Lab details:

- Natural language Processing (NLP) - CS429
- Lab Session-6
- Date- 22/09/2025 to 27/09/2025
- Marks – 10

Instruction:

1. The name of your Python file must follow this format: **studentID_Name**
   - Example: If Student ID = 2022B3A70617P and Name = Nikhil Manvendra Singh
   - File name should be: 2022B3A70617P_Nikhil_Manvendra_Singh.ipynb
2. Submission Deadline: 27/09/2025 11:55 PM
3. Students are required to submit zip file containing all necessary document related to project

Introduction to Lab Report Digitisation:

In many hospitals and labs, test results come as printed papers or scanned PDFs. Computers cannot easily understand these images. The goal of this lab is to teach how to convert those scanned lab reports into clean, organized digital data (JSON). This makes it easy to search, analyse, and use the results in other programs.

Lab-6: Home assignment

Project Title: Lab Report Digitization and Structured Data Extraction

Aim: To develop a system that can automatically extract and digitize patient details and test results from laboratory reports into a structured format.

Objective: The goal of this project is to build a system that can take a lab report file (PDF or image), recognize the important information (test names, values, units, patient details), and convert it into a structured JSON format. The system should also include a machine learning component so that it can adapt and handle new lab report formats over time.

Key Features:

Sr. No	Feature	Description	Example
1	File Input & Preprocessing	Accepts lab reports in PDF, JPG, or PNG. Converts PDFs to images, cleans and deskews scanned files.	Upload a scanned blood test PDF, system converts it to image for processing.
2	OCR & Text Extraction	Uses OCR to read text from images and capture word positions (bounding boxes).	Detects and extracts "Name: John Doe, Age: 42" from scanned text.
3	Rule-Based Extraction	Applies regex and positional rules to detect patient info and test results.	Rule finds "Age:" and extracts the number that follows (e.g., 42).
4	Machine Learning Extraction	Trains a model using corrected reports to handle new formats and layouts.	Model learns that "Pt Name" also means Patient Name in another report.
5	JSON Output	Converts extracted data into structured JSON with patient details and test results.	"patient":{"name":"John Doe","age": 42 }, "tests":[{"name": "Hemoglobin", "value": 13.5, "unit": "g/dL" }, { "name": "WBC", "value": 7600, "unit": "/µL" }]
6	Confidence Scoring	Assigns a confidence score to each extracted field to flag low-confidence items.	"Hemoglobin = 13.5 g/dL (confidence 0.92)"
7	Human-in-the-Loop Review	Displays extracted data to users for confirmation or correction before saving.	User corrects "Jhn Deo" → "John Doe" before final save.
8	Continuous Learning	Stores corrections as training data so the model improves over time.	Next time, system correctly extracts "John Doe" without error.
9	API & Demo	Provides a REST API to upload reports and return JSON. Use simple UI lets users upload a report and see extracted results.	User uploads a report via web form → sees structured JSON instantly.

Tech Stack (Suggested):

Sr.No	Component	Options	Example Use
1	Language / Framework	Python (FastAPI)	Build the backend REST API to upload reports and return JSON.
2	OCR Engine	Tesseract, PaddleOCR, EasyOCR	Extract text from scanned lab reports (PDF/JPG/PNG).
3	ML Framework	scikit-learn, PyTorch, TensorFlow	Train models to identify fields like Name, Age, Test, Value, Unit.
4	Database (Optional)	MongoDB, PostgreSQL	Store structured JSON reports and training data for model improvement.
5	Image Processing	OpenCV	Preprocess images (clean, deskew, enhance text quality).
6	API Documentation	Swagger / OpenAPI	Automatically generate documentation and allow easy testing of APIs.
7	Dataset	Kaggle or similar	https://www.kaggle.com/datasets/dikshaasinghhh/bajaj

Deliverables

1. Working codebase (GitHub repo).
2. REST API with documentation (Swagger/OpenAPI).
3. Sample input files and the corresponding JSON outputs.
4. Small labelled dataset of lab reports for training/testing.
5. Final presentation/demo of the system.

Learning Outcomes for Student

1. Hands-on experience in OCR and document processing.
2. Exposure to machine learning for text extraction.
3. Experience in API development and data modelling.
4. Understanding of real-world challenges like noisy scans, multiple formats, and unit standardization.

Stretch Goals (Optional)

1. Highlight extracted fields visually on the scanned report.
2. Confidence scoring for extracted values (e.g., mark low-confidence fields for manual review).
3. Auto-mapping lab tests to standard medical codes.

Implementation Steps Module Wise

Module-1: File Input & Preprocessing

Goal: get a clean image from any PDF/photo so OCR works well.
Tools: `pdf2image`, `OpenCV`, `Pillow` (Python)

Steps:

1. Accept upload (PDF / JPG / PNG).
2. If PDF — convert each page to a high-resolution image (300 dpi).
3. Convert color → grayscale.
4. Deskew (rotate slightly if tilted).
5. Denoise / sharpen / apply OTSU threshold if needed.
6. Save cleaned images for OCR.

Expected output: one cleaned image file per page (e.g., `page_01.png`).
Tip: test with 3–5 sample PDFs to tune thresholds.

Module-2: OCR & Tokenisation

Goal: read text and get each word with its position.
Tools: `Tesseract` or `PaddleOCR`, pytesseract `image_to_data` (Python)

Steps:

1. Run OCR on each cleaned image.
2. For every recognized word/token record: `text`, `left,top,width,height`, `confidence`.
3. Save token list per page as JSON or CSV.

Expected output: `tokens_page_01.json` containing a list of tokens with bboxes and confidences.
Tip: keep OCR confidence — it helps choose which items need human review.

Module-3: Rule-Based Extraction (fast baseline)

Goal: get quick, useful extractions using simple rules so system works immediately.
Tools: Python `re` (regex), simple positional heuristics

Steps:

1. Join nearby tokens into lines (group tokens by y-coordinate).
2. Use regex rules for obvious fields:
   - `Name[:\s]+(.+)`
   - `Age[:\s]+(\d+)`

   - `Patient ID[:\s]+(\S+)`
3. For test tables: detect rows where pattern is `TestName Value Unit` using spacing or bounding boxes alignment.
4. Build a JSON output with fields and test rows.

Expected output: a preliminary JSON per report with fields found and `confidence` derived from OCR + rule match strength.
Tip: this gets you usable results fast and provides training seeds.

Module-4: Human-in-the-Loop (HITL) UI

Goal: let a human confirm or correct extracted data before it becomes final.
Tools: small web UI (HTML + simple JS) or form in a notebook / minimal React form

Steps:

1. Show extracted JSON in an easy-to-read form (patient fields + a table for tests).
2. Highlight low-confidence fields (color or flag).
3. Provide `Confirm`, `Edit`, `Reject` buttons for each field/row.
4. When user corrects, save corrected example to a `corrections/` folder (JSON format): includes original tokens + corrected labels.

Expected output: `confirmed_report_001.json` and `corrections/correction_001.json`.
Tip: save both original OCR tokens and the corrected final labels — both are needed for training.

Module-5: Trainable Model (supervised learning)

Goal: learn from corrections so future extractions improve (not RL — supervised).
Tools: `spaCy` token classifier or a Transformers token-classifier (Hugging Face), optionally `LayoutLM` for bbox-aware training

Steps:

1. Prepare training data: for each token include `text`, `bbox`, and a label (BIO: `B-NAME`, `I-NAME`, `B-VALUE`, etc.). Use corrections saved in Module 4.
2. Train a simple model (start with spaCy) for token classification.
3. Validate on a small holdout set (compute token-level F1).
4. Export the model (saved weights) for inference.

Expected output: a saved model file (e.g., `model_epoch3.pkl` or HF model folder).
Tip: start small — 50–200 corrected reports are enough to see improvement on similar layouts.

Module-6: Inference — combine rules + model

Goal: use model and fallback rules together at runtime.
Tools: Python inference script (loads OCR tokens, runs model, applies rules)

Steps:

1. Run OCR (Module 2) → tokens.
2. Run model to tag tokens with labels.
3. Apply postprocessing rules to group tokens into fields and tests.
4. If model/confidence low, fallback to rule-based result or mark for human review.
5. Produce final `pending` JSON (awaiting confirmation if confidence low).

Expected output: `pending_report_001.json` with confidence per field.
Tip: combine model + rules to reduce errors while model learns.

Module-7: API & Demo UI

Goal: let users upload reports and get structured JSON via a REST endpoint; optional web demo to confirm results.
Tools: `FastAPI` (Python) + small frontend (HTML/React) + Swagger auto-docs

Steps:

1. Create `/upload` POST endpoint: accepts file, runs pipeline (preprocess → OCR → inference).
2. Return JSON response (extracted fields + confidences).
3. Add a demo page where user can upload and then press Confirm/Edit.
4. Connect Confirm button to store corrected JSON (Module 4).

Expected output: running local server `http://localhost:8000/docs` (Swagger) and demo page.
Tip: keep the demo simple — a single-page form is enough for students.

Module-8: Storage & Continuous Learning

Goal: save confirmed reports and use them to retrain the model periodically.
Tools: `MongoDB` or simple folder storage + periodic training script

Steps:

1. Store confirmed JSONs in DB or folder `/final_reports/`.
2. Add corrected examples to `/training_data/` (token-level labels).
3. Run `retrain.py` weekly or when you have N new corrections.
4. Version models (save with timestamp or checkpoint id).

Expected output: improved model versions and an organized dataset.
Tip: keep a small CSV or JSON index of files for easy tracking.

Module-9: Evaluation & Quality Checks

Goal: measure and show how well the system works.
Tools: `sklearn.metrics` for precision/recall/F1, simple scripts for end-to-end accuracy

Steps:

1. Use a test set of confirmed reports.
2. Compute token-level precision/recall/F1 for each label (Name, Age, Value, Unit).
3. Compute report-level accuracy (percentage of reports with all critical fields correct).
4. Create a short report showing errors to improve rules/model.

Expected output: an evaluation summary (e.g., F1 scores, error cases).
Tip: show students examples of common errors to learn how to improve preprocessing or rules.

Module-10: Documentation & Student Checklist

Goal: give students a short checklist and simple commands to run the system.
Tools: `README.md`, quick run scripts

What to include:

- How to install: `pip install -r requirements.txt`
- How to run server: `uvicorn app:app --reload`
- How to upload a file and confirm it via demo UI
- A short checklist (preprocess ok, OCR ok, confirm before save)

Note for Students:

The steps and modules described above are given only as guidance to help you understand how to build a lab report digitization system in a structured way.

- If you follow these modules, you will cover all the important parts: input, OCR, extraction, confirmation, machine learning, API, and evaluation.
- However, if you have similar or alternative idea (for example, using a different OCR tool, a different database, or a new way to design the UI), you are free to proceed with your own approach.
- The main expectation is that your solution should:
  1. Accept lab reports as input (PDF/image)
  2. Extract key fields and test results
  3. Output them in structured format (like JSON)

4. Include a way to correct or confirm the results

Remember: Creativity and innovation are encouraged, feel free to improve on these steps and you can apply new methods as long as the core goal is achieved.