🚀 Complete Regulatory Document Search System

From Zero to Deployment - Step-by-Step Guide

📋 What We're Building

A complete AI-powered regulatory document search system that:

- **Processes** regulatory documents with Claude AI
- **Stores** documents with semantic search capabilities
- **Provides** a web interface for natural language searching
- **Deploys** to the cloud for team access
- **Costs** ~$10-30/month total

🔧 Services You Need to Register For

1. Claude API (Anthropic)

- **Purpose:** AI document analysis and search
- **Cost:** ~$10-30/month depending on usage
- **Register:** https://console.anthropic.com
- **What it does:** Analyzes documents, generates embeddings, answers queries

2. GitHub Account (Free)

- **Purpose:** Code hosting and version control
- **Cost:** Free
- **Register:** https://github.com
- **What it does:** Stores your code and triggers deployments

3. Streamlit Cloud (Free)

- **Purpose:** Web app hosting and deployment
- **Cost:** Free for public apps
- **Register:** https://share.streamlit.io
- **What it does:** Hosts your web interface with automatic deployments

📁 Complete Folder Structure

```
regulatory-search-system/
├── .streamlit/
│   └── config.toml              # Streamlit configuration
├── data/
│   ├── documents/               # Uploaded documents storage
│   ├── embeddings/              # Vector embeddings cache
│   └── processed/               # Processed document metadata
├── src/
│   ├── __init__.py
│   ├── document_processor.py    # Claude AI document processing
│   ├── search_engine.py         # Semantic search functionality
│   ├── embeddings_manager.py    # Vector embeddings management
│   └── utils.py                 # Utility functions
├── pages/
│   ├── 1_Upload_Documents.py    # Document upload interface
│   ├── 2_Search_Documents.py    # Search interface
│   └── 3_System_Status.py       # Monitoring dashboard
├── Home.py                      # Main Streamlit app
├── requirements.txt             # Python dependencies
├── config.yaml                  # System configuration
├── .gitignore                   # Git ignore file
└── README.md                    # Documentation
```

🏗️ Step 1: Initial Setup

1.1 Create Project Directory

bash

```bash
# Create main directory
mkdir regulatory-search-system
cd regulatory-search-system

# Create subdirectories
mkdir -p .streamlit data/{documents,embeddings,processed} src pages
```

1.2 Get Claude API Key

1. Go to **https://console.anthropic.com**
2. **Sign up** for an account
3. **Add billing information** (required for API access)
4. **Create API key** in the dashboard
5. **Save the key** (starts with `sk-ant-`)

📝 Step 2: Core Configuration Files

2.1 Requirements File

```txt
# Core dependencies
streamlit>=1.28.0
anthropic>=0.3.0
pandas>=2.0.0
numpy>=1.24.0

# Document processing
PyPDF2>=3.0.0
python-docx>=0.8.11
openpyxl>=3.1.0

# Vector operations
scikit-learn>=1.3.0
sentence-transformers>=2.2.0

# Utilities
pyyaml>=6.0
python-dateutil>=2.8.0
streamlit-authenticator>=0.2.0
```

2.2 Streamlit Configuration

```toml
```

```
# .streamlit/config.toml
[theme]
primaryColor = "#FF6B6B"
backgroundColor = "#FFFFFF"
secondaryBackgroundColor = "#F0F2F6"
textColor = "#262730"

[server]
enableCORS = false
enableXsrfProtection = false
maxUploadSize = 200

[browser]
gatherUsageStats = false
```

2.3 System Configuration

yaml

```
# config.yaml
app:
  name: "Regulatory Document Search System"
  version: "1.0.0"
  description: "AI-powered regulatory document search with Claude"

authentication:
  enable: true
  default_password: "regulatory2024"

claude:
  model: "claude-3-haiku-20240307"
  max_tokens: 4000
  temperature: 0.1

embeddings:
  model: "all-MiniLM-L6-v2"
  chunk_size: 1000
  chunk_overlap: 200

search:
  max_results: 10
  similarity_threshold: 0.3
```

2.4 Git Ignore

gitignore

```
# .gitignore
__pycache__/
*.pyc
*.pyo
*.pyd
.Python
env/
venv/
.venv/
.env

# Data files
data/documents/*
!data/documents/.gitkeep
data/embeddings/*
!data/embeddings/.gitkeep
data/processed/*
!data/processed/.gitkeep

# Streamlit secrets
.streamlit/secrets.toml

# IDE
.vscode/
.idea/
*.swp
*.swo

# OS
.DS_Store
Thumbs.db
```

🔨 Step 3: Core Python Modules

3.1 Document Processor

python

```python
# src/document_processor.py
import os
import json
import hashlib
from datetime import datetime
from typing import Dict, List, Optional
import pandas as pd
from anthropic import Anthropic
import PyPDF2
from docx import Document
import streamlit as st

class DocumentProcessor:
    def __init__(self, config: dict):
        self.config = config
        self.client = Anthropic(api_key=st.secrets["CLAUDE_API_KEY"])

    def extract_text(self, file_path: str, file_type: str) -> str:
        """Extract text from various document types"""
        try:
            if file_type == 'pdf':
                return self._extract_from_pdf(file_path)
            elif file_type == 'docx':
                return self._extract_from_docx(file_path)
            elif file_type == 'txt':
                return self._extract_from_txt(file_path)
            else:
                raise ValueError(f"Unsupported file type: {file_type}")
        except Exception as e:
            st.error(f"Error extracting text from {file_path}: {str(e)}")
            return ""

    def _extract_from_pdf(self, file_path: str) -> str:
        """Extract text from PDF"""
        text = ""
        with open(file_path, 'rb') as file:
            pdf_reader = PyPDF2.PdfReader(file)
            for page in pdf_reader.pages:
                text += page.extract_text() + "\n"
        return text.strip()

    def _extract_from_docx(self, file_path: str) -> str:
        """Extract text from Word document"""
```

```
        doc = Document(file_path)
        text = ""
        for paragraph in doc.paragraphs:
            text += paragraph.text + "\n"
        return text.strip()

    def _extract_from_txt(self, file_path: str) -> str:
        """Extract text from text file"""
        with open(file_path, 'r', encoding='utf-8') as file:
            return file.read().strip()

    def analyze_with_claude(self, text: str, filename: str) -> Dict:
        """Analyze document with Claude AI"""
        prompt = f"""
        Analyze this regulatory document and extract the following information:

        Document: {filename}
        Text: {text[:4000]}...

        Please provide a JSON response with:
        1. document_type: (regulation, guidance, notice, letter, etc.)
        2. regulator: (FDA, OSHA, EPA, etc.)
        3. subject_areas: [list of main topics]
        4. key_requirements: [list of key requirements or mandates]
        5. effective_date: (if mentioned)
        6. summary: (2-3 sentence summary)
        7. keywords: [relevant search keywords]

        Return only valid JSON.
        """

        try:
            response = self.client.messages.create(
                model=self.config['claude']['model'],
                max_tokens=self.config['claude']['max_tokens'],
                temperature=self.config['claude']['temperature'],
                messages=[{"role": "user", "content": prompt}]
            )

            result = json.loads(response.content[0].text)
            return result

        except Exception as e:
            st.error(f"Error analyzing document with Claude: {str(e)}")
```

```
        return {
            "document_type": "unknown",
            "regulator": "unknown",
            "subject_areas": [],
            "key_requirements": [],
            "effective_date": None,
            "summary": "Analysis failed",
            "keywords": []
        }

    def process_document(self, file_path: str, filename: str) -> Dict:
        """Complete document processing pipeline"""
        file_extension = filename.split('.')[-1].lower()

        # Extract text
        text = self.extract_text(file_path, file_extension)
        if not text:
            return None

        # Generate unique ID
        doc_id = hashlib.md5(f"{filename}{text[:100]}".encode()).hexdigest()

        # Analyze with Claude
        analysis = self.analyze_with_claude(text, filename)

        # Create document metadata
        document_data = {
            "id": doc_id,
            "filename": filename,
            "file_type": file_extension,
            "upload_date": datetime.now().isoformat(),
            "text": text,
            "analysis": analysis,
            "text_length": len(text),
            "processed": True
        }

        # Save to processed folder
        processed_path = f"data/processed/{doc_id}.json"
        os.makedirs("data/processed", exist_ok=True)

        with open(processed_path, 'w', encoding='utf-8') as f:
            json.dump(document_data, f, indent=2, ensure_ascii=False)
```

```
    return document_data
```

3.2 Embeddings Manager

python

```python
# src/embeddings_manager.py
import os
import json
import pickle
import numpy as np
from typing import List, Dict, Tuple
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
import streamlit as st

class EmbeddingsManager:
    def __init__(self, config: dict):
        self.config = config
        self.model_name = config['embeddings']['model']
        self.chunk_size = config['embeddings']['chunk_size']
        self.chunk_overlap = config['embeddings']['chunk_overlap']

        # Load sentence transformer model
        self._load_model()

        # Document embeddings storage
        self.embeddings_path = "data/embeddings"
        os.makedirs(self.embeddings_path, exist_ok=True)

    @st.cache_resource
    def _load_model(_self):
        """Load the sentence transformer model"""
        return SentenceTransformer(_self.model_name)

    def chunk_text(self, text: str) -> List[str]:
        """Split text into overlapping chunks"""
        chunks = []
        start = 0

        while start < len(text):
            end = start + self.chunk_size
            chunk = text[start:end]
            chunks.append(chunk)

            if end >= len(text):
                break

            start += self.chunk_size - self.chunk_overlap
```

```
    return chunks

def generate_embeddings(self, texts: List[str]) -> np.ndarray:
    """Generate embeddings for a list of texts"""
    model = self._load_model()
    embeddings = model.encode(texts, convert_to_tensor=False)
    return embeddings

def process_document_embeddings(self, document_data: Dict) -> Dict:
    """Process embeddings for a document"""
    doc_id = document_data['id']
    text = document_data['text']

    # Check if embeddings already exist
    embeddings_file = f"{self.embeddings_path}/{doc_id}_embeddings.pkl"

    if os.path.exists(embeddings_file):
        with open(embeddings_file, 'rb') as f:
            return pickle.load(f)

    # Generate chunks
    chunks = self.chunk_text(text)

    # Generate embeddings
    embeddings = self.generate_embeddings(chunks)

    # Prepare embeddings data
    embeddings_data = {
        "document_id": doc_id,
        "chunks": chunks,
        "embeddings": embeddings,
        "num_chunks": len(chunks),
        "embedding_model": self.model_name
    }

    # Save embeddings
    with open(embeddings_file, 'wb') as f:
        pickle.dump(embeddings_data, f)

    return embeddings_data

def search_similar_chunks(self, query: str, top_k: int = 10) -> List[Dict]:
    """Search for similar text chunks across all documents"""
```

```
        # Generate query embedding
        query_embedding = self.generate_embeddings([query])[0]

        results = []

        # Load all document embeddings
        for embedding_file in os.listdir(self.embeddings_path):
            if not embedding_file.endswith('_embeddings.pkl'):
                continue

            doc_id = embedding_file.replace('_embeddings.pkl', '')

            try:
                # Load embeddings
                with open(f"{self.embeddings_path}/{embedding_file}", 'rb') as f:
                    embeddings_data = pickle.load(f)

                # Calculate similarities
                similarities = cosine_similarity(
                    [query_embedding],
                    embeddings_data['embeddings']
                )[0]

                # Get top chunks for this document
                for i, similarity in enumerate(similarities):
                    if similarity > self.config['search']['similarity_threshold']:
                        results.append({
                            "document_id": doc_id,
                            "chunk_index": i,
                            "chunk_text": embeddings_data['chunks'][i],
                            "similarity": float(similarity)
                        })

            except Exception as e:
                st.error(f"Error processing embeddings for {doc_id}: {str(e)}")
                continue

        # Sort by similarity and return top results
        results.sort(key=lambda x: x['similarity'], reverse=True)
        return results[:top_k]
```

3.3 Search Engine

python

```python
# src/search_engine.py
import os
import json
from typing import List, Dict, Optional
from datetime import datetime
import pandas as pd
import streamlit as st
from anthropic import Anthropic

from .embeddings_manager import EmbeddingsManager

class SearchEngine:
    def __init__(self, config: dict):
        self.config = config
        self.embeddings_manager = EmbeddingsManager(config)
        self.client = Anthropic(api_key=st.secrets["CLAUDE_API_KEY"])

    def load_document_metadata(self, doc_id: str) -> Optional[Dict]:
        """Load document metadata from processed folder"""
        metadata_path = f"data/processed/{doc_id}.json"

        if not os.path.exists(metadata_path):
            return None

        try:
            with open(metadata_path, 'r', encoding='utf-8') as f:
                return json.load(f)
        except Exception as e:
            st.error(f"Error loading metadata for {doc_id}: {str(e)}")
            return None

    def search_documents(self, query: str, filters: Dict = None) -> List[Dict]:
        """Search documents using semantic similarity"""
        # Get similar chunks
        similar_chunks = self.embeddings_manager.search_similar_chunks(
            query,
            top_k=self.config['search']['max_results'] * 3
        )

        # Group by document and get metadata
        document_results = {}

        for chunk in similar_chunks:
```

```
            doc_id = chunk['document_id']

            if doc_id not in document_results:
                metadata = self.load_document_metadata(doc_id)
                if metadata:
                    document_results[doc_id] = {
                        "document": metadata,
                        "best_similarity": chunk['similarity'],
                        "matching_chunks": [chunk],
                        "total_matches": 1
                    }
            else:
                document_results[doc_id]['matching_chunks'].append(chunk)
                document_results[doc_id]['total_matches'] += 1
                if chunk['similarity'] > document_results[doc_id]['best_similarity']:
                    document_results[doc_id]['best_similarity'] = chunk['similarity']

        # Apply filters if provided
        if filters:
            document_results = self._apply_filters(document_results, filters)

        # Convert to list and sort
        results = list(document_results.values())
        results.sort(key=lambda x: x['best_similarity'], reverse=True)

        return results[:self.config['search']['max_results']]

    def _apply_filters(self, results: Dict, filters: Dict) -> Dict:
        """Apply search filters"""
        filtered_results = {}

        for doc_id, result in results.items():
            document = result['document']
            include = True

            # Regulator filter
            if filters.get('regulator') and filters['regulator'] != 'All':
                if document['analysis']['regulator'].lower() != filters['regulator'].lower():
                    include = False

            # Document type filter
            if filters.get('document_type') and filters['document_type'] != 'All':
                if document['analysis']['document_type'].lower() != filters['document_type'].lower():
                    include = False
```

```
        # Date range filter
        if filters.get('date_from') or filters.get('date_to'):
            doc_date = document.get('upload_date')
            if doc_date:
                doc_date = datetime.fromisoformat(doc_date.replace('Z', '+00:00')).date()

                if filters.get('date_from') and doc_date < filters['date_from']:
                    include = False
                if filters.get('date_to') and doc_date > filters['date_to']:
                    include = False

        if include:
            filtered_results[doc_id] = result

    return filtered_results

def generate_ai_summary(self, query: str, search_results: List[Dict]) -> str:
    """Generate AI summary of search results"""
    if not search_results:
        return "No relevant documents found for your query."

    # Prepare context from top results
    context = ""
    for i, result in enumerate(search_results[:5]):
        doc = result['document']
        context += f"\nDocument {i+1}: {doc['filename']}\n"
        context += f"Regulator: {doc['analysis']['regulator']}\n"
        context += f"Summary: {doc['analysis']['summary']}\n"
        context += f"Key Requirements: {', '.join(doc['analysis']['key_requirements'][:3])}\n"

        # Add best matching chunk
        best_chunk = result['matching_chunks'][0]
        context += f"Relevant excerpt: {best_chunk['chunk_text'][:300]}...\n"
        context += "-" * 50 + "\n"

    prompt = f"""
    Based on the following regulatory documents, provide a comprehensive answer to this query: "{query}"

    Available Documents:
    {context}

    Please provide:
    1. A direct answer to the query
```

```
    2. Key regulatory requirements that apply
    3. Which documents are most relevant
    4. Any important compliance considerations

    Keep the response concise but comprehensive.
    """

    try:
        response = self.client.messages.create(
            model=self.config['claude']['model'],
            max_tokens=self.config['claude']['max_tokens'],
            temperature=self.config['claude']['temperature'],
            messages=[{"role": "user", "content": prompt}]
        )

        return response.content[0].text

    except Exception as e:
        st.error(f"Error generating AI summary: {str(e)}")
        return "Error generating summary. Please review the search results directly."

def get_system_stats(self) -> Dict:
    """Get system statistics"""
    stats = {
        "total_documents": 0,
        "total_chunks": 0,
        "regulators": set(),
        "document_types": set(),
        "last_update": None
    }

    # Count processed documents
    processed_dir = "data/processed"
    if os.path.exists(processed_dir):
        for file in os.listdir(processed_dir):
            if file.endswith('.json'):
                try:
                    with open(f"{processed_dir}/{file}", 'r') as f:
                        doc = json.load(f)
                        stats["total_documents"] += 1
                        stats["regulators"].add(doc['analysis']['regulator'])
                        stats["document_types"].add(doc['analysis']['document_type'])

                        # Track latest update
```

```
                upload_date = datetime.fromisoformat(doc['upload_date'])
                if not stats["last_update"] or upload_date > stats["last_update"]:
                    stats["last_update"] = upload_date
            except:
                continue

    # Count embeddings
    embeddings_dir = "data/embeddings"
    if os.path.exists(embeddings_dir):
        import pickle
        for file in os.listdir(embeddings_dir):
            if file.endswith('_embeddings.pkl'):
                try:
                    with open(f"{embeddings_dir}/{file}", 'rb') as f:
                        embeddings_data = pickle.load(f)
                        stats["total_chunks"] += embeddings_data['num_chunks']
                except:
                    continue

    # Convert sets to lists
    stats["regulators"] = sorted(list(stats["regulators"]))
    stats["document_types"] = sorted(list(stats["document_types"]))

    return stats
```

3.4 Utility Functions

python

```python
# src/utils.py
import os
import yaml
import streamlit as st
from typing import Dict

@st.cache_data
def load_config() -> Dict:
    """Load system configuration"""
    with open('config.yaml', 'r') as f:
        return yaml.safe_load(f)

def setup_page_config():
    """Configure Streamlit page settings"""
    st.set_page_config(
        page_title="Regulatory Document Search",
        page_icon="📋",
        layout="wide",
        initial_sidebar_state="expanded"
    )

def authenticate_user(config: Dict) -> bool:
    """Simple password authentication"""
    if not config['authentication']['enable']:
        return True

    if 'authenticated' not in st.session_state:
        st.session_state.authenticated = False

    if not st.session_state.authenticated:
        st.title("🔐 Authentication Required")
        password = st.text_input("Enter password:", type="password")

        if st.button("Login"):
            if password == config['authentication']['default_password']:
                st.session_state.authenticated = True
                st.rerun()
            else:
                st.error("Incorrect password")

        st.stop()

    return True
```

```python
def display_sidebar_info(config: Dict):
    """Display system information in sidebar"""
    st.sidebar.markdown("---")
    st.sidebar.subheader("📊 System Info")
    st.sidebar.info(f"**Version:** {config['app']['version']}")
    st.sidebar.info(f"**Model:** {config['claude']['model']}")

    # API usage warning
    st.sidebar.warning("⚠️ **Monitor API Usage**\nClaude API calls cost money. Check your Anthropic console regularly

def save_uploaded_file(uploaded_file) -> str:
    """Save uploaded file and return path"""
    os.makedirs("data/documents", exist_ok=True)
    file_path = f"data/documents/{uploaded_file.name}"

    with open(file_path, "wb") as f:
        f.write(uploaded_file.getbuffer())

    return file_path

def create_placeholder_files():
    """Create placeholder files for git"""
    placeholder_dirs = [
        "data/documents",
        "data/embeddings",
        "data/processed"
    ]

    for dir_path in placeholder_dirs:
        os.makedirs(dir_path, exist_ok=True)
        gitkeep_path = f"{dir_path}/.gitkeep"
        if not os.path.exists(gitkeep_path):
            with open(gitkeep_path, 'w') as f:
                f.write("")
```

🎨 Step 4: Streamlit Interface Pages

4.1 Main Home Page

python

```python
# Home.py
import streamlit as st
import yaml
from src.utils import setup_page_config, authenticate_user, display_sidebar_info, load_config
from src.search_engine import SearchEngine

# Page configuration
setup_page_config()

# Load configuration
config = load_config()

# Authentication
authenticate_user(config)

# Main interface
st.title("📋 Regulatory Document Search System")
st.markdown(f"**{config['app']['description']}**")

# Display system stats on home page
if st.button("🔄 Refresh System Stats"):
    search_engine = SearchEngine(config)
    stats = search_engine.get_system_stats()

    col1, col2, col3, col4 = st.columns(4)

    with col1:
        st.metric("📄 Total Documents", stats["total_documents"])

    with col2:
        st.metric("🧩 Text Chunks", stats["total_chunks"])

    with col3:
        st.metric("🏛️ Regulators", len(stats["regulators"]))

    with col4:
        st.metric("📋 Document Types", len(stats["document_types"]))

    if stats["last_update"]:
        st.info(f"**Last Update:** {stats['last_update'].strftime('%Y-%m-%d %H:%M')}")

    if stats["regulators"]:
        st.subheader("Available Regulators")
```

```
        st.write(", ".join(stats["regulators"]))

    if stats["document_types"]:
        st.subheader("Document Types")
        st.write(", ".join(stats["document_types"]))

# Navigation instructions
st.markdown("---")
st.markdown("""
### 🚀 Getting Started

1. **📤 Upload Documents** - Use the Upload Documents page to add regulatory files
2. **🔍 Search Documents** - Use natural language to search your knowledge base
3. **📊 System Status** - Monitor system performance and usage

### 💡 Search Tips

- Use natural language: *"What are the requirements for pharmacy technician training in Ohio?"*
- Be specific: *"FDA guidance on software validation"*
- Ask about compliance: *"What do I need to know about OSHA bloodborne pathogen standards?"*
""")

# Sidebar
display_sidebar_info(config)

# Quick search from home page
st.markdown("---")
st.subheader("🔍 Quick Search")
quick_query = st.text_input("Enter your search query:", placeholder="e.g., 'FDA device registration requirements'")

if quick_query:
    if st.button("Search Now"):
        search_engine = SearchEngine(config)
        results = search_engine.search_documents(quick_query)

        if results:
            st.success(f"Found {len(results)} relevant documents")
            for i, result in enumerate(results[:3]):
                doc = result['document']
                with st.expander(f"📄 {doc['filename']} (Similarity: {result['best_similarity']:.1%})"):
                    st.write(f"**Regulator:** {doc['analysis']['regulator']}")
                    st.write(f"**Type:** {doc['analysis']['document_type']}")
                    st.write(f"**Summary:** {doc['analysis']['summary']}")
```

```
            st.info("Go to Search Documents page for full results and AI analysis")
        else:
            st.warning("No documents found. Try uploading some documents first!")
```

4.2 Upload Documents Page

python

```python
# pages/1_Upload_Documents.py
import streamlit as st
import os
from src.utils import setup_page_config, authenticate_user, display_sidebar_info, load_config, save_uploaded_file
from src.document_processor import DocumentProcessor
from src.embeddings_manager import EmbeddingsManager

# Page configuration
setup_page_config()

# Load configuration
config = load_config()

# Authentication
authenticate_user(config)

st.title("📤 Upload Documents")
st.markdown("Upload regulatory documents to add them to your searchable knowledge base.")

# Sidebar
display_sidebar_info(config)

# File upload interface
st.subheader("📋 Upload New Documents")

uploaded_files = st.file_uploader(
    "Choose regulatory documents",
    type=['pdf', 'docx', 'txt'],
    accept_multiple_files=True,
    help="Supported formats: PDF, Word (.docx), Text (.txt)"
)

if uploaded_files:
    st.success(f"Selected {len(uploaded_files)} files for upload")

    # Display selected files
    for file in uploaded_files:
        st.write(f"📄 **{file.name}** ({file.size:,} bytes)")

    if st.button("🚀 Process Documents", type="primary"):
        # Initialize processors
        doc_processor = DocumentProcessor(config)
        embeddings_manager = EmbeddingsManager(config)
```

```
progress_bar = st.progress(0)
status_text = st.empty()
results_container = st.container()

successful_uploads = 0
failed_uploads = 0

for i, uploaded_file in enumerate(uploaded_files):
    try:
        # Update progress
        progress = (i + 1) / len(uploaded_files)
        progress_bar.progress(progress)
        status_text.text(f"Processing {uploaded_file.name}...")

        # Save file
        file_path = save_uploaded_file(uploaded_file)

        # Process document
        document_data = doc_processor.process_document(file_path, uploaded_file.name)

        if document_data:
            # Generate embeddings
            embeddings_data = embeddings_manager.process_document_embeddings(document_data)

            # Display result
            with results_container.expander(f"✅ {uploaded_file.name} - Processed Successfully"):
                col1, col2 = st.columns(2)

                with col1:
                    st.write(f"**Document Type:** {document_data['analysis']['document_type']}")
                    st.write(f"**Regulator:** {document_data['analysis']['regulator']}")
                    st.write(f"**Text Length:** {document_data['text_length']:,} characters")

                with col2:
                    st.write(f"**Subject Areas:** {', '.join(document_data['analysis']['subject_areas'])}")
                    st.write(f"**Text Chunks:** {embeddings_data['num_chunks']}")
                    st.write(f"**Keywords:** {', '.join(document_data['analysis']['keywords'][:5])}")

                st.write(f"**Summary:** {document_data['analysis']['summary']}")

            successful_uploads += 1
        else:
            failed_uploads += 1
```

```
                with results_container.expander(f"❌ {uploaded_file.name} - Processing Failed"):
                    st.error("Failed to extract text or analyze document")

            except Exception as e:
                failed_uploads += 1
                with results_container.expander(f"❌ {uploaded_file.name} - Error"):
                    st.error(f"Error processing file: {str(e)}")

        # Final status
        progress_bar.progress(1.0)
        status_text.text("Processing complete!")

        # Summary
        if successful_uploads > 0:
            st.success(f"✅ Successfully processed {successful_uploads} documents")

        if failed_uploads > 0:
            st.error(f"❌ Failed to process {failed_uploads} documents")

        st.info("💡 Documents are now available for search. Go to the Search Documents page to find information.")

# Display existing documents
st.markdown("---")
st.subheader("📚 Uploaded Documents")

processed_dir = "data/processed"
if os.path.exists(processed_dir):
    processed_files = [f for f in os.listdir(processed_dir) if f.endswith('.json')]

    if processed_files:
        st.info(f"You have {len(processed_files)} documents in your knowledge base")

        # Load and display document list
        import json
        documents = []

        for file in processed_files:
            try:
                with open(f"{processed_dir}/{file}", 'r', encoding='utf-8') as f:
                    doc_data = json.load(f)
                    documents.append({
                        "Filename": doc_data['filename'],
                        "Regulator": doc_data['analysis']['regulator'],
                        "Type": doc_data['analysis']['document_type'],
```

```
                "Upload Date": doc_data['upload_date'][:10],
                "Size (chars)": f"{doc_data['text_length']:,}"
            })
        except:
            continue

    if documents:
        import pandas as pd
        df = pd.DataFrame(documents)
        st.dataframe(df, use_container_width=True)
    else:
        st.warning("No documents uploaded yet. Upload some documents to get started!")
else:
    st.warning("No documents uploaded yet. Upload some documents to get started!")

# Usage tips
st.markdown("---")
st.markdown("""
### 💡 Upload Tips

- **Supported formats:** PDF, Word (.docx), Text (.txt)
- **File size limit:** 200MB per file
- **Best results:** Use official regulatory documents with clear text
- **Processing time:** ~30-60 seconds per document
- **AI Analysis:** Each document is analyzed by Claude AI for better search

### 📋 What Happens During Processing

1. **Text Extraction** - Extract readable text from your documents
2. **AI Analysis** - Claude identifies key information and requirements
3. **Embedding Generation** - Create semantic vectors for search
4. **Metadata Storage** - Save document information for retrieval
""")
```

4.3 Search Documents Page

python

```python
# pages/2_Search_Documents.py
import streamlit as st
from datetime import datetime, date
from src.utils import setup_page_config, authenticate_user, display_sidebar_info, load_config
from src.search_engine import SearchEngine

# Page configuration
setup_page_config()

# Load configuration
config = load_config()

# Authentication
authenticate_user(config)

st.title("🔍 Search Documents")
st.markdown("Use natural language to search your regulatory knowledge base.")

# Sidebar
display_sidebar_info(config)

# Initialize search engine
search_engine = SearchEngine(config)
stats = search_engine.get_system_stats()

if stats["total_documents"] == 0:
    st.warning("📤 No documents uploaded yet! Go to Upload Documents to add files to your knowledge base.")
    st.stop()

# Search interface
st.subheader("🎯 Search Query")
query = st.text_input(
    "Enter your search query:",
    placeholder="e.g., 'What are the FDA requirements for software validation?'",
    help="Use natural language. Be specific about what you're looking for."
)

# Search filters
with st.expander("🔧 Advanced Filters"):
    col1, col2, col3 = st.columns(3)

    with col1:
        selected_regulator = st.selectbox(
```

```
            "Regulator",
            ["All"] + stats["regulators"]
        )

    with col2:
        selected_doc_type = st.selectbox(
            "Document Type",
            ["All"] + stats["document_types"]
        )

    with col3:
        max_results = st.slider("Max Results", 5, 20, 10)

    # Date filters
    col4, col5 = st.columns(2)
    with col4:
        date_from = st.date_input("From Date", value=None)
    with col5:
        date_to = st.date_input("To Date", value=None)

# Search execution
if query:
    if st.button("🔍 Search", type="primary"):
        # Prepare filters
        filters = {}
        if selected_regulator != "All":
            filters['regulator'] = selected_regulator
        if selected_doc_type != "All":
            filters['document_type'] = selected_doc_type
        if date_from:
            filters['date_from'] = date_from
        if date_to:
            filters['date_to'] = date_to

        # Execute search
        with st.spinner("🤖 Searching documents..."):
            results = search_engine.search_documents(query, filters)

        if results:
            # Generate AI summary
            with st.spinner("🧠 Generating AI analysis..."):
                ai_summary = search_engine.generate_ai_summary(query, results)

            # Display AI summary
```

```
st.subheader("🤖 AI Analysis")
st.markdown("**Based on your regulatory documents, here's what I found:**")
st.markdown(ai_summary)

# Display search results
st.markdown("---")
st.subheader(f"📄 Search Results ({len(results)} documents)")

for i, result in enumerate(results):
    doc = result['document']
    similarity = result['best_similarity']

    with st.expander(f"📋 {doc['filename']} (Relevance: {similarity:.1%})"):
        # Document metadata
        col1, col2, col3 = st.columns(3)

        with col1:
            st.write(f"**Regulator:** {doc['analysis']['regulator']}")
            st.write(f"**Document Type:** {doc['analysis']['document_type']}")

        with col2:
            st.write(f"**Upload Date:** {doc['upload_date'][:10]}")
            st.write(f"**Matching Sections:** {result['total_matches']}")

        with col3:
            st.write(f"**File Type:** {doc['file_type'].upper()}")
            st.write(f"**Size:** {doc['text_length']:,} chars")

        # Document analysis
        st.write(f"**Summary:** {doc['analysis']['summary']}")

        if doc['analysis']['key_requirements']:
            st.write("**Key Requirements:**")
            for req in doc['analysis']['key_requirements'][:3]:
                st.write(f"• {req}")

        if doc['analysis']['subject_areas']:
            st.write(f"**Subject Areas:** {', '.join(doc['analysis']['subject_areas'])}")

        # Show most relevant excerpts
        st.markdown("**📝 Most Relevant Excerpts:**")
        for j, chunk in enumerate(result['matching_chunks'][:2]):
            st.markdown(f"*Excerpt {j+1} (Similarity: {chunk['similarity']:.1%}):*")
            st.text(chunk['chunk_text'][:500] + "...")
```

```
                if j < len(result['matching_chunks']) - 1:
                    st.markdown("---")
        else:
            st.warning("No relevant documents found. Try:")
            st.markdown("""
            - **Broader terms** - Try more general keywords
            - **Different wording** - Rephrase your question
            - **Check filters** - Remove restrictive filters
            - **Upload more documents** - Add more relevant files
            """)

# Sample queries
st.markdown("---")
st.subheader("💡 Sample Queries")

sample_queries = [
    "What are the FDA requirements for medical device software validation?",
    "OSHA bloodborne pathogen exposure control plan requirements",
    "EPA hazardous waste disposal regulations for laboratories",
    "DEA controlled substance inventory requirements",
    "State pharmacy board technician training mandates"
]

st.markdown("**Try these example searches:**")
for query_example in sample_queries:
    if st.button(f"🔍 {query_example}"):
        st.rerun()

# Search tips
st.markdown("---")
st.markdown("""
### 🎯 Search Tips

**For best results:**
- **Be specific** about what you need to know
- **Use regulatory language** when possible
- **Ask complete questions** rather than just keywords
- **Include the regulator** if you know it (FDA, OSHA, EPA, etc.)

**Example good queries:**
- "What documentation is required for FDA 510(k) submissions?"
- "How often must OSHA safety training be conducted?"
- "What are the DEA requirements for controlled substance storage?"
```

The AI will:
- Find relevant documents in your knowledge base
- Provide a comprehensive answer
- Highlight key requirements and compliance points
- Reference specific documents and sections
""")

4.4 System Status Page

python

```python
# pages/3_System_Status.py
import streamlit as st
import os
import json
import pandas as pd
from datetime import datetime
from src.utils import setup_page_config, authenticate_user, display_sidebar_info, load_config
from src.search_engine import SearchEngine

# Page configuration
setup_page_config()

# Load configuration
config = load_config()

# Authentication
authenticate_user(config)

st.title("📊 System Status")
st.markdown("Monitor your regulatory document search system performance and usage.")

# Sidebar
display_sidebar_info(config)

# Get system statistics
search_engine = SearchEngine(config)
stats = search_engine.get_system_stats()

# Overview metrics
st.subheader("📈 System Overview")

col1, col2, col3, col4 = st.columns(4)

with col1:
    st.metric(
        label="📄 Total Documents",
        value=stats["total_documents"],
        help="Number of processed documents in your knowledge base"
    )

with col2:
    st.metric(
        label="🧩 Text Chunks",
```

```
        value=stats["total_chunks"],
        help="Number of searchable text segments"
    )

with col3:
    st.metric(
        label="🏛️ Regulators",
        value=len(stats["regulators"]),
        help="Number of different regulatory agencies"
    )

with col4:
    st.metric(
        label="📋 Document Types",
        value=len(stats["document_types"]),
        help="Variety of document types in your system"
    )

# Last update info
if stats["last_update"]:
    st.info(f"🕒 **Last Document Added:** {stats['last_update'].strftime('%Y-%m-%d at %H:%M')}")
else:
    st.warning("No documents have been uploaded yet.")

# System health
st.markdown("---")
st.subheader("💾 System Health")

health_col1, health_col2 = st.columns(2)

with health_col1:
    # Check API key
    api_status = "✅ Connected" if "CLAUDE_API_KEY" in st.secrets else "❌ Not Configured"
    st.write(f"**Claude API:** {api_status}")

    # Check directories
    dirs_to_check = ["data/documents", "data/processed", "data/embeddings"]
    missing_dirs = [d for d in dirs_to_check if not os.path.exists(d)]

    if missing_dirs:
        st.write(f"**Directory Structure:** ❌ Missing: {', '.join(missing_dirs)}")
    else:
        st.write("**Directory Structure:** ✅ All directories present")
```

```
with health_col2:
    # Check embedding model
    try:
        from src.embeddings_manager import EmbeddingsManager
        embeddings_manager = EmbeddingsManager(config)
        embeddings_manager._load_model()
        st.write("**Embedding Model:** ✅ Loaded successfully")
    except Exception as e:
        st.write(f"**Embedding Model:** ❌ Error: {str(e)[:50]}...")

    # Check configuration
    if os.path.exists("config.yaml"):
        st.write("**Configuration:** ✅ Loaded")
    else:
        st.write("**Configuration:** ❌ Missing config.yaml")

# Detailed statistics
if stats["total_documents"] > 0:
    st.markdown("---")
    st.subheader("📊 Detailed Analytics")

    # Regulator breakdown
    col1, col2 = st.columns(2)

    with col1:
        st.write("**📋 Documents by Regulator**")
        if stats["regulators"]:
            # Count documents per regulator
            regulator_counts = {}
            processed_dir = "data/processed"

            for file in os.listdir(processed_dir):
                if file.endswith('.json'):
                    try:
                        with open(f"{processed_dir}/{file}", 'r') as f:
                            doc = json.load(f)
                            regulator = doc['analysis']['regulator']
                            regulator_counts[regulator] = regulator_counts.get(regulator, 0) + 1
                    except:
                        continue

            regulator_df = pd.DataFrame(
                list(regulator_counts.items()),
                columns=['Regulator', 'Document Count']
```

```
        )
        st.dataframe(regulator_df, use_container_width=True, hide_index=True)

with col2:
    st.write("** 📄 Documents by Type**")
    # Count documents per type
    type_counts = {}
    processed_dir = "data/processed"

    for file in os.listdir(processed_dir):
        if file.endswith('.json'):
            try:
                with open(f"{processed_dir}/{file}", 'r') as f:
                    doc = json.load(f)
                    doc_type = doc['analysis']['document_type']
                    type_counts[doc_type] = type_counts.get(doc_type, 0) + 1
            except:
                continue

    type_df = pd.DataFrame(
        list(type_counts.items()),
        columns=['Document Type', 'Count']
    )
    st.dataframe(type_df, use_container_width=True, hide_index=True)

# Recent uploads
st.markdown("---")
st.subheader("🗓️ Recent Document Activity")

# Load recent documents
recent_docs = []
processed_dir = "data/processed"

for file in os.listdir(processed_dir):
    if file.endswith('.json'):
        try:
            with open(f"{processed_dir}/{file}", 'r') as f:
                doc = json.load(f)
                recent_docs.append({
                    "Filename": doc['filename'],
                    "Regulator": doc['analysis']['regulator'],
                    "Type": doc['analysis']['document_type'],
                    "Upload Date": doc['upload_date'],
                    "Size": f"{doc['text_length']:,} chars"
```

```
                })
            except:
                continue

    if recent_docs:
        # Sort by upload date (most recent first)
        recent_docs.sort(key=lambda x: x['Upload Date'], reverse=True)

        # Show last 10 documents
        recent_df = pd.DataFrame(recent_docs[:10])
        recent_df['Upload Date'] = pd.to_datetime(recent_df['Upload Date']).dt.strftime('%Y-%m-%d %H:%M')

        st.dataframe(recent_df, use_container_width=True, hide_index=True)
    else:
        st.info("No recent document activity.")

# Storage usage
st.markdown("---")
st.subheader("💾 Storage Usage")

storage_col1, storage_col2, storage_col3 = st.columns(3)

def get_directory_size(directory):
    """Calculate directory size in MB"""
    if not os.path.exists(directory):
        return 0

    total_size = 0
    for dirpath, dirnames, filenames in os.walk(directory):
        for filename in filenames:
            file_path = os.path.join(dirpath, filename)
            try:
                total_size += os.path.getsize(file_path)
            except:
                continue

    return total_size / (1024 * 1024)  # Convert to MB

with storage_col1:
    docs_size = get_directory_size("data/documents")
    st.metric("📁 Documents", f"{docs_size:.1f} MB")

with storage_col2:
    processed_size = get_directory_size("data/processed")
```

```
    st.metric("⚙️ Processed", f"{processed_size:.1f} MB")

with storage_col3:
    embeddings_size = get_directory_size("data/embeddings")
    st.metric("🧠 Embeddings", f"{embeddings_size:.1f} MB")

total_size = docs_size + processed_size + embeddings_size
st.info(f"**Total Storage Used:** {total_size:.1f} MB")

# System maintenance
st.markdown("---")
st.subheader("🔧 System Maintenance")

maintenance_col1, maintenance_col2 = st.columns(2)

with maintenance_col1:
    st.write("**🧹 Cleanup Actions**")

    if st.button("🗑️ Clear Cache"):
        # Clear Streamlit cache
        st.cache_data.clear()
        st.cache_resource.clear()
        st.success("Cache cleared successfully!")

    if st.button("📊 Refresh Statistics"):
        st.rerun()

with maintenance_col2:
    st.write("**⚙️ Configuration**")

    st.write(f"**Model:** {config['claude']['model']}")
    st.write(f"**Chunk Size:** {config['embeddings']['chunk_size']}")
    st.write(f"**Max Results:** {config['search']['max_results']}")
    st.write(f"**Similarity Threshold:** {config['search']['similarity_threshold']}")

# Warnings and recommendations
st.markdown("---")
st.subheader("⚠️ Recommendations")

if stats["total_documents"] == 0:
    st.warning("📤 **No documents uploaded** - Upload some regulatory documents to get started")

elif stats["total_documents"] < 5:
    st.info("📈 **Limited content** - Upload more documents for better search results")
```

```
if len(stats["regulators"]) == 1:
    st.info("🏛️ **Single regulator** - Consider adding documents from other regulatory agencies")

if total_size > 500:
    st.warning(f"💾 **High storage usage** ({total_size:.1f} MB) - Consider removing old or duplicate documents")

st.success("🎯 **System is operational** - Ready for document search and upload")
```

🚀 Step 5: Local Testing

5.1 Create Missing Files

python

```
# src/__init__.py (empty file)
# Just create an empty file

# Create placeholder files
from src.utils import create_placeholder_files
create_placeholder_files()
```

5.2 Test Local Installation

bash

```
# Install dependencies
pip install -r requirements.txt

# Test the application
streamlit run Home.py
```

Expected result: Browser opens to `http://localhost:8501` with your app

5.3 Test with Sample Document

1. **Create a test document** (test_document.txt):

```
FDA Guidance Document

This guidance provides recommendations for software validation in medical devices.

Key Requirements:
1. Software documentation must be maintained
2. Risk analysis is required for all software components
3. Testing protocols must be established

Effective Date: January 1, 2024
```

2. **Upload via interface** and verify processing works
3. **Test search** with query: "software validation requirements"

📤 Step 6: GitHub Repository Setup

6.1 Create GitHub Repository

1. **Go to GitHub.com** and sign in
2. **Click "New Repository"**
3. **Repository settings:**
   - **Name:** `regulatory-search-system`
   - **Description:** `AI-powered regulatory document search with Claude`
   - **Visibility:** ✅ **Public** (required for free Streamlit hosting)
   - **Initialize:** ❌ Leave unchecked

6.2 Push Code to GitHub

```bash
```

```
# Initialize git (in your project directory)
git init

# Add remote repository (replace YOUR_USERNAME)
git remote add origin https://github.com/YOUR_USERNAME/regulatory-search-system.git

# Add all files
git add .

# Create initial commit
git commit -m "Initial commit: Regulatory Document Search System"

# Push to GitHub
git branch -M main
git push -u origin main
```

6.3 Verify Repository

Check that all files are visible on GitHub, especially:

- `Home.py` (main application)
- `requirements.txt` (dependencies)
- `config.yaml` (configuration)
- `pages/` folder with all page files
- `src/` folder with all Python modules

☁️ Step 7: Streamlit Cloud Deployment

7.1 Create Streamlit Cloud Account

1. **Go to** https://share.streamlit.io
2. **Sign up** using your GitHub account
3. **Authorize** Streamlit to access your GitHub repositories

7.2 Deploy Your App

1. **Click "New app"**
2. **Repository settings:**
   - **Repository:** `your-username/regulatory-search-system`

- **Branch:** `main`
- **Main file path:** `Home.py`
- **App URL:** Choose a custom URL (e.g., `your-company-regulatory-search`)

3. **Advanced settings:**
   - **Python version:** 3.9+
   - **Requirements file:** `requirements.txt`

7.3 Configure Secrets

1. **In Streamlit Cloud dashboard**, click on your app
2. **Go to Settings → Secrets**
3. **Add the following secrets:**

```toml
[secrets]
CLAUDE_API_KEY = "sk-ant-your-actual-api-key-here"
```

4. **Save secrets**

7.4 Deploy and Test

1. **Click "Deploy"**
2. **Wait 3-5 minutes** for deployment
3. **Your app URL** will be: `https://your-app-name.streamlit.app`
4. **Test the live app** - upload a document and search

👥 Step 8: Team Access Setup

8.1 Share Access Credentials

Send to your team:

🔗 Regulatory Document Search System
URL: https://your-app-name.streamlit.app
Password: regulatory2024

📋 How to Use:
1. Go to the URL above
2. Enter the password
3. Upload documents in "Upload Documents" page
4. Search using natural language in "Search Documents" page

8.2 Train Your Team

Create a team guide:

```
markdown

# Team Guide: Regulatory Document Search

## Getting Started
1. **Access:** Use the shared URL and password
2. **Upload:** Add regulatory documents via Upload Documents page
3. **Search:** Use natural language queries for best results

## Search Examples
- "What are FDA requirements for software validation?"
- "OSHA training requirements for bloodborne pathogens"
- "DEA controlled substance storage regulations"

## Best Practices
- **Upload originals:** Use official regulatory documents
- **Be specific:** Include regulator names in searches
- **Check AI analysis:** Review the AI summary for key points
- **Use filters:** Narrow results by regulator or document type
```

8.3 Change Default Password (Optional)

1. **Edit** `config.yaml` in your GitHub repository:

```
yaml

authentication:
  enable: true
  default_password: "your-new-password-here"
```

2. **Commit and push** changes:

```bash
git add config.yaml
git commit -m "Update authentication password"
git push
```

3. **App auto-redeploys** in 2-3 minutes

💰 Step 9: Cost Management

9.1 Monitor Claude API Usage

Check regularly:

- **Anthropic Console:** https://console.anthropic.com
- **Current usage:** API calls and costs
- **Set alerts:** Get notified at spending thresholds

Expected costs:

- **Light usage (10 docs/month):** $5-10/month
- **Moderate usage (50 docs/month):** $15-25/month
- **Heavy usage (100+ docs/month):** $25-50/month

9.2 Optimize Usage

Reduce costs by:

- **Using Haiku model** (faster, cheaper) instead of Sonnet
- **Chunking documents** appropriately (not too small)
- **Caching results** when possible
- **Training team** on effective search techniques

9.3 Usage Monitoring Dashboard

Add to your **System Status** page to track costs:

```python
```

```
# Add to pages/3_System_Status.py
st.subheader("💰 API Usage Estimate")

# Rough calculation based on processed documents
total_api_calls = stats["total_documents"] * 2  # Process + analyze
estimated_cost = total_api_calls * 0.10  # ~$0.10 per call

st.info(f"""
**Estimated Usage:**
- Total API calls: ~{total_api_calls}
- Estimated cost: ~${estimated_cost:.2f}
- Check actual usage at: https://console.anthropic.com
""")
```

✅ Step 10: System Maintenance

10.1 Regular Maintenance Tasks

Weekly:

- **Check API usage** and costs
- **Review system status** page
- **Clean up failed uploads**

Monthly:

- **Update Claude API** model if needed
- **Review team usage** patterns
- **Archive old documents** if storage grows large

10.2 Troubleshooting Common Issues

❌ "Module not found" errors:

```
bash

# Solution: Check requirements.txt and redeploy
git add requirements.txt
git commit -m "Update dependencies"
git push
```

❌ **"API key not configured":**

- Check Streamlit Cloud secrets configuration
- Verify API key format: `sk-ant-...`

❌ **"Document processing failed":**

- Check document format (PDF, DOCX, TXT only)
- Verify file isn't corrupted
- Check Claude API limits

❌ **"Search returns no results":**

- Upload more documents
- Try broader search terms
- Check similarity threshold in config.yaml

10.3 Adding New Features

To add features:

1. **Edit code** locally
2. **Test changes** with `streamlit run Home.py`
3. **Commit to GitHub:**

```bash
git add .
git commit -m "Add new feature: [description]"
git push
```

4. **Streamlit auto-redeploys** in 2-3 minutes

🎉 Congratulations! You're Live!

✅ What You Now Have:

- 🌐 **Public web app** accessible from anywhere
- 🔒 **Password protected** team access
- 📱 **Mobile responsive** interface

- 🤖 **AI-powered search** with Claude
- ☁️ **Zero server maintenance**
- 💰 **Pay-per-use** cost model (~$10-30/month)

📊 Success Metrics:

Track these to measure success:

- **Documents uploaded** per month
- **Search queries** per week
- **Team adoption** rate
- **Search accuracy** (user feedback)
- **Cost per document** processed

🚀 Next Steps:

1. **Upload your regulatory documents**
2. **Train your team** on effective searching
3. **Monitor costs** and usage patterns
4. **Gather feedback** for improvements
5. **Expand document** collection over time

🆘 Support Resources:

- **Streamlit Documentation:** https://docs.streamlit.io
- **Claude API Docs:** https://docs.anthropic.com
- **GitHub Help:** https://docs.github.com
- **Your System Status Page:** Monitor health and performance

Your regulatory search system is now fully operational! 🚀

📋 Quick Reference Checklist

Pre-Deployment:

- [ ] Claude API key obtained and tested
- [ ] GitHub account created
- [ ] Local system tested with sample document
- [ ] All code files created and verified

Deployment:

- [ ] GitHub repository created and populated
- [ ] Streamlit Cloud account connected
- [ ] App deployed successfully
- [ ] API key configured in secrets
- [ ] Live system tested

Team Setup:

- [ ] Access credentials shared with team
- [ ] Team training guide created
- [ ] Password changed (if desired)
- [ ] Usage monitoring configured
- [ ] Maintenance schedule established

You're all set! 🎊