

Document Processing Flow

```python
# 1. Text Extraction (format-aware)
PDF → PyMuPDF (fast, reliable)
DOCX → python-docx
XLSX → pandas + openpyxl
TXT → charset detection

# 2. Semantic Chunking
- Chunk size: 512-1024 tokens
- Overlap: 50-100 tokens
- Preserve document structure metadata

# 3. Embeddings Generation
- Local Ollama (privacy + cost control)
- Model: sentence-transformers/all-MiniLM-L6-v2
- Fallback to cloud embeddings if needed

# 4. Vector Storage
- Qdrant with metadata filtering
- Hierarchical context preservation
```

Search Algorithm

```python
def rag_search(query: str, user_hierarchy_context: List[int]) -> SearchResults:
    """
    1. Query preprocessing (cleaning, expansion)
    2. Generate query embeddings
    3. Vector similarity search in Qdrant
    4. Apply hierarchy filters based on user permissions
    5. Rank results by relevance + context
    6. Format response with citations
    """
    pass
```

🔐 SECURITY - ENTERPRISE GRADE

Authentication Strategy

python

```python
# Multi-provider support
providers = {
    'local': LocalAuthProvider(),      # Username/password fallback
    'saml': SAMLProvider(),            # Enterprise SSO
    'oidc': OIDCProvider(),            # Modern SSO
    'ldap': ActiveDirectoryProvider() # Legacy AD
}

# JWT + Redis backing (revocation capability)
JWT_SECRET_KEY = "your-secret-key"
REDIS_SESSION_PREFIX = "session:"
```

Authorization Model

python

```python
# Role-based access control
roles = {
    'system_admin': ['*'],  # Full access
    'hierarchy_manager': ['hierarchy.*', 'documents.read'],
    'document_manager': ['documents.*', 'hierarchy.read'],
    'user': ['documents.read', 'search.*']  # Based on hierarchy assignment
}
```

🐋 DEPLOYMENT - ONE COMMAND MAGIC

Docker Compose Services

yaml

```yaml
services:
  frontend:    # Nginx + static files
  api:         # FastAPI application
  postgres:    # Metadata database
  qdrant:      # Vector database
  redis:       # Cache & sessions
  ollama:      # Local LLM for embeddings
  nginx:       # Reverse proxy
```

File Storage Structure

```
/data/uploads/
  /{year}/      # 2025/
    /{month}/   # 01/
      /{hash}/  # abc123.../
        file.pdf      # original filename
        meta.json     # extracted metadata
```

🧪 TESTING PHILOSOPHY

Test Pyramid

```python
# Unit Tests - Business logic
def test_hierarchy_node_creation():
    node = create_hierarchy_node("Plant A", "plant", parent=None)
    assert node.name == "Plant A"
    assert node.level_type == "plant"

# Integration Tests - API endpoints
def test_document_upload_endpoint(client, auth_headers):
    response = client.post("/api/v1/documents/",
                           files={"file": test_pdf},
                           headers=auth_headers)
    assert response.status_code == 201

# E2E Tests - Critical workflows
def test_complete_rag_workflow():
    # Upload document → Process → Search → Verify results
    pass
```

📊 PERFORMANCE TARGETS

- **Response Time**: < 3s for RAG queries (95th percentile)
- **Throughput**: 100+ concurrent users
- **Document Processing**: < 30s per document
- **System Uptime**: > 99.5%
- **Search Accuracy**: > 90% relevant results

🚨 COMMON PITFALLS TO AVOID

❌ Don't Do This

python

```python
# No error handling
def process_file(file):
    return extract_text(file)

# Magic numbers everywhere
if file_size > 52428800:  # What is this number?
    raise Exception("Too big")

# Hardcoded paths
with open("/var/data/file.txt"):
    pass
```

✅ Do This Instead

python

```python
# Proper error handling with context
def process_file(file_path: str) -> ProcessingResult:
    try:
        return extract_text(file_path)
    except FileNotFoundError:
        raise DocumentNotFoundError(f"File not found: {file_path}")
    except PermissionError:
        raise DocumentAccessError(f"Permission denied: {file_path}")

# Named constants
MAX_FILE_SIZE = 50 * 1024 * 1024  # 50MB
if file_size > MAX_FILE_SIZE:
    raise FileTooLargeError(f"File size {file_size} exceeds limit {MAX_FILE_SIZE}")

# Configuration-driven paths
file_path = config.UPLOAD_DIR / filename
```

🎯 SPRINT SUCCESS CRITERIA

Technical Milestones

- **Sprint 1**: Complete Docker Compose environment running
- **Sprint 3**: Basic CRUD operations working end-to-end
- **Sprint 5**: First successful RAG query with proper results
- **Sprint 7**: Complete user workflow functioning
- **Sprint 10**: Production-ready deployment with monitoring

Quality Gates (ALL MUST PASS)

- [ ] All tests passing (unit + integration + e2e)
- [ ] Code coverage > 80%
- [ ] Security scan passing
- [ ] Performance benchmarks met
- [ ] Documentation complete
- [ ] Code review approved

🔥 DEVELOPMENT WORKFLOW

Git Flow

```bash
# Feature branches
git checkout -b feature/sprint-X-feature-name

# Conventional commits
git commit -m "feat(documents): add PDF text extraction with error handling"
git commit -m "fix(search): resolve vector similarity calculation bug"
git commit -m "docs(api): update OpenAPI specification for auth endpoints"
```

Code Review Checklist

- [ ] Type hints present and correct
- [ ] Error handling implemented
- [ ] Tests written and passing
- [ ] Logging added for debugging
- [ ] Documentation updated
- [ ] Security considerations addressed
- [ ] Performance impact assessed

🚀 THE MANTRA

"Make it work, make it right, make it fast, make it maintainable"

1. **Work**: Solve the business problem first
2. **Right**: Clean, readable, well-architected code
3. **Fast**: Optimize for performance where it matters
4. **Maintainable**: Future developers will understand and extend it

---

Remember: We're building systems for industrial environments where downtime costs money and operators need answers fast. Every line of code should reflect this reality.

Now go build something awesome! 🛠️