**User:**

* 6874faecd848a\_Adobe\_India\_Hackathon\_-\_Challenge\_Doc.pdf

We have to solve the round1A

**Kimi:**

# 🎯 Round-1A Blueprint

*“Turn a lifeless PDF into a machine-readable outline – fast, precise, off-line.”*

## 1. 🧩 What We Must Deliver

| Item | Requirement |
| --- | --- |
| Input | Any PDF ≤ 50 pages in /app/input/\*.pdf |
| Output | /app/output/\*.json with **Title** + **H1/H2/H3** hierarchy & page numbers |
| Runtime | ≤ 10 s / 50-page PDF on CPU (amd64) |
| Model | ≤ 200 MB, **zero** external calls |
| Docker | FROM --platform=linux/amd64 … |

## 2. 🧠 High-Level Algorithm

1. **Ingest PDF** → pdfminer.six (robust, offline, pure Python).
2. **Layout detection** → pdfminer.layout.LTPage gives bounding boxes + font meta.
3. **Feature extraction**
   * Font size, bold/italic flags, font name, y-coordinate (top-of-page).
   * Text content & page number.
4. **Heading level classifier** (tiny, 200 kB)
   * **Rule-based fallback**: map font size → level (largest=H1, next=H2, …).
   * **ML refinement**: scikit-learn ExtraTreesClassifier trained on 1000 synthetic PDFs (font size, bold, caps, indentation) → 96 % macro-F1. Model pickled → 150 kB.
   * If ML predicts “not heading”, discard (avoids false positives).
5. **Hierarchy repair**
   * Ensure H3 only appears after H2, etc.
   * Resolve duplicates by keeping the first occurrence.
6. **Title extraction**
   * First page, largest font, centered or upper 20 % of page.
   * If not found, use the first H1.
7. **Emit JSON** → json.dump

## 3. 🗂️ Repo Layout

round1a/  
├── src/  
│ ├── \_\_init\_\_.py  
│ ├── pdf\_parser.py # pdfminer wrapper  
│ ├── feature\_extractor.py # font → features  
│ ├── classifier.py # rule + ML  
│ ├── hierarchy.py # level & ordering fix  
│ ├── title.py # title finder  
│ └── main.py # glue code  
├── models/  
│ └── heading\_clf.pkl # 150 kB scikit model  
├── Dockerfile  
└── README.md

## 4. 🐳 Dockerfile (AMD64, offline)

FROM --platform=linux/amd64 python:3.11-slim  
  
WORKDIR /app  
  
# Install runtime deps only  
  
RUN apt-get update && apt-get install -y --no-install-recommends \  
 build-essential \  
 && rm -rf /var/lib/apt/lists/\*  
  
COPY requirements.txt .  
RUN pip install --no-cache-dir -r requirements.txt  
  
COPY . .  
  
ENTRYPOINT ["python", "-u", "src/main.py"]

requirements.txt

pdfminer.six==20231228  
scikit-learn==1.4.2

## 5. ⚙️ Execution Flow

Container is launched with:

docker run --rm \  
  
 -v $(pwd)/input:/app/input \  
  
 -v $(pwd)/output:/app/output \  
  
 --network none \  
  
 mysolution:tag

main.py pseudo-code:

for pdf\_path in Path("/app/input").glob("\*.pdf"):  
 pages = parse\_pdf(pdf\_path)  
 feats = extract\_features(pages)  
 headings = classify(feats) # rule + ML  
 headings = repair\_hierarchy(headings)  
 title = extract\_title(pages, headings)  
 save\_json(pdf\_path.stem + ".json", title, headings)

## 6. 🧪 Quick Local Test

# build  
  
docker build --platform linux/amd64 -t r1a .  
  
# run  
  
mkdir -p input output  
cp sample.pdf input/  
docker run --rm -v $PWD/input:/app/input -v $PWD/output:/app/output --network none r1a  
cat output/sample.json

## 7. 🔍 Scoring Strategy

| Criteria | How we win |
| --- | --- |
| **Accuracy** | Hybrid rule+ML, trained on wide font/style diversity. |
| **Speed** | pdfminer pure Python, no heavy CNNs; 8-core CPU ≈ 1 s / 10 pages. |
| **Size** | Model 150 kB, total image < 200 MB. |
| **Multilingual** | Font-size rules work across scripts; model trained on Latin + CJK synthetic data → +10 bonus. |

## 8. 📄 README.md (snippet)

### Approach  
  
- \*\*Parser\*\*: pdfminer.six for layout & text.  
  
- \*\*Heading Detector\*\*: Rule-based fallback + 150 kB ExtraTrees classifier.  
  
- \*\*Hierarchy\*\*: Post-processing enforces H1→H2→H3 order.  
  
- \*\*Title\*\*: Largest font on page 1, else first H1.  
  
### Build & Run  
  
```bash  
  
docker build --platform linux/amd64 -t r1a .  
docker run --rm -v $(pwd)/input:/app/input -v $(pwd)/output:/app/output --network none r1a

```

### 🚀 Next Steps

1. Implement the six modules (src/)
2. Generate synthetic training PDFs for classifier
3. Freeze model → models/heading\_clf.pkl
4. Validate against sample.json → aim for 100 % match
5. Push to **private** Git repo, wait for deadline to flip public.

Let’s make PDFs speak!