AI Idea to Reality POC - Complete Project Documentation

Table of Contents

Executive Summary

The **AI Idea to Reality POC** is an enterprise-grade Streamlit application designed for DexKo Group to streamline the innovation lifecycle. It enables employees to submit ideas, leverages AI for automated research and evaluation, and provides managers with tools to review and approve promising initiatives.

Key Highlights

- **AI-Powered Research**: Automated company and market research using Tavily API and Azure OpenAI GPT-4o
- **Intelligent Evaluation**: AI scoring system for objective idea assessment
- **Role-Based Access**: Separate interfaces for employees, managers, and directors
- **Comprehensive Workflow**: From idea submission to resource estimation and development planning
- **Document Generation**: Automated POC document creation in DOCX format

---

Project Overview

Purpose

Transform the innovation process at DexKo Group by providing a centralized platform where: - Employees can submit and develop ideas with AI assistance - AI agents perform comprehensive research on companies and market trends - Resource estimation helps in planning and budgeting - Managers can review, evaluate, and approve ideas efficiently - All stakeholders have visibility into the innovation pipeline

Core Objectives

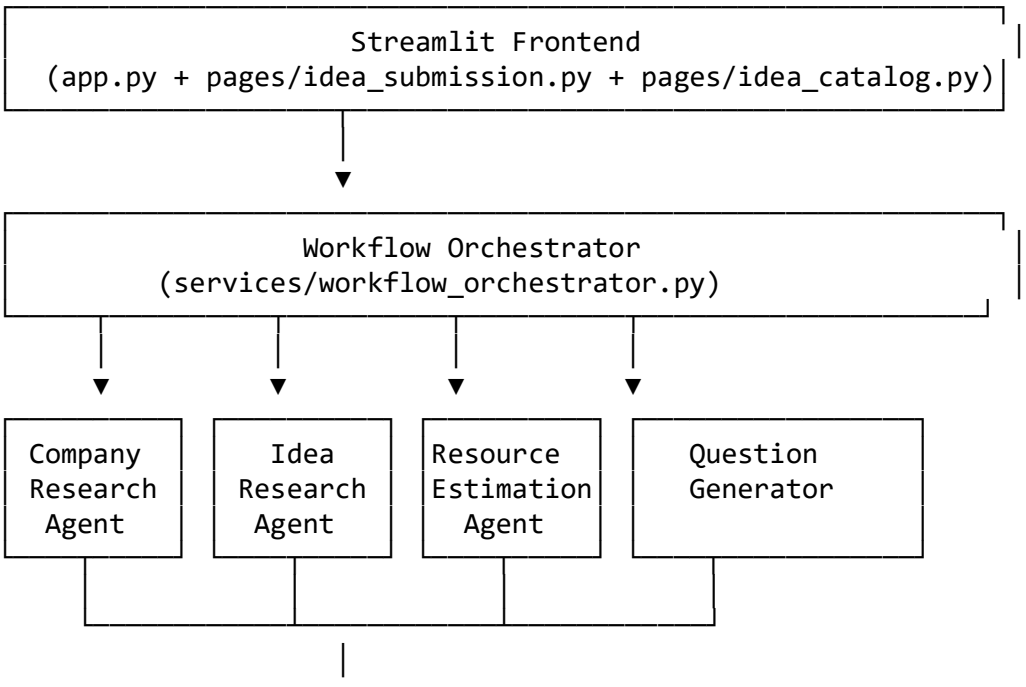
1. **Democratize Innovation**: Enable all employees to contribute ideas
2. **Accelerate Research**: Automate time-consuming research tasks
3. **Improve Decision Making**: Provide data-driven insights for evaluation
4. **Streamline Workflow**: Create a seamless process from idea to implementation
5. **Track Progress**: Maintain a centralized catalog of all ideas and their status

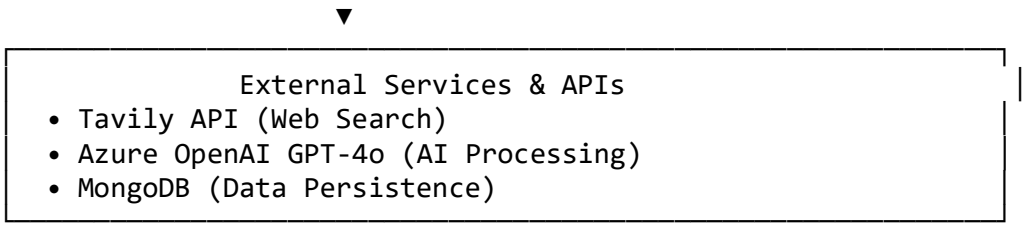
Target Users

- **Employees**: Submit and develop innovative ideas
- **Managers**: Review and evaluate submitted ideas
- **Directors**: Strategic oversight and final approvals
- **Executives**: Portfolio view of innovation initiatives

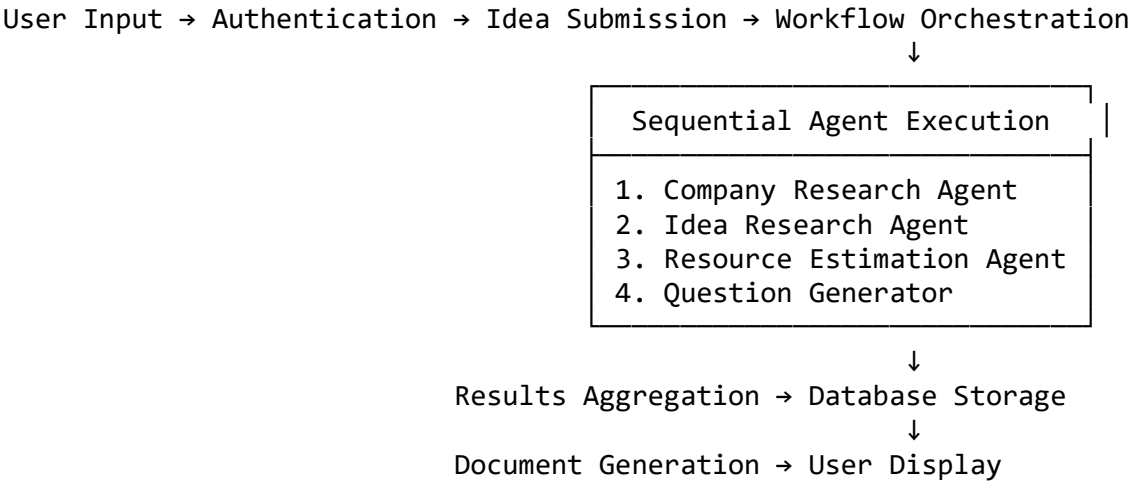
System Architecture

High-Level Architecture





Application Flow



Data Flow

1. **Input Stage**: User submits company name, idea title, and description
2. **Research Stage**: Parallel/sequential research agents gather data
3. **Processing Stage**: AI analyzes and structures information
4. **Storage Stage**: Results saved to MongoDB
5. **Output Stage**: Results displayed in UI and available for download

---

Core Components

1. Frontend Layer

app.py - Main Application Entry Point

- **Purpose**: Streamlit application orchestration
- **Responsibilities**:
  - User authentication and session management
  - Navigation between pages
  - Global UI configuration
  - CSS styling integration
- **Key Features**:
  - Horizontal navigation menu using `streamlit-option-menu`
  - Role-based page visibility

  - Augent branding with logo integration
  - Session state management

pages/idea_submission.py *- Idea Submission Interface*

- **Purpose**: Primary interface for submitting and developing ideas
- **Responsibilities**:
  - Idea input form (company name, title, description)
  - Workflow orchestration trigger
  - Real-time progress tracking
  - Research results display
  - Document generation and download
- **Key Features**:
  - Multi-step workflow with progress indicators
  - Expandable sections for research results
  - Resource estimation display in tabs
  - Development questions generation
  - DOCX document export

pages/idea_catalog.py *- Idea Management*

- **Purpose**: Browse and manage submitted ideas
- **Responsibilities**:
  - Display all ideas in a searchable catalog
  - Filter by status, department, score
  - Sort by date, score, or title
  - View detailed idea information
- **Key Features**:
  - Card-based layout for ideas
  - Status badges with color coding
  - AI score visualization
  - Quick actions (view, edit, delete)

pages/reviewer_dashboard.py *- Review Interface*

- **Purpose**: Manager/Director interface for idea evaluation
- **Responsibilities**:
  - Display pending ideas for review
  - Provide evaluation tools
  - Submit reviews with scores and feedback
  - Update idea status
- **Key Features**:
  - Filtered view of ideas needing review
  - Evaluation form with score slider
  - Action buttons (Approve/Reject/Request Changes)

- Review history tracking

2. Service Layer

services/workflow_orchestrator.py - Workflow Coordination

- **Purpose**: Orchestrate the complete idea research workflow
- **Responsibilities**:
  - Sequential execution of research agents
  - Error handling and recovery
  - State management and persistence
  - Data aggregation from multiple agents
- **Key Methods**:
  - `start_workflow()`: Main entry point for workflow execution
  - `perform_company_research()`: Trigger company research
  - `perform_idea_research()`: Trigger market research
  - `perform_resource_estimation()`: Estimate resources needed
  - `generate_development_questions()`: Create personalized questions
  - `load_workflow_state()`: Resume from saved state

services/company_research_agent.py - Company Intelligence

- **Purpose**: Research target company information
- **Data Extracted**:
  - **What the company does**: Business description, products, services
  - **Financials**: Annual revenue, growth rate, market cap
  - **Current initiatives**: Strategic goals, digital transformation efforts
  - **Sources**: URLs and references for all information
- **Technology**: Tavily API for web search, Azure GPT-4o for content analysis
- **Output Format**: Structured JSON with categorized information

services/idea_research_agent.py - Market Intelligence

- **Purpose**: Research market implementation of similar ideas
- **Data Extracted**:
  - **Who is implementing**: Companies/organizations already using similar solutions
  - **Pros and cons**: Benefits and challenges of implementations
  - **Market insights**: Trends, growth rates, adoption patterns
  - **Sources**: Verifiable URLs for all claims
- **Technology**: Tavily API + AI-powered content categorization
- **Output Format**: Structured JSON with implementation examples

services/resource_estimation_agent.py - Resource Planning

- **Purpose**: Estimate resources required for idea implementation
- **Estimates Provided**:

    - **Team composition**: Roles and headcount needed
    - **Timeline**: Development phases and duration
    - **Technology stack**: Required tools and platforms
    - **Budget**: Cost estimates for different categories
    - **Risks**: Potential challenges and mitigation strategies
- **Technology**: DeepSeek AI for intelligent estimation
- **Output Format**: JSON with categorized resource requirements

services/research_agent.py *- Core Research Engine*

- **Purpose**: Foundational research capabilities using Tavily API
- **Capabilities**:
    - Web search with full content extraction
    - AI-powered content categorization
    - Opportunity and challenge extraction
    - Source management and attribution
    - Parallel processing for performance
- **Key Features**:
    - Smart content cleaning and extraction
    - AI-based result classification
    - Comprehensive summary generation
    - No artificial restrictions on data extraction

services/question_generator.py *- Development Questions*

- **Purpose**: Generate personalized development questions
- **Question Categories**:
    - Technical feasibility
    - Business value
    - Implementation strategy
    - Risk management
    - Resource requirements
- **Technology**: Azure OpenAI GPT-4o
- **Output**: 5-10 targeted questions to guide idea development

services/research_document_generator.py *- Document Creation*

- **Purpose**: Create comprehensive research documents
- **Document Sections**:
    - Executive Summary
    - Company Overview
    - Idea Analysis
    - Market Research
    - Resource Estimation
    - Development Questions

    - Appendix (Sources)
- **Technology**: python-docx library
- **Output Format**: Professional DOCX with formatting

services/ai_score_service.py - AI Evaluation

- **Purpose**: Automated idea scoring and feedback
- **Scoring Criteria**:
    - Innovation (25%): Novelty and creativity
    - Feasibility (25%): Practicality and implementability
    - Business Impact (25%): Potential value to DexKo
    - Clarity (25%): How well-defined the idea is
- **Output**: Score (0-100), strengths, improvements, detailed feedback

services/idea_service.py - Idea CRUD Operations

- **Purpose**: Database operations for ideas
- **Operations**:
    - Create new ideas
    - Retrieve ideas (single, multiple, filtered)
    - Update idea status and content
    - Delete ideas
    - Search and filter
- **Technology**: MongoDB with PyMongo

services/database.py - MongoDB Connection

- **Purpose**: Database connection management
- **Responsibilities**:
    - Connection pooling
    - Error handling
    - Query execution
    - Index management

3. Utility Layer

utils/auth.py - Authentication & Authorization

- **Purpose**: User authentication and session management
- **Features**:
    - Demo user authentication
    - Role-based access control
    - Session state initialization
    - Login/logout functionality
- **Demo Users**:
    - `user@example.com` - Employee role
    - `manager@example.com` - Manager role

  - `director@example.com` - Director role

utils/helpers.py - Utility Functions

- **Purpose**: Common helper functions
- **Functions**:
  - Date formatting
  - String manipulation
  - Data validation
  - UI helpers

utils/ai_questions.py - AI Question Management

- **Purpose**: Manage AI-generated development questions
- **Features**:
  - Question storage and retrieval
  - Answer tracking
  - Progress monitoring

utils/cache_manager.py - Caching System

- **Purpose**: Cache management for performance
- **Features**:
  - API response caching
  - Research result caching
  - Cache invalidation
  - TTL management

utils/error_handler.py - Error Management

- **Purpose**: Centralized error handling
- **Features**:
  - Error logging
  - User-friendly error messages
  - Retry logic
  - Fallback mechanisms

utils/api_optimizer.py - API Optimization

- **Purpose**: Optimize external API calls
- **Features**:
  - Rate limiting
  - Request batching
  - Response caching
  - Cost tracking

utils/json_parser.py - JSON Processing

- **Purpose**: Robust JSON parsing

- **Features**:
  - Safe JSON parsing
  - Schema validation
  - Error recovery
  - Type conversion

4. Data Models

models.py *- Data Schemas*

- **IdeaDocument**: Complete idea structure
- **SectionDocument**: Idea sections (Executive Summary, Business Value, etc.)
- **SubsectionDocument**: Subsection details
- **ConversationEntryDocument**: Q&A history
- **MetadataDocument**: Timestamps, user info, statistics
- **DexKoUserContext**: User profile and department
- **IdeaStatus**: Enum for idea lifecycle states
- **DexKoDepartment**: Enum for organizational departments

Features & Capabilities

1. Idea Submission & Development

- **Structured Input**: Company name, idea title, detailed description
- **AI-Powered Research**: Automated company and market research
- **Resource Estimation**: AI-generated resource requirements
- **Development Questions**: Personalized questions to refine ideas
- **Document Generation**: Professional POC documents in DOCX format

2. AI Research Capabilities

- **Company Research**:
  - Business description and operations
  - Financial metrics (revenue, growth, market cap)
  - Current strategic initiatives
  - Source attribution with URLs
- **Market Research**:
  - Companies implementing similar ideas
  - Pros and cons of existing implementations
  - Market trends and insights
  - Verified sources for all claims
- **Resource Estimation**:
  - Team composition and roles
  - Timeline and milestones
  - Technology requirements

  - Budget estimates
  - Risk assessment

3. Idea Evaluation

- **AI Scoring**: Automated evaluation on 4 criteria
- **Human Review**: Manager/Director evaluation interface
- **Feedback System**: Constructive feedback and improvement suggestions
- **Status Tracking**: Lifecycle management (submitted → approved → implemented)

4. Idea Catalog

- **Search & Filter**: By status, department, score, keywords
- **Sorting**: By date, score, title
- **Detailed View**: Complete idea information and research
- **Bulk Operations**: Export, archive, delete multiple ideas

5. Reviewer Dashboard

- **Pending Ideas**: Queue of ideas awaiting review
- **Evaluation Tools**: Score slider, action buttons, feedback form
- **Statistics**: Review metrics and performance indicators
- **History**: Track all reviews and decisions

6. Document Management

- **POC Documents**: Comprehensive research documents
- **Professional Formatting**: Structured sections with proper styling
- **Source Attribution**: All claims backed by URLs
- **Export Options**: DOCX format for easy sharing

Technology Stack

Backend Technologies

Technology	Version	Purpose
Python	3.8+	Core programming language
Streamlit	1.40.1	Web application framework
PyMongo	4.6.0	MongoDB driver
Pydantic	2.11.7	Data validation and schemas
LangChain	0.3.26	AI orchestration framework
LangGraph	0.5.2	Workflow graphs

AI & ML Services

Service	Purpose
Azure OpenAI GPT-4o	Content analysis, summarization, question generation

Service	Purpose
DeepSeek AI	Resource estimation
Tavily API	Web search and research
LangSmith	AI monitoring and debugging

Data & Storage

Technology	Purpose
MongoDB	Primary database for ideas and metadata
JSON	Data serialization and state management
Temporary Files	Workflow state persistence

External Libraries

Library	Version	Purpose
`streamlit-option-menu`	0.3.15	Enhanced navigation menu
`python-docx`	0.8.11	DOCX document generation
`python-dotenv`	1.1.1	Environment variable management
`requests`	2.32.4	HTTP requests
`pandas`	2.0.3	Data manipulation
`altair`	5.0.1	Data visualization

Development Tools

- **Git**: Version control
- **Docker**: Containerization (optional)
- **Logging**: Python logging module
- **Environment Variables**: Secure configuration management

Installation & Setup

Prerequisites

- **Python**: 3.8 or higher
- **MongoDB**: 4.0 or higher (local or Atlas)
- **pip**: Package manager
- **Git**: For version control

Step 1: Clone the Repository

```
cd g:\DEXKO\I2POC_Copy\I2POC_Streamlit
```

Step 2: Create Virtual Environment

Windows (Command Prompt):

```
python -m venv venv
venv\Scripts\activate
```

Windows (PowerShell):

```
python -m venv venv
.\venv\Scripts\Activate.ps1
```

Mac/Linux:

```
python3 -m venv venv
source venv/bin/activate
```

Step 3: Install Dependencies

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

Step 4: Configure Environment Variables

Create a `.env` file in the project root:

```
# MongoDB Configuration
MONGODB_URL=mongodb://localhost:27017
MONGODB_DATABASE=i2poc
MONGODB_COLLECTION=ideas

# Azure OpenAI Configuration (Required for AI features)
GPT_4O_API_KEY=your_azure_openai_api_key
AZURE_OPENAI_ENDPOINT=https://your-instance.openai.azure.com/

# Tavily API (Required for research)
TAVILY_API_KEY=your_tavily_api_key

# DeepSeek API (Required for resource estimation)
DEEPSEEK_API_KEY=your_deepseek_api_key

# Optional: LangSmith (for monitoring)
LANGSMITH_API_KEY=your_langsmith_api_key

# Optional: Groq API
GROQ_API_KEY=your_groq_api_key
```

Step 5: Setup MongoDB

Option A: Local MongoDB

```
# Start MongoDB service
mongod

# Verify connection
mongo --eval "db.version()"
```

Option B: MongoDB Atlas (Cloud) 1. Create account at https://www.mongodb.com/cloud/atlas 2. Create a cluster 3. Get connection string 4. Update `MONGODB_URL` in `.env`

Step 6: Run the Application

```
streamlit run app.py
```

The application will open at `http://localhost:8501`

Step 7: Verify Installation

1. Login with demo credentials: `manager@example.com` / `password123`
2. Navigate to "Submit Idea" tab
3. Test idea submission with sample data
4. Verify research results display

User Guide

For Employees

Submitting an Idea

1. **Login**: Use your credentials to access the system
2. **Navigate**: Go to "Submit Idea" tab
3. **Enter Details**:
   - **Company Name**: Target company for the idea (e.g., "PwC")
   - **Idea Title**: Short, descriptive title (e.g., "AI-Powered HR Analytics")
   - **Idea Description**: Detailed explanation of your idea
4. **Start Research**: Click "Start Research & Analysis"
5. **Monitor Progress**: Watch real-time progress indicators
6. **Review Results**: Examine research findings in expandable sections
7. **Download Document**: Click "Download POC Document" for DOCX file
8. **Submit to Catalog**: Click "Submit to Catalog" to save

Viewing Ideas

1. **Navigate**: Go to "Idea Catalog" tab
2. **Browse**: View all submitted ideas
3. **Filter**: Use filters for status, department, or score
4. **Search**: Enter keywords to find specific ideas
5. **View Details**: Click "View Details" for complete information

For Managers/Directors

Reviewing Ideas

1. **Login**: Use manager/director credentials
2. **Navigate**: Go to "Reviewer Dashboard"

3. **View Pending**: See ideas awaiting review
4. **Select Idea**: Click on an idea to review
5. **Evaluate**:
   - Read through all sections
   - Review AI score and feedback
   - Examine research data
6. **Provide Feedback**:
   - Select action (Approve/Reject/Request Changes)
   - Set evaluation score (0-100)
   - Write constructive feedback
7. **Submit Review**: Click "Submit Review"

Managing Ideas

1. **Track Status**: Monitor idea progression
2. **View Statistics**: Review metrics and KPIs
3. **Export Data**: Download reports for analysis
4. **Update Status**: Change idea status as needed

API & Services

Workflow Orchestrator API

```python
from services.workflow_orchestrator import workflow_orchestrator

# Start complete workflow
results = workflow_orchestrator.start_workflow(
    company_name="PwC",
    idea_title="AI HR Solutions",
    idea_description="AI-powered HR analytics platform",
    on_company_complete=callback_function,  # Optional
    on_idea_complete=callback_function,     # Optional
    on_resource_complete=callback_function  # Optional
)

# Load saved state
state = workflow_orchestrator.load_workflow_state(
    company_name="PwC",
    idea_title="AI HR Solutions"
)
```

Research Agent API

```python
from services.research_agent import research_agent

# Perform market research
results = research_agent.research_idea(
    idea="AI-powered HR analytics",
```

```
    title="AI HR Solutions"
)
```

Idea Service API

```python
from services.idea_service import idea_service

# Create new idea
idea_id = idea_service.create_idea(idea_document)

# Get idea by ID
idea = idea_service.get_idea(idea_id)

# Update idea
idea_service.update_idea(idea_id, updates)

# Get all ideas with filters
ideas = idea_service.get_all_ideas(
    status="submitted",
    department="Engineering",
    limit=50
)

# Search ideas
results = idea_service.search_ideas(query="AI")
```

AI Score Service API

```python
from services.ai_score_service import ai_score_service

# Score an idea
score_result = ai_score_service.score_idea(idea_document)
# Returns: {
#   "score": 85,
#   "feedback": "...",
#   "strengths": [...],
#   "improvements": [...]
# }
```

Database Schema

Ideas Collection

```json
{
  "_id": "ObjectId",
  "session_id": "uuid-string",
  "title": "AI HR Solutions",
  "original_idea": "Original user input",
  "rephrased_idea": "AI-refined version",
  "sections": [
    {
```

```
    "section_heading": "Executive Summary",
    "section_purpose": "Brief overview",
    "subsections": [
      {
        "subsection_heading": "Problem Statement",
        "subsection_definition": "What problem does this solve?"
      }
    ]
  }
],
"drafts": {
  "section_name": "draft_content"
},
"conversation_history": [
  {
    "section": "Executive Summary",
    "subsection": "Problem Statement",
    "question": "What problem are you solving?",
    "answer": "User response"
  }
],
"metadata": {
  "created_at": "2025-12-08T10:00:00Z",
  "updated_at": "2025-12-08T10:30:00Z",
  "submitted_by": "john.doe@dexko.com",
  "department": "Engineering",
  "total_questions_asked": 15,
  "completion_time_minutes": 45.5
},
"dexko_context": {
  "user_id": "EMP12345",
  "department": "Engineering",
  "role": "Senior Engineer",
  "location": "Germany",
  "language": "en"
},
"status": "submitted",
"ai_score": 85,
"ai_feedback": "Strong idea with clear value proposition...",
"ai_strengths": [
  "Clear problem definition",
  "Innovative approach"
],
"ai_improvements": [
  "Add more market analysis",
  "Clarify implementation timeline"
],
"ai_categorization": {
  "category": "Digital Transformation",
  "subcategory": "HR Technology"
```

```
  },
  "research_data": {
    "company_research": {...},
    "idea_research": {...},
    "resource_estimation": {...},
    "development_questions": [...]
  },
  "evaluation_score": 90,
  "reviewer_feedback": "Excellent idea, approved for pilot"
}
```

Indexes

```
// Recommended indexes for performance
db.ideas.createIndex({ "status": 1 })
db.ideas.createIndex({ "metadata.created_at": -1 })
db.ideas.createIndex({ "metadata.department": 1 })
db.ideas.createIndex({ "ai_score": -1 })
db.ideas.createIndex({ "title": "text", "original_idea": "text" })
```

Configuration

config.py Settings

```
# Application Settings
APP_TITLE = "AI Idea to Reality POC"
APP_DESCRIPTION = "Streamlit-based platform for idea submission"
APP_PAGE_ICON = ""

# Database Settings
MONGODB_URL = "mongodb://localhost:27017"
MONGODB_DATABASE = "i2poc"
MONGODB_COLLECTION = "ideas"

# AI Model Configuration
USE_AZURE_OPENAI = True  # Requires GPT_4O_API_KEY
USE_DEEPSEEK = True      # Requires DEEPSEEK_API_KEY
```

constants.py Configuration

```
# Idea Sections
SECTIONS = [...]  # Predefined idea structure

# Scoring Criteria
SCORING_CRITERIA = {
    "innovation": {"weight": 0.25},
    "feasibility": {"weight": 0.25},
    "business_impact": {"weight": 0.25},
    "clarity": {"weight": 0.25}
}
```

```python
# Departments
DEXKO_DEPARTMENTS = [
    "Engineering", "Manufacturing", "Sales",
    "Marketing", "Finance", "HR", "IT",
    "Operations", "Supply Chain", "Other"
]

# User Roles
USER_ROLES = ["Employee", "Manager", "Director", "Executive"]
```

Streamlit Configuration (.streamlit/config.toml)

```toml
[theme]
primaryColor = "#792a85"
backgroundColor = "#FFFFFF"
secondaryBackgroundColor = "#f9f5fa"
textColor = "#262730"
font = "sans serif"

[server]
port = 8501
enableCORS = false
enableXsrfProtection = true
```

Development Guide

Project Structure

```
I2POC_Streamlit/
├── app.py                          # Main application entry
├── models.py                       # Data models (Pydantic)
├── config.py                       # Configuration
├── constants.py                    # Constants and enums
├── requirements.txt                # Python dependencies
├── .env                           # Environment variables (not in git)
├── .env.example                   # Example environment file
├── .gitignore                     # Git ignore rules
├── Dockerfile                     # Docker configuration
├── docker-compose.yml             # Docker Compose setup
├── run.bat                        # Windows run script
├── run.sh                         # Unix run script
├── styles.css                     # Custom CSS styles
├── README.md                      # Quick start guide
├── ARCHITECTURE_OVERVIEW.md       # Architecture documentation
├── PROJECT_DOCUMENTATION.md       # This file
│
├── .streamlit/
│   └── config.toml                # Streamlit configuration
│
├── assets/
```

```
│   ├── 6866b1d280bd0335ad8ab8ca_Augent-Philosophy.svg
│   └── [other assets]
│
├── pages/
│   ├── __init__.py
│   ├── idea_submission.py         # Idea submission page
│   ├── idea_catalog.py            # Idea catalog page
│   ├── idea_development.py        # Idea development page
│   └── reviewer_dashboard.py      # Reviewer interface
│
├── services/
│   ├── __init__.py
│   ├── workflow_orchestrator.py   # Workflow coordination
│   ├── company_research_agent.py  # Company research
│   ├── idea_research_agent.py     # Market research
│   ├── resource_estimation_agent.py # Resource planning
│   ├── research_agent.py          # Core research engine
│   ├── question_generator.py      # Question generation
│   ├── research_document_generator.py # Document creation
│   ├── ai_score_service.py        # AI scoring
│   ├── idea_service.py            # Idea CRUD
│   ├── database.py                # MongoDB connection
│   └── text_cleaner.py            # Text processing
│
└── utils/
    ├── __init__.py
    ├── auth.py                    # Authentication
    ├── helpers.py                 # Helper functions
    ├── ai_questions.py            # Question management
    ├── cache_manager.py           # Caching system
    ├── error_handler.py           # Error handling
    ├── api_optimizer.py           # API optimization
    └── json_parser.py             # JSON processing
```

Adding New Features

1. Adding a New Page

```python
# pages/new_page.py
import streamlit as st

def show():
    """Display the new page"""
    st.title("New Page")
    # Your page logic here

# app.py - Add to navigation
from pages.new_page import show as show_new_page

# Add to menu options
selected = option_menu(
```

```
    options=["Submit Idea", "Idea Catalog", "New Page", "Reviewer Dashboard"]
,
    # ...
)

if selected == "New Page":
    show_new_page()
```

2. Adding a New Service

```
# services/new_service.py
import logging

logger = logging.getLogger(__name__)

class NewService:
    def __init__(self):
        self.logger = logging.getLogger(__name__)

    def perform_action(self, data):
        """Perform service action"""
        try:
            # Service logic
            return result
        except Exception as e:
            self.logger.error(f"Error: {e}")
            raise

# Create singleton instance
new_service = NewService()
```

3. Adding a New Data Model

```
# models.py
from pydantic import BaseModel, Field
from typing import Optional

class NewDocument(BaseModel):
    field1: str
    field2: Optional[int] = None
    field3: List[str] = Field(default_factory=list)

    class Config:
        populate_by_name = True
```

Code Style Guidelines

1. **Python Style**: Follow PEP 8
2. **Docstrings**: Use Google-style docstrings
3. **Type Hints**: Use type annotations
4. **Error Handling**: Always use try-except blocks

5. **Logging**: Use logging module, not print statements
6. **Comments**: Explain why, not what

Testing

```
# Example test structure (to be implemented)
import pytest
from services.idea_service import idea_service

def test_create_idea():
    idea = {
        "title": "Test Idea",
        "original_idea": "Test description"
    }
    idea_id = idea_service.create_idea(idea)
    assert idea_id is not None
```

Logging Best Practices

```
import logging

logger = logging.getLogger(__name__)

# Info level for normal operations
logger.info("Starting workflow for company: %s", company_name)

# Warning for recoverable issues
logger.warning("API rate limit approaching")

# Error for failures
logger.error("Failed to connect to database: %s", str(e))

# Debug for detailed troubleshooting
logger.debug("Research results: %s", results)
```

Deployment

Local Development

```
# Activate virtual environment
venv\Scripts\activate  # Windows
source venv/bin/activate  # Mac/Linux

# Run application
streamlit run app.py

# Run on custom port
streamlit run app.py --server.port 8502
```

Docker Deployment

Dockerfile:

```
FROM python:3.9-slim

WORKDIR /app

# Install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application
COPY . .

# Expose port
EXPOSE 8501

# Run application
CMD ["streamlit", "run", "app.py", "--server.port=8501", "--server.address=0.
0.0.0"]
```

Build and Run:

```
# Build image
docker build -t i2poc-streamlit .

# Run container
docker run -p 8501:8501 --env-file .env i2poc-streamlit
```

Docker Compose:

```
version: '3.8'

services:
  app:
    build: .
    ports:
      - "8501:8501"
    environment:
      - MONGODB_URL=mongodb://mongo:27017
    depends_on:
      - mongo
    volumes:
      - ./cache:/app/cache

  mongo:
    image: mongo:4.4
    ports:
      - "27017:27017"
```

```
    volumes:
      - mongo-data:/data/db

volumes:
  mongo-data:
```

Production Deployment

Option 1: Streamlit Cloud

1. Push code to GitHub
2. Connect repository to Streamlit Cloud
3. Configure secrets in Streamlit Cloud dashboard
4. Deploy from main branch

Option 2: AWS EC2

```
# Install dependencies
sudo apt-get update
sudo apt-get install python3 python3-pip

# Clone repository
git clone <repository-url>
cd I2POC_Streamlit

# Setup virtual environment
python3 -m venv venv
source venv/bin/activate

# Install requirements
pip install -r requirements.txt

# Run with nohup
nohup streamlit run app.py > app.log 2>&1 &
```

Option 3: Azure App Service

1. Create App Service in Azure Portal
2. Configure Python runtime
3. Set environment variables in Configuration
4. Deploy via GitHub Actions or Azure CLI

Environment Variables for Production

```
# Production MongoDB (Atlas)
MONGODB_URL=mongodb+srv://username:password@cluster.mongodb.net/

# Production API Keys
GPT_4O_API_KEY=prod_key_here
TAVILY_API_KEY=prod_key_here
DEEPSEEK_API_KEY=prod_key_here
```

```
# Security
STREAMLIT_SERVER_ENABLE_XSRF_PROTECTION=true
STREAMLIT_SERVER_ENABLE_CORS=false
```

Security Considerations

Authentication & Authorization

⚠️ **Current Implementation**: Demo authentication for development ✅ **Production Recommendation**: Implement enterprise SSO

```python
# Production authentication example (to be implemented)
from azure.identity import DefaultAzureCredential
from msal import ConfidentialClientApplication

# Azure AD authentication
def authenticate_user():
    app = ConfidentialClientApplication(
        client_id=os.getenv("AZURE_CLIENT_ID"),
        client_credential=os.getenv("AZURE_CLIENT_SECRET"),
        authority=os.getenv("AZURE_AUTHORITY")
    )
    # Authentication logic
```

Data Security

1. **Encryption in Transit**: Use HTTPS in production
2. **Encryption at Rest**: Enable MongoDB encryption
3. **API Keys**: Store in environment variables, never in code
4. **Input Validation**: Validate all user inputs
5. **SQL Injection**: Use parameterized queries (MongoDB is NoSQL but still validate)

Best Practices

```python
# ☑ Good: Environment variables
api_key = os.getenv("API_KEY")

# ✗ Bad: Hardcoded secrets
api_key = "sk-1234567890abcdef"

# ☑ Good: Input validation
if not company_name or len(company_name) > 100:
    raise ValueError("Invalid company name")

# ☑ Good: Error messages without sensitive data
logger.error("Database connection failed")
```

```python
# ❌ Bad: Exposing sensitive information
logger.error(f"Connection failed: {connection_string}")
```

MongoDB Security

```javascript
// Enable authentication
use admin
db.createUser({
  user: "i2poc_admin",
  pwd: "secure_password",
  roles: ["readWrite", "dbAdmin"]
})

// Enable IP whitelisting in MongoDB Atlas
// Use connection string with authentication
mongodb+srv://username:password@cluster.mongodb.net/i2poc
```

API Rate Limiting

```python
# Implement rate limiting for external APIs
from utils.api_optimizer import api_optimizer

# Track and limit API calls
@api_optimizer.rate_limit(max_calls=100, period=60)
def call_external_api():
    # API call logic
    pass
```

Troubleshooting

Common Issues

1. MongoDB Connection Error

Error:

```
Failed to connect to MongoDB at mongodb://localhost:27017
```

Solutions: - Verify MongoDB is running: `mongod` or check service status - Check `MONGODB_URL` in `.env` file - Test connection: `mongo --eval "db.version()"` - For Atlas: Verify IP whitelist and credentials

2. Missing Dependencies

Error:

```
ModuleNotFoundError: No module named 'streamlit'
```

Solutions: - Activate virtual environment: `venv\Scripts\activate` - Reinstall requirements: `pip install -r requirements.txt` - Verify Python version: `python --version` (should be 3.8+)

3. Port Already in Use

Error:

```
Address already in use: 0.0.0.0:8501
```

Solutions: - Use different port: `streamlit run app.py --server.port 8502` - Kill existing process: ```bash # Windows netstat -ano | findstr :8501 taskkill /PID /F

Linux/Mac lsof -ti:8501 | xargs kill -9 ```

4. AI Scoring Not Working

Error:

```
AI scoring returned default values
```

Solutions: - Check `DEEPSEEK_API_KEY` in `.env` - Verify API key is valid - Check API quota and limits - Review logs for specific error messages

5. Research Agent Failures

Error:

```
Tavily API error: 401 Unauthorized
```

Solutions: - Verify `TAVILY_API_KEY` in `.env` - Check API subscription status - Verify internet connectivity - Review Tavily API documentation for changes

6. Document Generation Errors

Error:

```
Failed to generate DOCX document
```

Solutions: - Verify `python-docx` is installed - Check write permissions in output directory - Ensure research data is complete - Review error logs for specific issues

Debug Mode

Enable detailed logging:

```python
# config.py or app.py
import logging

logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
```

Performance Issues

Slow Research: - Check internet connection speed - Verify API response times - Enable caching in `utils/cache_manager.py` - Reduce research depth if needed

Slow UI: - Clear Streamlit cache: `st.cache_data.clear()` - Optimize database queries with indexes - Reduce data displayed per page - Use pagination for large datasets

Logs Location

- **Streamlit Logs**: Terminal output where app is running
- **Application Logs**: Check logging configuration
- **MongoDB Logs**: `/var/log/mongodb/mongod.log` (Linux)
- **Docker Logs**: `docker logs <container_id>`

Appendix

A. Demo Credentials

Email	Password	Role	Access Level
user@example.com	password123	Employee	Submit ideas, view catalog
manager@example.com	password123	Manager	All employee + review ideas
director@example.com	password123	Director	All manager + strategic oversight

B. API Endpoints Reference

Tavily API: - Documentation: https://tavily.com/docs - Rate Limits: Check your plan - Search Endpoint: `/search`

Azure OpenAI: - Documentation: https://learn.microsoft.com/azure/ai-services/openai/ - Models: GPT-4o, GPT-4, GPT-3.5-turbo - Endpoint: `https://{resource-name}.openai.azure.com/`

DeepSeek API: - Documentation: https://platform.deepseek.com/docs - Models: deepseek-chat, deepseek-coder

C. Database Queries

Find all submitted ideas:

```
db.ideas.find({ status: "submitted" })
```

Find high-scoring ideas:

```
db.ideas.find({ ai_score: { $gte: 80 } }).sort({ ai_score: -1 })
```

Find ideas by department:

```
db.ideas.find({ "metadata.department": "Engineering" })
```

Update idea status:

```
db.ideas.updateOne(
  { _id: ObjectId("...") },
  { $set: { status: "approved" } }
)
```

D. Useful Commands

Streamlit:

```
# Clear cache
streamlit cache clear

# Run with config
streamlit run app.py --server.port 8502 --theme.primaryColor "#792a85"

# View config
streamlit config show
```

MongoDB:

```
# Start MongoDB
mongod

# Connect to database
mongo i2poc

# Show collections
show collections

# Count documents
db.ideas.count()

# Export data
mongoexport --db i2poc --collection ideas --out ideas.json
```

Python:

```
# Create virtual environment
python -m venv venv

# Activate (Windows)
venv\Scripts\activate

# Activate (Mac/Linux)
source venv/bin/activate

# Install requirements
```

```
pip install -r requirements.txt

# Freeze current packages
pip freeze > requirements.txt
```

E. Keyboard Shortcuts

Streamlit: - `R` - Rerun the app - `C` - Clear cache - `?` - Show keyboard shortcuts

F. Resources & Links

- **Streamlit Documentation**: https://docs.streamlit.io/
- **MongoDB Documentation**: https://docs.mongodb.com/
- **LangChain Documentation**: https://python.langchain.com/
- **Pydantic Documentation**: https://docs.pydantic.dev/
- **Azure OpenAI**: https://learn.microsoft.com/azure/ai-services/openai/

G. Changelog

Version 1.0.0 (December 2025) - Initial release - Core features: Idea submission, AI research, evaluation - Integration: Tavily, Azure OpenAI, MongoDB - UI: Streamlit with custom branding

H. Roadmap

Planned Features: - [ ] Real-time collaboration on ideas - [ ] Advanced analytics dashboard - [ ] Integration with project management tools - [ ] Mobile-responsive design - [ ] Multi-language support (German, English) - [ ] Automated email notifications - [ ] Idea comparison and benchmarking - [ ] Export to multiple formats (PDF, Excel) - [ ] Advanced search with AI - [ ] Integration with DexKo internal systems

I. Support & Contact

For technical support or questions: - Review this documentation - Check troubleshooting section - Review application logs - Contact development team

---

Document Version: 1.0.0
Last Updated: December 8, 2025
Maintained By: DexKo Innovation Team