🧠 AI Document Analyzer – Phase 2: Design Documentation

---

1. Phase Title

Phase 2: System Design

---

2. Objective

The goal of this phase is to develop a **comprehensive, modular, and scalable system design** for the AI Document Analyzer. This includes the architecture, data flow, components, and user interface based on the research from Phase 1. The design addresses key challenges such as handling multiple document formats, ensuring NLP reliability, and managing fallbacks for offline/local processing.

---

3. Activities Performed

3.1 System Architecture Design

- Adopted a **3-tier modular architecture**:
  - **Presentation Layer:** Flask-based web interface
  - **Business Logic Layer:** NLP processing and text analysis
  - **Data Layer:** Handles file uploads, logs, and output storage
- Followed **Client-Server model**:
  - Users upload documents via UI
  - Server processes them, returns analysis
- Enabled **dual-mode NLP processing**:
  - **Cloud-first (IBM Watson NLU)**
  - **Local fallback** (vaderSentiment, regex, transformers)

3.2 Component Design

Component	Description
Text Extraction Module	Handles different formats: PDFs (pdfplumber, PyMuPDF), DOCX (python-docx), TXT (standard file I/O), Images (pytesseract + preprocessing)
NLP Module	Sentiment (Watson/vader), Keywords/Entities (Watson/spaCy), Summarization & QA (Hugging Face transformers)
Web Interface	Flask-based with upload, keyword input, QA form, result display, and download options (JSON, TXT, CSV, PDF)
Logging Module	Logs errors and processing details: logs/analyzer.log, logs/app.log

---

3.3 Data Flow Design

1. **User uploads** document via Flask UI
2. File is **temporarily stored** in static/uploads/
3. File type is **detected** and appropriate extraction module is invoked
4. Extracted text is passed to **NLP analysis pipeline**
5. Results are:
   - Displayed on the frontend
   - Saved in static/outputs/ as TXT/JSON
6. Clean-up process **removes temporary files** after response is served

---

3.4 User Interface Wireframes (Planned)

Page	Elements
Home Page	File Upload box, Keyword input, Submit button
Results Page	Summary output, Sentiment score, Keywords list, Entity highlights, Charts (Chart.js)
QA Page	Text input box for user questions, Display of AI-generated answers

Page	Elements
Error Display	Alert boxes for: unsupported format, large file, upload failure, API errors

---

3.5 Error Handling and Fallbacks

Risk	Fallback / Strategy
IBM Watson NLU API failure	Use vaderSentiment + regex + spaCy/transformers locally
OCR on noisy images	Preprocess with OpenCV (grayscale, blur, thresholding) before pytesseract
Large file or unsupported type	Validate file size and extension, return custom error messages to user
Unexpected user inputs	Form validations, user-friendly error alerts

---

4. Deliverables

Document/Asset	Status
✅ System Architecture Diagram	Created using 3-layer design
✅ Data Flow Diagrams	Defined for document lifecycle
✅ UI Wireframes (mockups)	Sketched using Figma and Paper
✅ Component Specifications	Prepared for each module
✅ Error Handling Plan	Integrated with backend design

---

5. Outcomes

- Well-structured and **scalable architecture** prepared
- Modular component design ensures **ease of implementation and testing**
- Detailed **data flow and user interaction workflow** defined
- Ready-to-build **UI and error management strategy** in place

---

6. Next Steps: Phase 3 – Development

- Begin development of Flask application and backend modules
- Integrate and test each NLP feature independently
- Build frontend templates and wire them to Flask routes
- Conduct unit tests and performance validation for each task