Got it Kowshik 👍 — let's lay everything out **clearly** for each member: **pipeline, architecture, deliverables, and sequence of execution**. This will give you and your teammates a single source of truth.

🏗️ Overall End-to-End Architecture (System View)

Flow:
DICOM ingest → Preprocessing → Detection → Segmentation → Risk classification → Explainability → Findings JSON → Reporting (Clinician + Patient PDFs) → FHIR Export → Frontend Viewer (UI).

Sequence:

1. **Teja (Data/MLOps)** prepares data + endpoints + queue.
2. **Dinesh (ML-A)** builds detection, segmentation, risk, Grad-CAM.
3. **Kowshik (ML-B)** builds schema, reports, FHIR, Q&A.
4. **Rahul (Frontend)** builds offline UI to tie everything together.

👨‍💼 Dinesh — ML-A (Vision Lead)

📌 Pipeline

1. **Preprocessing** → resample (1mm), HU clip, normalize, 2.5D slice packs.
2. **Detection** → LoG blob detector (v1) / CenterNet-lite (v2).
3. **Segmentation** → Mobile-UNet / Half-UNet (lightweight).
4. **Risk Head** → deep features + radiomics → MLP classifier.
5. **Explainability** → Grad-CAM++ overlays on nodules.
6. **Integration** → run_inference(study_dir) returns findings.json, masks, saliency.

📌 Architecture

- **Model stack**:
    - Detector (classical + small CNN head)
    - Segmenter (2.5D/3D UNet)
    - Risk classifier (MLP with features + radiomics)
    - Grad-CAM++ (XAI)
- **Frameworks**: PyTorch + MONAI, SimpleITK.
- **Inputs/Outputs**:
    - Input: DICOM → preprocessed volume.
    - Output: JSON + overlays.

📌 **Deliverables**

- ml/preprocess.py, ml/detector.py, ml/segmenter.py, ml/risk.py, ml/xai.py, ml/infer.py.
- findings.json with nodules + risk.
- Mask PNGs + saliency PNGs.
- Metrics: Dice ≥0.8, AUC ≥0.9, ≤60s/study latency.

📌 **Sequence**

- Week 1: Preprocessing + detector v1.
- Week 2: Train UNet seg baseline, generate masks.
- Week 3: Risk head + Grad-CAM overlays.
- Week 4: Integrate → run_inference() end-to-end.

👷 **Kowshik — ML-B (Reporting/NLP)**

📌 **Pipeline**

1. **Schema & Validation** → findings.schema.json + validator.
2. **Reporting** → template-based PDF reports: clinician (EN) + patient (EN/HI).
3. **FHIR Export** → DiagnosticReport + ImagingStudy JSONs.
4. **Q&A** → rule-based engine answering from findings.json.

📌 **Architecture**

- **Validator**: JSON schema + rules (size, lobe, Lung-RADS).
- **Reporter**: Markdown → PDF (WeasyPrint/ReportLab).
- **FHIR Exporter**: Structured JSON compliant with HL7 FHIR.
- **Q&A Engine**: JSON-grounded rule-based retrieval.
- **Inputs/Outputs**:
    - Input: findings.json (from Dinesh).
    - Output: clinician.pdf, patient_en.pdf, patient_hi.pdf, diagnostic_report.json.

📌 **Deliverables**

- app/validators.py, app/reporter.py, app/fhir_export.py, app/qa_engine.py.
- Templates: templates/clinician_en.md, patient_en.md, patient_hi.md.
- Sample outputs: PDFs + FHIR JSONs.
- Validator passes ≥10 cases, no hallucination in reports.

📌 **Sequence**

- Week 1: Schema draft + validator v1.
- Week 2: Report generator v1.
- Week 3: FHIR exporter + Q&A engine.
- Week 4: BLEU/ROUGE eval vs reference reports + polish.

👷 **Teja — Data & MLOps**

📌 **Pipeline**

1. **Data Handling** → DICOM loader, save overlays, save JSON.
2. **Data Versioning** → DVC pipeline for raw → processed CTs.
3. **Queue/Orchestration** → RQ/Redis or Celery to manage pipeline jobs.
4. **Evaluation Harness** → scripts for latency, FROC, Dice.

📌 **Architecture**

- **Backend**: FastAPI server + RQ/Celery workers.
- **DB**: Redis/SQLite for queue states.
- **Eval Scripts**:
  - latency_bench.py
  - mini_froc.py
  - dice_smallset.py
- **Inputs/Outputs**:
  - Input: DICOM studies (10–20 proto_pack).
  - Output: metrics CSVs + stored results in /artifacts/metrics/.

📌 **Deliverables**

- ml/io.py, data/proto_pack/, data/dataset_card.md.
- app/queue.py (job queue).
- Metrics: latency.csv, mini_froc.csv, dice.csv.
- README with one-command run.

📌 **Sequence**

- Week 1: Dataset prep + proto_pack.
- Week 2: Queue skeleton + endpoints (/ingest, /analyze, /status).

- Week 3: Eval scripts (latency, Dice).
- Week 4: Crash-safe job queue, reproducible runs.

👷 **Rahul — Frontend**

📌 **Pipeline**

1. **UI Framework** → Streamlit (offline-first).
2. **Pages**:
    - Home/Ingest → upload DICOM.
    - Analyze → POST to server, poll /status.
    - Results → axial viewer, overlay toggle, downloads.
3. **Features**: bilingual toggle, offline queue (outbox), error handling.

📌 **Architecture**

- **Frontend**: Streamlit UI with pages.
- **Viewer**: Slice slider, overlay toggles (mask/saliency).
- **Downloads**: PDFs, JSONs.
- **Inputs/Outputs**:
    - Input: study_uid (after ingest).
    - Output: Visual viewer + downloads.

📌 **Deliverables**

- ui/app.py with pages (Ingest, Analyze, Results).
- Responsive axial viewer (512×512).
- Toggle overlays + bilingual UI.
- Download buttons working.

📌 **Sequence**

- Week 1: Skeleton UI with dummy JSON.
- Week 2: Integrate viewer + overlay toggles.
- Week 3: Add downloads (PDFs, JSONs).
- Week 4: Offline queue + polish UX.

🗓️ **Sequential Execution (Team)**

1. **Teja (Week 1)** → prepare proto_pack + ingest/analyze endpoints.
2. **Dinesh (Week 1–2)** → preprocessing + baseline UNet segmentation.
3. **Kowshik (Week 1–2)** → schema + validator + draft templates.
4. **Rahul (Week 1)** → Streamlit skeleton with dummy JSON.
5. **Dinesh + Teja (Week 2–3)** → integrate detector + risk head.
6. **Kowshik (Week 3)** → FHIR export + Q&A.
7. **Rahul (Week 3)** → integrate downloads + bilingual toggle.
8. **Final Week** → integrate all modules → dry-run demo → freeze repo.

✅ This gives you **per-member pipeline, architecture, deliverables, and sequence**.
👉 Do you want me to now **generate the starter repo structure + code skeletons** (empty functions, file scaffolds) for all 4 roles so your team can directly start coding?