Summarizer & Quiz Generator Agent with Streamlit

By Nida Khurram

🚀 Project Idea: Summarizer & Quiz Generator Agent

A powerful web-based agent that summarizes PDF documents and generates interactive quizzes, built with Streamlit, Gemini CLI, and Context7 MCP servers.

📋 Project Blueprint

Step 1: Prerequisites

Install Node.js and Gemini CLI

- Install Node.js from nodejs.org
- Run `npm install -g @google/gemini-cli`
- Test MCP Commands: `/mcp list`, `/mcp refresh`

MCP (Model Context Protocol) Setup

Context7 MCP Configuration

Add this configuration to your Gemini CLI config file (typically located at `.gemini/settings.json`):

```
json

{
	"mcpServers": {
		"context7": {
			"command": "npx",
```

```
                    "args": [
                        "-y",
                        "@upstash/context7-mcp"
                    ],
                    "env": {
                        "CONTEXT7_API_KEY": "YOUR_CONTEXT7_API_KEY_HERE"
                    }
                }
        }
}
```

Getting Your Context7 API Key:

1. Visit https://context7.com/
2. Sign up for a free account
3. Navigate to your dashboard to generate an API key
4. Replace `YOUR_CONTEXT7_API_KEY_HERE` in the configuration

Step 2: Create Project

```bash
# Create project directory
mkdir summarizer-quiz-agent
cd summarizer-quiz-agent

# Setup with UV (Python package manager)
# Option 1: Install via pip
pip install uv

# Option 2: Install via PowerShell (Windows)
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"

# Option 3: Install via curl (macOS/Linux)
curl -LsSf https://astral.sh/uv/install.sh | sh

# Initialize project and create virtual environment
uv init
```

```
uv venv

# Activate virtual environment
# Windows:
.venv\Scripts\activate

# macOS/Linux:
source .venv/bin/activate

# Install required packages
uv add streamlit openai-agents python-dotenv pypdf2
```

Required Packages:

- `streamlit` - Modern web app framework for data applications
- `openai-agents` - OpenAI/Gemini API client library
- `python-dotenv` - Environment variable management
- `pypdf2` - PDF text extraction library

Step 3: Create Main gemini.md

```
markdown

# Role: Senior Python AI Engineer

**Objective:** Build a "Summarizer & Quiz Generator Agent" using Streamlit and the
`openai-agents` SDK.

## 1. Project Overview
The goal is to develop an intelligent web application that processes PDF documents and
generates both summaries and quizzes.
* **UI:** Streamlit (Modern, interactive web interface)
* **Model:** Google Gemini model named `gemini-2.0-flash` (via OpenAI Agents SDK)
* **PDF Processing:** PyPDF2 for text extraction
* **Features:** PDF summarization and quiz generation
```

```
## 2. Critical Technical Constraints
**You must adhere to the following strict configuration rules:**

1.  **Zero-Bloat Protocol (CRITICAL):**
    * **Do NOT write extra code.** Focus strictly on core functionality.
    * **No "Hallucinated" Features:** If it's not in the SDK docs, do not invent it.
2.  **API Configuration:**
    * Use the **OpenAI Agents SDK** Python Library configured for Gemini.
    * **Base URL:** `https://generativelanguage.googleapis.com/v1beta/openai/`
    * **API Key:** Load `GEMINI_API_KEY` from environment variables.
    * **Model:** Use `OpenaiChatCompletionModel` adapted for Gemini.
3.  **SDK Specificity:** You are using `openai-agents` SDK. This is **NOT** the standard `openai` library.
4.  **Error Recovery Protocol:**
    * If you encounter a `SyntaxError`, `ImportError`, or `AttributeError` related to `openai-agents` during development, **STOP**.
    * Do not guess the fix. **You MUST call the `get-library-docs` tool again** to re-read the documentation.
5.  **Dependency Management:** Use `uv` for package management.

## 3. Architecture & File Structure
*Note: The current directory is the root. Do not create a subfolder.*

```text
.
├── .env                    # Environment variables
├── tools.py                # PDF processing & utility functions
├── agent.py                # Agent configuration & tool binding
├── app.py                  # Streamlit UI & Application Logic
├── temp_pdfs/              # Temporary PDF storage (Auto-created)
└── pyproject.toml          # UV Config
```

4. Implementation Steps

Follow this exact logical flow. Do not skip steps.
```

Step 1: Documentation & Pattern Analysis

Before writing any code, you must verify the SDK syntax.

1. **Action:** Use the MCP tool `get-library-docs` to fetch the official documentation for the `openai-agents` **SDK**.
2. **Analysis:** Deeply analyze the returned documentation. Look specifically for:
    a. How to define tools (decorators vs classes)
    b. How to initialize the `Agent`
    c. How to pass the `OpenaiChatCompletionModel` to the agent
    d. **Check:** If you are unsure, query the docs again

Step 2: Tool Implementation (`tools.py`)

Create the PDF processing functions **using the strict format found in Step 1**.

- **Functions:**
    - `extract_text_from_pdf(pdf_file_path: str) -> str`: Extracts text from PDF using PyPDF2
    - `save_uploaded_file(uploaded_file) -> str`: Saves uploaded file to temporary location
    - `cleanup_file(file_path: str)`: Removes temporary files
    - `generate_quiz_from_text(text: str, quiz_type: str) -> str`: Generates quiz based on text content
- **Format:** Ensure these are defined as tools recognizable by the `openai-agents` SDK

Step 3: Agent Configuration (`agent.py`)

Configure the LLM and Agent using the patterns found in Step 1.

- Initialize the Gemini client using the Base URL
- Initialize the `OpenaiChatCompletionModel` with `gemini-2.0-flash`
- **Bind Tools:** Import tools from `tools.py` and register them to the agent instance
- **System Prompt:** Set instructions for summarization and quiz generation tasks

Step 4: UI & Application Logic (`app.py`)

Build the Streamlit interface with the following components:

- **File Upload Section:**
  - Streamlit file uploader for PDF files
  - Display upload status and file details
- **Summary Section:**
  - Button to trigger summarization
  - Display summary in a clean, styled container
- **Quiz Generation Section:**
  - Button to generate quiz (appears after summarization)
  - Quiz type selector (MCQs or mixed-style)
  - Display quiz in an interactive format
- **Session State Management:** Store PDF text and summary between interactions

Step 5: Environment & Dependencies

- Create a `.env` template with Gemini API key
- List necessary packages in `pyproject.toml`
- **Smart Install:** Check dependencies and avoid redundant installations

5. Testing Scenarios

1. **PDF Upload:** User uploads a PDF document → System extracts text successfully
2. **Summary Generation:** Click "Generate Summary" → Agent produces concise summary
3. **Quiz Generation:** Click "Create Quiz" → Agent generates relevant MCQs
4. **Multiple PDFs:** Test with different types of PDFs (articles, textbooks, reports)

```
text

---


### Prompt for Gemini
```

```jsx
Kick off development according to the provided instructions.
```

Step 4: Implementation Details

File: tools.py

```python
import json
import os
from typing import Dict, Any
from PyPDF2 import PdfReader

def extract_text_from_pdf(pdf_file_path: str) -> str:
    """Extract text content from PDF file using PyPDF2"""
    try:
        reader = PdfReader(pdf_file_path)
        text = ""
        for page in reader.pages:
            text += page.extract_text() + "\n"
        return text
    except Exception as e:
        return f"Error extracting PDF text: {str(e)}"

def save_uploaded_file(uploaded_file) -> str:
    """Save uploaded file to temporary location"""
    # Create temp directory if it doesn't exist
    os.makedirs("temp_pdfs", exist_ok=True)
    file_path = f"temp_pdfs/{uploaded_file.name}"
    with open(file_path, "wb") as f:
        f.write(uploaded_file.getbuffer())
    return file_path

def cleanup_file(file_path: str):
    """Remove temporary files"""
    try:
        if os.path.exists(file_path):
```

```
            os.remove(file_path)
    except Exception as e:
        print(f"Error cleaning up file: {e}")

def generate_quiz_from_text(text: str, quiz_type: str = "mcq") -> str:
    """Generate quiz questions from text content"""
    # This function will be enhanced by the agent's capabilities
    return f"Quiz generation for {quiz_type} type would be handled by the agent"
```

File: agent.py

```
python

import os
from dotenv import load_dotenv
from openai_agents import Agent
from openai_agents.models import OpenaiChatCompletionModel

# Load environment variables
load_dotenv()

def create_agent():
    """Create and configure the AI agent with tools"""

    # Configure Gemini client
    model = OpenaiChatCompletionModel(
        model="gemini-2.0-flash",
        api_key=os.getenv("GEMINI_API_KEY"),
        base_url="https://generativelanguage.googleapis.com/v1beta/openai/"
    )

    # Import tools
    from tools import extract_text_from_pdf, generate_quiz_from_text

    # Create agent with tools
    agent = Agent(
        model=model,
        tools=[extract_text_from_pdf, generate_quiz_from_text],
        system_prompt="""You are a PDF processing expert. Your tasks:
```

```
        1. Summarize PDF documents clearly and concisely
        2. Generate relevant quiz questions based on PDF content
        3. Create multiple-choice questions with proper options
        4. Provide mixed-style quizzes when requested

        Always focus on the key concepts and main ideas from the document."""
    )

    return agent
```

File: app.py

```
python

import streamlit as st
import os
from agent import create_agent
from tools import extract_text_from_pdf, save_uploaded_file, cleanup_file


# Configure Streamlit page
st.set_page_config(
    page_title="PDF Summarizer & Quiz Generator",
    page_icon="📄",
    layout="wide"
)


# Initialize session state
if 'pdf_text' not in st.session_state:
    st.session_state.pdf_text = ""
if 'summary' not in st.session_state:
    st.session_state.summary = ""
if 'quiz' not in st.session_state:
    st.session_state.quiz = ""
if 'agent' not in st.session_state:
    st.session_state.agent = create_agent()


def main():
    st.title("📄 PDF Summarizer & Quiz Generator")
    st.markdown("Upload a PDF document to get a summary and generate interactive
```

```
quizzes!")

    # File upload section
    st.header("1. Upload PDF Document")
    uploaded_file = st.file_uploader("Choose a PDF file", type="pdf")

    if uploaded_file is not None:
        # Save uploaded file
        file_path = save_uploaded_file(uploaded_file)
        st.success(f"File uploaded successfully: {uploaded_file.name}")

        # Extract text from PDF
        if st.button("Extract Text from PDF"):
            with st.spinner("Extracting text from PDF..."):
                st.session_state.pdf_text = extract_text_from_pdf(file_path)

            if st.session_state.pdf_text and not
st.session_state.pdf_text.startswith("Error"):
                st.success("Text extracted successfully!")
                with st.expander("View Extracted Text"):
                    st.text_area("PDF Text", st.session_state.pdf_text, height=200)
            else:
                st.error("Failed to extract text from PDF")

        # Summary generation section
        st.header("2. Generate Summary")
        if st.session_state.pdf_text and not
st.session_state.pdf_text.startswith("Error"):
            if st.button("Generate Summary"):
                with st.spinner("Generating summary..."):
                    response = st.session_state.agent.run(
                        f"Please summarize this PDF content:
{st.session_state.pdf_text[:4000]}"
                    )
                    st.session_state.summary = response.content

                st.subheader("📋 Summary")
                st.info(st.session_state.summary)

        # Quiz generation section
```

```
        st.header("3. Generate Quiz")
        if st.session_state.pdf_text and not
st.session_state.pdf_text.startswith("Error"):
            col1, col2 = st.columns(2)

            with col1:
                quiz_type = st.selectbox(
                    "Select Quiz Type",
                    ["MCQs", "Mixed Style", "True/False", "Fill in the Blanks"]
                )

            with col2:
                if st.button("🎯 Create Quiz"):
                    with st.spinner(f"Generating {quiz_type} quiz..."):
                        response = st.session_state.agent.run(
                            f"Generate a {quiz_type} quiz based on this PDF content:
{st.session_state.pdf_text[:4000]}"
                        )
                        st.session_state.quiz = response.content

                    st.subheader("📝 Generated Quiz")
                    st.success(st.session_state.quiz)

        # Cleanup
        cleanup_file(file_path)

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

File: .env

```
text

GEMINI_API_KEY=your_gemini_api_key_here
```

File: pyproject.toml

```
toml
```

```toml
[project]
name = "summarizer-quiz-agent"
version = "0.1.0"
description = "PDF Summarizer and Quiz Generator Agent"
authors = [
    {name = "Your Name", email = "your.email@example.com"},
]
dependencies = [
    "streamlit>=1.28.0",
    "openai-agents>=0.1.0",
    "python-dotenv>=1.0.0",
    "pypdf2>=3.0.1",
]

[project.scripts]
summarizer-quiz-agent = "app:main"

[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"
```

Step 5: Running the Application

```bash
# Make sure virtual environment is activated
# Windows:
.venv\Scripts\activate

# macOS/Linux:
source .venv/bin/activate

# Install dependencies
uv sync

# Run the Streamlit app
streamlit run app.py
```

The application will start on http://localhost:8501

Step 6: Example Workflow

PDF Upload & Text Extraction:

```
text

User: Uploads "Machine Learning Basics.pdf"
System: "File uploaded successfully: Machine Learning Basics.pdf"
User: Clicks "Extract Text from PDF"
System: "Text extracted successfully!" [Shows extracted text in expandable section]
```

Summary Generation:

```
text

User: Clicks "Generate Summary"
System: [Processing...]
Output: "📋 Summary: This document covers fundamental machine learning concepts including supervised vs unsupervised learning, neural networks, and common algorithms like linear regression and decision trees..."
```

Quiz Generation:

```
text

User: Selects "MCQs" → Clicks "Create Quiz"
System: [Generating quiz...]
Output: "📝 Generated Quiz:


1. What is the main difference between supervised and unsupervised learning?
   A) Supervised uses labeled data, unsupervised uses unlabeled data
   B) Supervised is faster than unsupervised
   C) Unsupervised requires more computational power
```

```
   D) Both use labeled data


2. Which algorithm is commonly used for classification tasks?
   A) Linear Regression
   B) K-Means Clustering
   C) Decision Trees
   D) Principal Component Analysis


..."
```

Step 7: Troubleshooting

Common Issues:

1. **Gemini API Errors**
    a. Ensure GEMINI_API_KEY is set in `.env`
    b. Verify the base URL is correct
    c. Check API quota limits
2. **PDF Extraction Issues**
    a. Ensure PDF is not scanned/image-based
    b. Check file permissions
    c. Verify PyPDF2 installation
3. **Streamlit Performance**
    a. Use session state to avoid reprocessing
    b. Implement file cleanup after processing
    c. Limit text length for API calls
4. **Context7 Integration**
    a. Verify Context7 API key is configured
    b. Check rate limits (60 req/hour on free tier)
    c. Ensure MCP server is running

This project provides a complete solution for document processing and educational content generation! The agent can handle various PDF types and create meaningful summaries and interactive quizzes. 🎓✨
7i609i