FYP - Early-1

awesome — here's a super-concrete, role-by-role prototype plan you can execute immediately. It tells each of you **what to build, how to build it, what data to use, the exact files to touch, acceptance criteria, and a day-by-day schedule** to land the end-to-end demo.

Prototype Goal (Sep 29)

A working, offline, end-to-end **Lung-CT** demo:

- DICOM **ingest → analyze → results viewer** (masks + saliency).
- **Detector+segmenter v1**, **risk head v0**, **Grad-CAM++**.
- Outputs: **findings.json**, **Clinician PDF (EN)**, **Patient PDF (EN/HI)**, **FHIR DiagnosticReport & ImagingStudy** JSON.
- Queue-based, crash-safe, **⩽60–90 s/study** on RTX 4060.
- Mini-eval: latency, 1 FROC point, a few Dice examples, screenshots.

Repo root: `lungct/` (layout below). Everyone sticks to this structure.

```bash
lungct/
  app/            # APIs & services
  ui/             # Streamlit (or React) UI
  ml/             # models & inference
  data/           # proto_pack (10–20 CTs)
  eval/           # quick metrics
  artifacts/      # outputs (json/pdf/fhir/overlays)
  config.yaml
  requirements.txt
  README.md
```

Shared setup (all)

- Environment: Python 3.10; PyTorch (CUDA), pydicom, SimpleITK or MONAI, numpy, fastapi, uvicorn, streamlit, reportlab/weasyprint, onnxruntime (CPU ok for now), rq/redis or celery, pydantic, pillow, scikit-image.
- Git: branch per role (`feat/ml-a`, `feat/ml-b`, `feat/data-mlops`, `feat/frontend`); PRs into `main`.
- Data: create a **proto_pack** of **10–20 CT studies** (de-identified). Prefer **LIDC-IDRI** DICOMs (with XML annotations) for 10 subjects + **LNDb v4** 5 subjects (NIfTI+CSV) if you already have it. Keep total size ⩽ 3 GB.
- For 5–8 nodules, produce **rough masks** (slice-wise contours) so we can compute Dice in the demo. (Use 3D Slicer or draw simple masks around nodules — good enough for prototype.)

Dinesh — ML-A (Vision Lead)

Mission

Ship the **detector+segmenter v1**, **risk head v0**, and **Grad-CAM++ overlays**, callable via one function: `run_inference(study_dir) -> findings.json + overlays`.

What to build (files & functions)

- `ml/preprocess.py`
  - `resample_to_iso(vol, spacing=1.0)` → SimpleITK/MONAI.
  - `hu_clip(vol, low=-1000, high=400)`, `normalize01(vol)`.
  - `make_2p5d_packs(vol, k=5, stride=1)` → returns tiles (C=5, H, W) + z indices.
- `ml/detector.py`
  - **v1 candidate generator** (fast): LoG blob detector across slices **or** small CenterNet-lite head.
  - `detect_candidates(packs) -> [(z,y,x,score)]` (keep top-K per slab; N≈50–200).
- `ml/segmenter.py`
  - **Mobile-UNet / Half-UNet** 2.5D (single-class).
  - `segment_cubes(cubes) -> masks` (threshold→morphology→3D merge).
- `ml/risk.py`
  - `extract_deep_features(cubes)` (global avg pooled penultimate layer).
  - `quick_radiomics(mask, vol)` (sphericity, volume, mean HU).
  - `risk_head(features+radios) -> p_malignant` (tiny MLP).
- `ml/xai.py`
  - `gradcam_pp(model, cube) -> heatmap` (over risk head's features).
  - Compose **mask + heatmap** overlays (save as PNGs).
- `ml/infer.py`
  - Tie it all:

```python
def run_inference(study_dir, out_dir):
    vol = load_dicom_series(study_dir)
    vol = normalize01(hu_clip(resample_to_iso(vol)))
    packs = make_2p5d_packs(vol)
    cands = detect_candidates(packs)
    cubes = extract_cubes(vol, cands, size=64)
    masks = segment_cubes(cubes)
    nods  = merge_3d_blobs(masks, cands)  # (centroid, size, vol)
    feats = extract_deep_features(cubes)
    radios = [quick_radiomics(m, vol) for m in masks]
    risk  = risk_head(concat(feats, radios))
    sal   = [gradcam_pp(risk_head, cb) for cb in cubes]
    findings = build_findings_json(nods, risk, sal)
    save_overlays(vol, masks, sal, out_dir)
    save_json(findings, out_dir/"findings.json")
```

Data to use

- **LIDC-IDRI** DICOMs (10 subjects). For a few nodules, read XML ROI to craft quick slice masks (or rough boxes).
- Optional: **LNDb v4** samples for extra masks.

Minimal training plan (fast)

- Train the 2.5D **Mobile-UNet** on your rough masks (patch-based). 4–6 hours on RTX 4060; batch 16; CE/Dice loss; early stop on val Dice.
- Detector v1 can be classical (LoG blob + threshold); upgrade later.

Acceptance criteria (for you)

- 1 study end-to-end runs in **⩽90 s**; produces **findings.json** + **mask & saliency PNGs**.
- **At least 3 cases** with reasonable masks; **Dice ⩾0.70** on the few labeled nodules.
- **One FROC point** on proto_pack: Sens@2 FP/scan computed by Teja's scripts.

Kowshik — ML-B (Reporting/NLP)

Mission

Own the **findings schema**, **validators**, **RSNA clinician report**, **patient EN/HI summaries**, **PDF generator**, **grounded Q&A**, and **FHIR exporter**.

What to build (files & functions)

- `app/validators.py`
  - `validate_findings(json)` → checks count/size/site, units (mm/mm^3), Lung-RADS rules.
- `app/reporter.py`
  - **Templates**:
    - `templates/clinician_en.md` (Technique, Findings, Impression, Lung-RADS, Recommendation).
    - `templates/patient_en.md`, `templates/patient_hi.md` (8th-grade, glossary).
  - `render_reports(findings.json) -> clinician.pdf, patient_en.pdf, patient_hi.pdf` (ReportLab/WeasyPrint).
- `app/fhir_export.py`
  - `to_fhir_diagnostic_report(findings.json) -> diagnostic_report.json`
  - `to_fhir_imaging_study(dicom_meta) -> imaging_study.json`
  - Reference observations for each nodule (size, location, risk).
- `app/qa_engine.py`
  - Grounded **Q&A**: given a question (e.g., "Why malignant?"), return an answer **only** from findings JSON + pointers (nodule ID, slices). (Rule-based now; LLM later.)
- `app/samples/`
  - `findings.schema.json` (final), `sample_findings.json`, `sample_fhir.json`.

Data to use

- **LNDb v4** report tables (if available) to sanity-check phrasing.

- Otherwise: craft 10 "reference" reports for proto_pack cases (manual) to test BLEU/ROUGE later.

Acceptance criteria (for you)

- `findings.schema.json` validated with at least **10 cases**.
- **Clinician PDF (EN)** and **Patient PDF (EN/HI)** render in **<2 s** each; no hallucinations (strict template).
- **FHIR DiagnosticReport & ImagingStudy** JSONs produce (valid JSON; cross-referenced IDs).
- Q&A returns **evidence-linked** answers ("Nodule N1 in RUL, 7.2 mm; features: solid, spiculation absent").

Teja — Data / MLOps

Mission

Deliver the **data pipeline, DVC-tracked proto_pack, job queue orchestration, and the eval/benchmark scripts**. Make the demo robust and reproducible.

What to build (files & functions)

- `ml/io.py`
  - `load_dicom_series(study_dir) -> (vol, meta)` using pydicom/SimpleITK.
  - `save_overlays(vol, masks, saliency, out_dir)` → axial PNG stacks.
  - `save_json(obj, path)`.
- `data/`
  - `proto_pack/` (10–20 CTs).
  - `dataset_card.md` (source, license, preprocessing).
  - DVC pipeline: `dvc init`; track raw → processed.
- `app/queue.py`
  - RQ/Celery workers; job model: **Ingested → Queued → Preprocess → Detect → Segment → Risk → Explain → Report → Export → Completed**; state persisted in DB.
- `eval/latency_bench.py`
  - Measure per-study time, per-stage breakdown, memory/VRAM peaks; CSV output.
- `eval/mini_froc.py`
  - On proto_pack, compute a **single FROC point**: Sens@2 FP/scan (require you to store GT centers for a handful of nodules).
- `eval/dice_smallset.py`
  - Dice & 95% HD on the few hand-labeled masks.

Data to use

- **LIDC-IDRI** DICOMs (10). Pull series with thin slices (⩽1.5 mm).
- Use LIDC XML to extract center coords for a few nodules (for mini-FROC ground truth).
- If LNDb v4 is available, include 3–5 NIfTI volumes as external test; otherwise skip.

Acceptance criteria (for you)

- One-command **repro**: `python ml/infer.py --study_dir ...` runs & writes outputs.
- Queue survives a kill/restart (job resumes).
- Produce **latency.csv**, **mini_froc.csv**, **dice_smallset.csv** in `/artifacts/metrics/`.
- `README.md` with **exact run commands** and environment pins.

Rahul — Frontend

Mission

Deliver the **offline viewer UI**: load CT, run analysis, show overlays (mask + saliency), bilingual text, downloads (PDFs, JSON), and clear progress/error states.

What to build (files & pages)

- Choose **Streamlit** (fastest) for the prototype.
- `ui/app.py` — pages:
    1. **Home / Ingest**: select DICOM folder/zip → POST `/ingest` → shows `study_uid`.
    2. **Analyze**: button calls POST `/analyze/{study_uid}` → poll `/status/{job_id}` → progress bar with ETA.
    3. **Results**:
        - Axial viewer (slice slider), window/level controls.
        - Toggle **Masks** and **Saliency**; click a nodule → show **size, risk, lobe**.
        - Buttons: **Download Clinician PDF**, **Patient PDF (EN/HI)**, **Findings JSON**, **FHIR JSON**.
        - **Language toggle** (EN/HI); high-contrast theme; large buttons.
- Error handling: friendly messages, retry button; show "offline" if ABDM sync isn't reachable.
- Store-and-forward: a local folder `outbox/fhir/` with queued JSONs; show counts.

Data to use

- Use Teja's **proto_pack**; develop with 1–2 small studies for fast iterations.
- Use `artifacts/samples/` for UI placeholders before ML is ready.

Acceptance criteria (for you)

- Clean flow: Ingest → Analyze → Results, **without touching the terminal**.
- Viewer is responsive for **512–768 px** axial slices; overlays toggle without lag.
- All 4 downloads work; language toggle flips Patient PDF labels.

Interfaces (what you all agree on)

FastAPI endpoints (Teja implements, Rahul consumes, ML writes outputs)

- `POST /ingest` → `{study_uid}` (server writes the DICOM path mapping).
- `POST /analyze/{study_uid}` → `{job_id}`.
- `GET /status/{job_id}` → `{progress: 0–100, stage: string, eta_s: int}`.

- `GET /results/{study_uid}` → `{links: {findings_json, overlays_zip, clinician_pdf, patient_en_pdf, patient_hi_pdf, fhir_json}}`.
- `POST /export/fhir/{study_uid}` → stores JSON to `outbox/fhir/`, returns path.

Findings JSON (Kowshik owns)

```json
{
  "study_uid": "1.2.840....",
  "meta": {"series_uid": "1.2.3", "spacing_mm": [1.0,1.0,1.0]},
  "nodules": [
    { "id":"N1", "lobe":"RUL", "type":"solid",
      "center_zyx":[128,192,240], "long_axis_mm":7.2, "vol_mm3":190,
      "risk":0.72, "uncertain": false,
      "slices":[127,128,129], "mask_path":"masks/N1_mask.npz",
      "saliency_path":"saliency/N1_axial_128.png"
    }
  ],
  "summary": {"nodule_count":1, "largest_mm":7.2, "lung_rads":"3"},
  "quality": {"motion":"mild","noise":"moderate"},
  "recommendation": "CT follow-up in 6 months"
}
```

Day-by-day plan to hit the demo

D-10 to D-8

- Teja: DICOM loader, resample; `ingest` & `analyze` endpoints; queue skeleton; dataset card & proto_pack.
- Dinesh: preprocess; LoG-based detector; cube extraction; Mobile-UNet code scaffold.
- Rahul: Streamlit skeleton (pages, file pickers, progress bar), dummy result page (reads `sample_findings.json`).
- Kowshik: finalize `findings.schema.json`; draft templates (clinician EN, patient EN/HI); validator v0.

D-7 to D-6

- Dinesh: segmenter train (4–6h); merge 3D blobs; size/volume.
- Teja: wire `ml/infer.py` into queue; save overlays; `/results` returns links; start latency logger.
- Rahul: axial viewer with overlays; window/level; language toggle.
- Kowshik: PDF generation; FHIR builder v0; Q&A stub grounded in JSON.

D-5 to D-4

- Dinesh: risk head v0; Grad-CAM++; save saliency PNGs.
- Teja: mini_froc & dice scripts (with 3–5 GT nodules); latency CSV.
- Rahul: download buttons; error handling; "store-and-forward" outbox panel.
- Kowshik: validator v1 (count/size/site); finalize templates; sample FHIR JSON.

D-3

- Integrate: end-to-end on 3 studies; fix timings; crash/restart resume; audit log table.
- Collect screenshots.

D-2

- Dry-run demo; fill metrics tables; polish PDFs (fonts for Hindi).

D-1

- Freeze repo; export **Prototype.pdf** (screenshots, metrics, sample JSON/PDFs).

Per-role acceptance checklist (copy into your own todo)

Dinesh (ML-A)

- ☐ `run_inference()` outputs `findings.json`, masks, saliency for any study.
- ☐ 3 demo cases with reasonable masks; Dice ⩾0.70 on labeled nodules.
- ☐ Latency end-to-end ⩽90 s/study on 4060.

Kowshik (ML-B)

- ☐ `findings.schema.json` + validator pass on 10 cases.
- ☐ Clinician PDF (EN) & Patient PDFs (EN/HI) generated, consistent with JSON.
- ☐ FHIR DiagnosticReport & ImagingStudy JSONs saved; grounded Q&A returns evidence-linked answers.

Teja (Data/MLOps)

- ☐ DVC-tracked proto_pack (10–20 CTs); dataset card.
- ☐ FastAPI + queue resilient to kill/restart; metrics scripts produce CSVs.
- ☐ README with one-command run; environment pins.

Rahul (Frontend)

- ☐ Ingest → Analyze → Results works offline; overlays toggle smoothly.
- ☐ Bilingual toggle; all downloads work.
- ☐ Error states friendly; show queued FHIR JSONs.

Quick tips / pitfalls per role

- **Dinesh**: If UNet is slow, reduce cube size to 48 or stride to 2; keep batch small (fits 8 GB). Accept coarse masks for proto; quality later.
- **Kowshik**: Use a Devanagari font (e.g., Noto Sans Devanagari) so Hindi renders in PDFs. Keep Q&A strictly **schema-grounded** (no free-text LLM yet).
- **Teja**: Don't load entire volume into RAM for all steps; batch slices; log per-stage time. Persist job state to DB so resume works.
- **Rahul**: Precache overlays (PNG) so toggles are instant. Window/level on the client with numpy → PIL conversions.

If you want, I can also drop **starter files** (skeleton code for each of the modules above), plus a **sample findings.json**, **FHIR JSON**, and **report templates (EN/HI)** so you can kick off implementation today.