Automated Information Extraction from Scanned Documents

1. **Project Title:** API Service for Information Extraction from Scanned Documents
2. **Project Goal/Objective:** To develop and deploy an API service that accepts scanned document images (e.g., invoices, receipts) and automatically extracts predefined key information fields (e.g., invoice number, date, total amount, vendor name) and potentially a text summary using OCR and NLP techniques.
3. **Problem Statement/Background:** Many businesses still handle paper or scanned documents like invoices and receipts. Manually extracting information for data entry into accounting or other systems is time-consuming, tedious, and prone to errors. An automated system can significantly speed up processing, reduce errors, and lower costs.
4. **Scope:**
    - **In Scope:** Sourcing or creating a dataset of scanned documents (focus on a specific type, e.g., invoices); implementing an Optical Character Recognition (OCR) step to convert images to text; developing NLP techniques (e.g., Named Entity Recognition - NER, regex) to identify and extract target fields; potentially implementing text summarization for document content; building a REST API that accepts an image file and returns extracted fields (e.g., in JSON format) and/or a summary; containerizing and deploying the service.
    - **Out of Scope:** Handling highly complex or varied document layouts beyond the chosen type; handwritten text recognition (unless using advanced cloud services); building a UI; real-time processing of high volumes.
5. **Key Deliverables:**
    - Dataset of scanned documents (or documentation of source).
    - Code implementing the OCR and NLP extraction pipeline.
    - Documented API specification (e.g., OpenAPI/Swagger).
    - A containerized (Docker) REST API application.
    - Deployed API endpoint accessible via URL, capable of processing sample documents.
    - Final technical report detailing data, methods, API design, deployment, results, and limitations.
    - Final presentation demonstrating the service with sample documents.
6. **Potential Datasets to Get Started:**
    - **FUNSD Dataset (From Understanding to Natively Digitized):** A dataset for Form Understanding in Noisy Scanned Documents, includes annotations for fields. Good for training/evaluating structured information extraction.
        - *Link:* https://guillaumejaume.github.io/FUNSD/

- **RVL-CDIP (Ryerson Vision Lab Complex Document Information Processing):** Large dataset of scanned documents, categorized (letter, invoice, resume, etc.). Good source of raw images, but may require manual annotation for specific field extraction.
  - *Link:* https://www.cs.cmu.edu/~aharley/rvl-cdip/
- **CORD (Consolidated Receipt Dataset for Post-OCR Parsing):** Focused on receipts, includes scanned images and annotated entities.
  - *Link:* https://github.com/clovaai/cord
- **Self-Created Dataset:** Students can scan sample invoices/receipts (removing sensitive info) to create a small, targeted dataset.

7. **Potential Technical Approach:**
   - **OCR:** Use open-source libraries like Tesseract (via `pytesseract` wrapper) or leverage cloud-based OCR services (Google Cloud Vision AI OCR, Azure Cognitive Services for Vision - Read API, AWS Textract) which often provide higher accuracy and handle layouts better.
   - **NLP:** Use regex for simple patterns (dates, amounts). For more complex field extraction, use NER techniques (spaCy, or fine-tuning models like BERT/LayoutLM from Hugging Face). Text summarization could use pre-trained models (e.g., BART, T5).
   - **API Framework:** Flask or FastAPI.
   - **Containerization:** Docker. Ensure OCR engines/dependencies are correctly installed in the container.
   - **Cloud Platform:** AWS/GCP/Azure. Consider using cloud provider's integrated AI services (Vision, Textract, Cognitive Services) as part of the pipeline, or deploy custom models/Tesseract within containers (e.g., on Cloud Run, Azure Container Apps, Fargate).
8. **Deployment Requirements & Tips:**
   - **Requirement:** The final system must be deployed as a containerized REST API on AWS, GCP, or Azure, capable of accepting a document image (e.g., via POST request) and returning structured extracted data (JSON). The endpoint URL must be functional.
   - **Tips for Students:**
     - **OCR Choice Matters:** Compare local OCR (Tesseract - free, more setup) vs. Cloud OCR (easier API call, potentially higher accuracy, cost implications). Using cloud OCR might simplify the container build.
     - **Pipeline Design:** Decide if OCR and NLP happen sequentially within one API call, or if they could be separate steps (more complex). For this project, a single call is likely sufficient.

    - **Container Size:** OCR engines and large NLP models can make containers large. Be mindful of image size limits on some deployment platforms (especially serverless functions). Use multi-stage builds in Docker if possible to keep the final image smaller.
    - **Handling Files in APIs:** Learn how Flask/FastAPI handle file uploads in POST requests.
    - **Error Handling:** What happens if OCR fails or fields aren't found? Implement robust error handling and return meaningful responses.
    - **Cloud Services Integration:** If using cloud AI services (e.g., Google Vision AI), manage API keys and SDKs securely within the deployed application (use environment variables/secret managers).
    - **Iterative Deployment:** Deploy a basic version first (e.g., just OCR), then add NLP extraction, redeploy. Test frequently.
    - **Resource:** Utilize documentation for the chosen OCR tools, NLP libraries, API framework, and cloud deployment service. The Cloud Computing for Machine Learning and NLP courses are highly relevant.
9. **Potential Challenges:** OCR accuracy on varied document quality/layouts; training/fine-tuning NER models effectively; handling PDF vs. image inputs; managing dependencies in Docker (especially Tesseract); deployment complexity.
10. **Success Metrics:** Accuracy of extracted fields (precision, recall); success rate of processing documents; usability and reliability of the deployed API; quality of code/documentation.
11. **Team Size / Duration:** 3-4 Students / 14 Weeks