Perfect — let’s make a **one-day, knock-out hackathon plan**. I’ll give a tightly prioritized, task-by-task roadmap your 2-person team can follow, parallel work so nothing blocks, fallbacks if something fails, and what to show in the demo. This plan picks a realistic MVP of the system described in your PDF (certificate upload → OCR → immutable ledger/hash verification → dashboard). Reference: your project blueprint.

Summary MVP (what to finish today)

Minimal, demoable product that proves the idea and maps to the blueprint:
User can sign up, upload a certificate image/PDF → **OCR extracts key fields** → app computes a **certificate hash** and stores an **append-only ledger record** (demo of blockchain immutability) → dashboard shows certificate + verification status. Include a manual “mark verified by issuer” admin button to show verification flow. Keep Gmail scanning / Hyperledger / advanced forensic/ML as future features (callouts in slides).

Tech stack (fastest for 1 day)

- Frontend: React (Create React App or Vite). Use plain React components + Bootstrap/Tailwind CDN.
- Backend: Node.js + Express.
- DB: SQLite (file) or PostgreSQL if you already have it. SQLite easiest for quick deploy.
- File upload: multer (Express).
- OCR: tesseract.js (client-side) or server-side pytesseract if you prefer Python — **use tesseract.js in frontend** to avoid server work.
- “Blockchain” ledger: simple append-only table in DB storing `{certificate_id, hash, timestamp, prev_hash}` to demo immutability (Merkle-like / chained hash).
- Auth: simple email/password (JWT session) OR Firebase Auth if you want OAuth and fewer lines of auth code. For speed: email/password (no external OAuth).
- Hosting (demo): Frontend → Vercel / Netlify, Backend → Render / Railway / Heroku (free tier).

Roles

- **Person A (Frontend + OCR & demo UI)** — build upload UI, call OCR, show extracted fields, show ledger and verification status, implement visual demo flow.
- **Person B (Backend + DB + ledger + API + deploy)** — build Express API for saving cert metadata & ledger, compute and verify hashes, admin endpoints, deploy backend.

Work in parallel: frontend can use mocked API endpoints while backend is being built.

Hour-by-hour task plan (one full day)

(Follow in order, parallelize where noted. Replace "hours" with blocks — keep strict timeboxing.)

Morning — Setup & skeleton (2.0 hours)

- A1 (30m): Create Git repo, branches (`main`, `frontend`, `backend`). Add README and demo goals.
- A2 (30m): Person B — init Express app, setup SQLite, create DB schema: `users`, `certificates`, `ledger`. (Create SQL file.)
- A3 (30m): Person A — init React app, basic layout: login/signup, dashboard, upload page.
- A4 (30m): Agree API contract (endpoints, JSON shapes) so components can work in parallel. (E.g. `POST /api/certs/upload`, `GET /api/certs/:id`, `POST /api/ledger`.)

Late morning — Core flows parallel (3.0 hours)

- B1 (90m): Implement backend endpoints:
  - `POST /api/auth/signup`, `POST /api/auth/login` (simple JWT).
  - `POST /api/certs` to accept metadata (filename, fields, computed hash).
  - `GET /api/certs` and `GET /api/certs/:id`.
  - `POST /api/ledger` to append a record (compute `record_hash = SHA256(prev_hash + cert_hash + timestamp)`), store `prev_hash`.
  - `POST /api/certs/:id/verify` admin toggle.
- A1 (90m): Implement frontend upload:
  - File input & preview.
  - Integrate **tesseract.js** to OCR file client-side; extract candidate fields (name, issuer, issue date). Provide editable fields if OCR misreads (manual override).
  - Compute cert hash on frontend (SHA256 of file bytes or extracted data) OR send file to backend and let backend hash — either is fine; prefer backend hashing for consistency.
  - Call backend to save cert metadata (mock while backend not ready).
- Parallel: set up sample test certificates (2–3 images / PDFs) to demo.

Early afternoon — Ledger + verification UI (1.5 hours)

- B2 (45m): Implement ledger append logic and an endpoint `GET /api/ledger/:certId` to retrieve chain for a cert.
- A2 (45m): Build UI to display the certificate card: extracted fields, certificate hash, ledger chain display (list entries with timestamps and hashes), verification badge (red/yellow/green).

Mid afternoon — Connect, polish, and fallback (1.5 hours)

- B3 (45m): Ensure endpoints are secure enough for demo: basic auth middleware, CORS, input validation. Add a simple admin route to mark a cert verified by issuer.
- A3 (45m): Add manual field editing fallback if OCR fails. Add progress indicators, success/error messages. Add a demo walkthrough panel explaining the steps.

Late afternoon — Testing, deployment, and demo prep (2.0 hours)

- B4 (60m): Deploy backend to Render/Heroku and test endpoints with Postman/curl. Seed DB with sample certs and ledger entries for demo.
- A4 (60m): Deploy frontend (Vercel/Netlify), point config to deployed API. Verify OCR still works in deployed site (tesseract.js should run in browser).

Evening — Final polish, slides, and backup plan (2.0 hours)

- Both (60m): Create a 4–6 slide deck:
    1. Problem + value proposition (1 slide).
    2. Architecture diagram (1 slide) — show that future Hyperledger and Gmail scanning map onto your MVP. Cite blueprint if needed.
    3. Demo flow + tech stack (1 slide).
    4. Future work & stretch goals (1 slide).
- Both (60m): Record a 2–3 minute demo video (screen record) showing upload → OCR → ledger append → verification toggle — this is your backup if live demo fails. Prepare short script.

Concrete API + DB schema (copyable)

DB tables

- `users(id, email, password_hash, name)`
- `certificates(id, user_id, issuer, holder_name, issue_date, raw_text, file_path, cert_hash, created_at, verified_bool)`
- `ledger(id, certificate_id, record_hash, prev_hash, timestamp, details_json)`

Key endpoints

- `POST /api/auth/signup` — {email, password}
- `POST /api/auth/login` — returns JWT
- `POST /api/certs` — {issuer, holder_name, issue_date, raw_text, cert_hash} (file upload via multipart)
- `GET /api/certs`
- `GET /api/certs/:id`
- `POST /api/ledger` — {certificate_id} (backend calculates new record_hash)
- `POST /api/certs/:id/verify` — admin toggles verify.

Demo script (30–60s)

1. Login as demo user.
2. Upload certificate image → show OCR extracting fields (show manual correction).
3. Click “Save” → backend stores cert and computes cert hash; show certificate hash on card.
4. Click “Record on Ledger” → backend appends ledger (show chain with prev_hash).
5. Admin toggles “Verified by issuer” → verification badge turns green.
6. Optionally show raw ledger JSON to emphasize immutability.

Backup / shortcuts if time runs out

- Skip auth — use a shared demo account or no auth to save hours.
- Skip file uploads; instead use a form to paste certificate text and a sample image pre-loaded.
- If OCR integration is flaky, **mock OCR**: let user paste text and show it as “OCR result” (still demonstrates flow).
- If deployment fails, run local demo and use a quick screen recording as the submission.

What to prepare right now (immediately)

1. Create repo and branches.
2. Create 3 sample certificate images (one “good”, one skewed, one complex).
3. Person A: scaffold React and add tesseract.js demo page.
4. Person B: scaffold Express and create DB with the three tables.
5. Draft 4 slide deck skeleton.

Final tips & judging talking points

- Emphasize **immutability demo** (show hashes chained) — judges love a tangible proof of “blockchain-like” immutability even if not full Hyperledger.
- Make UX obvious: “Upload → OCR → Append to ledger → Verified” with clear colors.
- Mention security & privacy (minimal PII stored), and roadmap items (Hyperledger Fabric, Gmail aggregator, forensic analysis) so judges see scalability and technical depth.
