Source Code Documentation

1. Overview

This project, "PDF Analyser using AI," is designed to extract and analyze content from PDF documents using advanced AI models. It includes features like text extraction, summarization, and named entity recognition (NER), offering insights in an organized format.

2. Code Architecture

2.1 High-Level Overview

The project is divided into the following primary components:

1. **Frontend**: Handles user interaction and file uploads.
2. **Backend**: Processes user requests and coordinates with AI modules.
3. **AI Modules**: Implements text summarization, entity recognition, and OCR (if necessary).
4. **PDF Parser**: Extracts raw text and metadata from PDF files.
5. **Database (Optional)**: Stores user inputs and results for future access.

2.2 Folder Structure

```
├── app/
│   ├── static/                # Static files for the frontend (CSS, JS)
│   ├── templates/             # HTML templates for user interface
│   ├── main.py                # Main application file (entry point)
│   ├── pdf_parser.py          # PDF parsing logic
│   ├── ai_modules/
│   │   ├── summarizer.py      # Summarization model logic
│   │   ├── ner.py             # Named entity recognition logic
│   │   ├── ocr.py             # OCR processing (for scanned PDFs)
│   └── database.py            # Database connection and operations
├── env/                       # Virtual environment
├── requirements.txt           # Python dependencies
├── README.md                  # Project documentatio
```

3. Key Components

3.1 Frontend

- **File Location:** `templates/` and `static/`
- **Purpose:** Provides the user interface for uploading PDFs and displaying results.
- **Key Features:**
    - Upload PDF files.
    - View extracted summaries and entity recognition results.

3.2 Backend

- **File Location:** `main.py`
- **Purpose:** Coordinates requests between the frontend, AI modules, and PDF parser.
- **Endpoints:**

- `/`: Renders the home page.
- `/upload`: Handles PDF uploads and validates file format.
- `/process`: Calls parsing and AI models for processing.
- `/results`: Sends results back to the frontend.

3.3 PDF Parser

- **File Location:** `pdf_parser.py`
- **Purpose:** Extracts raw text and metadata from uploaded PDFs.
- **Functions:**
  - `extract_text(file_path)`: Reads text from PDF pages.
  - `extract_metadata(file_path)`: Retrieves metadata like author and page count.
- **Libraries Used:** PyPDF2, PDFMiner, or Tesseract (for OCR).

3.4 AI Modules

3.4.1 Summarization

- **File Location:** `ai_modules/summarizer.py`
- **Purpose:** Generates a concise summary from the PDF text.
- **Function:**

```
def generate_summary(text):
    ...
```

3.4.2 Named Entity Recognition (NER)

- **File Location:** `ai_modules/ner.py`
- **Purpose:** Extracts named entities like persons, dates, and locations.
- **Function:**

```
def extract_entities(text):
    ...
```

3.4.3 Optical Character Recognition (OCR)

- **File Location:** `ai_modules/ocr.py`
- **Purpose:** Converts scanned PDF images into text.
- **Function:**

```
def perform_ocr(image):
    ...
```

3.5 Database (Optional)

- **File Location:** `database.py`
- **Purpose:** Stores user-uploaded PDFs and their processed results.
- **Database Used:** SQLite or PostgreSQL.

---

4. Code Flow

4.1 Execution Flow

1. **Input:**
   User uploads a PDF via the frontend.
2. **Processing:**
   - PDF Parser extracts raw text and metadata.
   - AI modules process the text for summarization and entity recognition.
3. **Output:**
   Processed insights (summary and entities) are sent back to the frontend.

4.2 Interaction Between Modules

- `main.py` orchestrates the interaction:
  - Calls `pdf_parser.py` for text extraction.
  - Passes extracted text to AI modules (`summarizer.py`, `ner.py`, etc.).
  - Combines results and sends them to the frontend.

5. Key Code Snippets

5.1 File Upload Handler

```
@app.route('/upload', methods=['POST'])
def upload_file():
    file = request.files['file']
    if file and file.filename.endswith('.pdf'):
        file_path = save_file(file)
        return jsonify({"message": "File uploaded successfully", "file_path": file_path})
    else:
        return jsonify({"error": "Invalid file format"})
```

5.2 Text Summarization

```
from transformers import pipeline

summarizer = pipeline("summarization")

def generate_summary(text):
    return summarizer(text, max_length=150, min_length=30, do_sample=False)
```

5.3 Entity Recognition

```
import spacy

nlp = spacy.load("en_core_web_sm")

def extract_entities(text):
    doc = nlp(text)
    return [{"text": ent.text, "label": ent.label_} for ent in doc.ents]
```

6. Testing

6.1 Unit Testing

- Tests written for each module to ensure correctness.
- Example:

```
def test_extract_text():
    text = extract_text("sample.pdf")
    assert isinstance(text, str)
```

6.2 Integration Testing

- Tests interaction between components:
    - PDF parsing → Summarization → Result compilation.

6.3 Manual Testing

- Upload various types of PDFs (e.g., text-heavy, scanned) and validate outputs.

7. Future Enhancements

1. **Multi-Language Support:** Extend AI modules to support languages other than English.
2. **Real-Time Processing:** Improve speed for large PDFs.
3. **User Authentication:** Add user accounts for history and personalized insights.