Resume Parser - Take-Home Assignment

Problem Statement

Build a resume parsing system that extracts key information from uploaded resumes (PDF/DOCX format) and displays the results through both a REST API and web interface.

Requirements

1. Extract These Elements from Resumes

You must use an LLM (Large Language Model) for extraction.

Model to use: gemma3:1b running in local machine with ollma exposed port.

Extract: as many as you can

- **Contact Information** (Name, Email, Phone, Location)
- **Professional Summary/Objective**
- **Work Experience** (Company, Role, Duration, Responsibilities)
- **Education** (Degree, Institution, Year)
- **Skills** (Technical and soft skills)
- **Certifications** (if present)

2. FastAPI Backend

Create two endpoints:

```
POST /api/upload
- Accept PDF/DOCX file
- Process and extract information
- Return extracted data with unique document ID
- Handle errors appropriately

GET /api/resume/{document_id}
- Retrieve previously extracted resume data
- Return 404 if not found
```

Must Have:

- Proper error handling with meaningful HTTP status codes
- Request/response validation using Pydantic models
- Structured logging (use Python's logging or loguru)
- Auto-generated API docs (FastAPI default)

3. Streamlit UI

- **File Upload:** Accept PDF/DOCX files
- **Extract Button:** Trigger processing with loading state
- **Results Display:** Show extracted information in organized cards/sections
- Clean, readable layout with proper formatting

4. Code Quality

- Follow Python best practices (PEP 8)
- Use type hints
- Separate concerns (routes, services, models)
- Add docstrings
- Handle exceptions gracefully
- Include basic input validation
- **Use environment variables for API keys (never hardcode)**

Suggested Tech Stack

```
fastapi==0.104.1 uvicorn==0.24.0
streamlit==1.28.0 pymupdf==1.23.8
# PDF extraction python-docx==1.1.0
# DOCX extraction python-multipart==0.0.6
# File uploads pydantic==2.5.0
loguru==0.7.2

# LLM Integration
ollama # For local
```

Project Structure

```
resume-parser/
├── README.md
├── requirements.txt
├── .env.example             # Example environment variables
├── app/
│   ├── main.py               # FastAPI app
│   ├── models.py             # Pydantic models
│   ├── services/
│   │   ├── parser.py         # Resume parsing logic
│   │   └── llm_service.py    # LLM integration
│   └── utils/ │
└── logger.py
├── ui/
│   └── streamlit_app.py
└── data/
    └── uploads/              # Store uploaded files
```

What We're Evaluating

1. **Functionality** - Does it work? Can it extract information accurately?
2. **LLM Integration** - Effective use of LLM for extraction with good prompting
3. **Code Quality** - Clean, organized, readable code
4. **Error Handling** - Graceful failures with helpful messages
5. **API Design** - RESTful conventions, proper status codes
6. **Documentation** - Clear README with setup instructions