PHASE 0 — Project Idea Clarity

Target User

Primary target user for MVP:

> **Hardware / robotics / rocketry project teams (like IGNITION)**

This includes:

- Student engineering teams
- Robotics & electronics labs
- Experimental hardware projects
- Simulation + testing-heavy workflows

Final Core Idea (Crisp & Defensible)

> **LabLedger is a process-centric project and experiment management system designed for hardware and robotics teams, where learning happens through trials, failures, iterations, and observations — not just through final code.**

“Why not GitHub?”

Answer:

- GitHub tracks **files and code**
- LabLedger tracks **experiments, decisions, failures, and learning**

For teams like IGNITION:

- Many experiments don’t even produce code

- Value lies in *why something failed*, not just *what worked*
- Data is scattered across notebooks, WhatsApp, PDFs, and memory

👉 **LabLedger becomes the team's shared engineering memory.**

Key Differentiators (Contextual)

1️⃣ Experiment-first, not code-first

- Every task is an **experiment**
- Logs are structured: objective → procedure → observation → outcome
- Failures are explicitly recorded and preserved

This directly mirrors how you work in real hardware teams.

2️⃣ User-defined customization

Hardware teams need:

- Voltage
- Current
- Temperature
- RPM
- Firmware version
- Test conditions
- Environment notes

So:

- Users define **custom fields**

- Users define **experiment templates**
- One team ≠ another team

This alone makes the project **non-trivial and original**.

3️⃣ Append-only engineering logs

- You can't "rewrite history"
- Every iteration stays
- Clear traceability of decisions

This is **real engineering discipline**, not just CRUD.

4️⃣ Designed for intelligence later (ML-ready)

Not ML-heavy now — but **ML-meaningful later**.

Examples:

- "Which parameters most often cause failures?"
- "Summarize all propulsion tests"
- "What changed between successful and failed runs?"

These questions *naturally emerge* in teams like IGNITION.

Important Realization (This is BIG)

> **A software system that formalizes how engineering teams actually think and work**

PHASE 1 — SYSTEM DESIGN

Goal of Phase 1:
Convert the *idea* into a **clear, stable system blueprint** that won't change later.

1.1 Design Philosophy

Before diagrams, we freeze **how the system thinks**.

Core principles

- **Experiment-first system**
- **Append-only knowledge**
- **User-defined structure**
- **Failure is data**
- **Process > outcome**

1.2 Actors (WHO uses the system)

Actor	Description
Admin	Manages users, roles, global settings
Engineer (Contributor)	Logs experiments, creates projects
Reviewer / Mentor	Views logs, adds comments, feedback

1.3 Core Entities (REFINED & EXTENDED)

1️⃣ Team (Top-level organization)

Represents the **entire organization**.

Example: IGNITION

Fields

- team_id
- name
- domain (rocketry / robotics / electronics)
- description
- created_at

👉 A Team can have **multiple SubTeams** and **multiple Projects**.

2️⃣ SubTeam (Department / Division)

Optional but powerful.

Examples

- Payload
- Propulsion
- Avionics
- Structures

Fields

- subteam_id
- name
- description
- parent_team_id

🔹 A SubTeam:

- Belongs to **one Team**
- Has **its own members**

- Can create **independent projects**
- Still stays visible to the parent Team

This matches real-world department autonomy.

3️⃣ User (Member)

Unchanged conceptually, but role handling improves.

Fields

- user_id
- name
- email
- team_id
- subteam_id (nullable)

A user may:

- Belong to only Team
- Or Team + one SubTeam

4️⃣ Project

Projects can exist at **two levels**.

Project Types

1. **Team-level Project**
    - Example: *IgniteX Rocket*
    - Involves all subteams
2. **SubTeam-level Project**

- Example: *Hybrid Engine Development*
- Owned by Propulsion subteam

Fields

- project_id
- name
- description
- project_type (TEAM / SUBTEAM)
- team_id
- subteam_id (nullable)
- project_lead_id
- status

5️⃣ Project Lead

This is a **role**, not an entity.

Responsibilities

- Owns the project
- Approves access
- Assigns project-specific roles
- Controls visibility (read/write)

Think of it like:

> **Project-level IAM admin**

6️⃣ Module / Subsystem

Applies to both project types.

Examples

- Combustion
- Injector Design
- Control Logic
- Testing

Fields

- module_id
- name
- project_id

7️⃣ Experiment

This is the heart of LabLedger.

- experiment_id
- title
- objective
- hypothesis (optional)
- module_id
- created_by
- created_at

> Experiments are **intentional actions**, not tasks.

8️⃣ Experiment Logs (Append-only)

Each experiment has **multiple logs**.

- log_id
- experiment_id
- timestamp
- procedure
- observations
- outcome
- learnings
- status (success / fail / partial)

❗ Logs can never be edited — only appended.

9️⃣ Parameters & Attachments

This is where your differentiation shines.

- parameter_id
- experiment_id
- key (Voltage, Temp, RPM)
- value
- unit
- user_defined

This allows **domain-specific logging**.

8️⃣ Attachments

- file_id
- linked_to (log_id)

- file_type
- storage_url

1.4 Role & Permission Model

We introduce **hierarchical + customizable roles** (AWS-style).

Role Levels

🔹 **Team-level Roles**

- **Team Lead**
    - Overall owner of the team
    - Creates subteams
    - Assigns subteam leads
- **Team Member**
    - Default access (read-only unless granted)

🔹 **SubTeam-level Roles**

- **SubTeam Lead**
    - Owns subteam projects
    - Assigns project leads
- **SubTeam Member**
    - Works on subteam projects

🔹 **Project-level Roles (CUSTOMIZABLE)**

Defined by Project Lead.

Examples:

- Project Lead
- Contributor
- Reviewer
- Viewer

Roles are:

- **Project-scoped**
- **User-defined**
- **Permission-mapped**

Example permissions:

- Create experiment
- Add logs
- Upload files
- Comment
- View only

1.5 Access Control Scenarios

Let’s test the design with real cases 👇

Case 1 — Full Rocket Project

- Project: *IgniteX*
- Type: TEAM project
- All subteams involved

- Team Lead assigns Project Lead
- Each subteam contributes via modules

✔ Everyone collaborates
✔ Clear ownership
✔ Central visibility

Case 2 — Propulsion Engine Development

- Project: *Hybrid Engine v1*
- Type: SUBTEAM project
- Owned by Propulsion subteam

Access:

- Propulsion members → full access
- Other subteams → read-only (on approval)

✔ Isolation + transparency
✔ No clutter
✔ Knowledge sharing without interference

1.6 Updated System Flow (High-level)

```
User
 ↓
Team
 ↓
(Optional) SubTeam
 ↓
Project (Team-level or SubTeam-level)
 ↓
Modules / Subsystems
 ↓
Experiments
 ↓
```

```
Logs (Append-only)
```

This flow is **clean, scalable, and realistic**.

PERMISSIONS & ACCESS CONTROL DESIGN

1️⃣ Design Philosophy

Before tables, freeze the **rules of power**:

1. **Permissions are hierarchical**
2. **Ownership > Membership**
3. **Project autonomy is respected**
4. **Visibility ≠ Editability**
5. **Roles are customizable, permissions are atomic**

Think **AWS IAM + GitHub + Lab workflow**, simplified.

2️⃣ Levels of Control (Hierarchy)

Permissions exist at **three levels**:

```
TEAM LEVEL
    ↓
SUBTEAM LEVEL
    ↓
PROJECT LEVEL
```

Lower levels **cannot override higher-level hard blocks**,
but higher levels **should not micromanage projects**.

3️⃣ Role Types

A. Team-Level Roles

Applies across the entire organization.

Role	Description
Team Lead	Supreme authority for the team
Team Member	Default member

B. SubTeam-Level Roles

Applies only inside a department.

Role	Description
SubTeam Lead	Owns department projects
SubTeam Member	Works in that department

C. Project-Level Roles (Customizable)

Defined **per project** by Project Lead.

Examples (not fixed):

- Project Lead
- Contributor

- Reviewer
- Viewer
- Analyst
- Observer

👉 These are **labels**, permissions are what matter.

4️⃣ Atomic Permissions (The Real Control Units)

These are the **smallest permission blocks**.

Core Permission Set

Project Management

- CREATE_PROJECT
- DELETE_PROJECT
- ASSIGN_PROJECT_ROLES
- APPROVE_ACCESS

Experiment Control

- CREATE_EXPERIMENT
- EDIT_EXPERIMENT_META
- ADD_EXPERIMENT_LOG
- COMMENT_ON_LOG

Log & Data Control

- UPLOAD_FILES
- VIEW_LOGS

- EXPORT_DATA

Structure Control

- CREATE_MODULE
- MODIFY_MODULE

5️⃣ Permission Matrix (CRITICAL SECTION)

🟦 Team-Level Permissions

Action	Team Lead	Team Member
Create SubTeams	✅	❌
Assign SubTeam Leads	✅	❌
View all projects	✅	✅
Override access (emergency)	✅	❌

🟩 SubTeam-Level Permissions

Action	SubTeam Lead	SubTeam Member
Create SubTeam projects	✅	❌
Assign Project Leads	✅	❌

Full access to subteam projects	✅	✅
Approve read-only access	✅	❌

🟨 Project-Level Permissions (Example Default)

Action	Project Lead	Contributor	Viewer
Create experiments	✅	✅	❌
Add logs	✅	✅	❌
Upload files	✅	✅	❌
Comment	✅	✅	❌
View logs	✅	✅	✅
Assign roles	✅	❌	❌

⚠️ Project Lead can **customize this table per project**.

6️⃣ Access Scenarios (Your Real Use Cases)

Scenario A — Full Rocket Project (TEAM Project)

- All subteams involved

- Team Lead → assigns Project Lead
- SubTeam Leads manage their modules
- Members contribute via experiments

✔ Collaborative
✔ Structured
✔ No chaos

Scenario B — Propulsion Engine R&D (SUBTEAM Project)

- Owned by Propulsion
- Only propulsion members have write access
- Others can request **read-only**

Approval flow:

```
Request → Project Lead → Approved → View-only
```

✔ Isolation + transparency
✔ Knowledge sharing without interference

7️⃣ Conflict Resolution Rules

These rules prevent ambiguity:

1. **Project Lead > SubTeam Lead > Team Member**
2. **Higher scope cannot silently edit lower scope**
3. **Read-only never implies comment**
4. **No one edits logs once submitted**
5. **Team Lead override is logged & auditable**

Mentioning *auditable overrides* makes this look very professional.

8️⃣ Why This Permission Model Is STRONG

From an evaluator’s view, you’ve shown:

- Multi-level RBAC
- Custom role definition
- Least-privilege principle
- Real organizational modeling
- Inspiration from industry IAM systems

This is **far beyond CRUD authentication**.

DIAGRAMS:

1️⃣ ER DIAGRAM (Entity Relationship Diagram)

Purpose

To model **data structure, ownership, and relationships** in LabLedger.

This diagram answers:

> *What data exists and how is it connected?*

Entities to Include (FINAL & FROZEN)

Core entities

- User
- Team
- SubTeam
- Project
- Module
- Experiment
- ExperimentLog
- Parameter
- Attachment
- Role
- Permission

Relationships (VERY IMPORTANT)

Team ↔ SubTeam

- One **Team** has many **SubTeams**
- Each **SubTeam** belongs to exactly one **Team**

Relationship: `Team 1 – N SubTeam`

Team ↔ User

- One Team has many Users
- A User belongs to one Team

Relationship: `Team 1 – N User`

SubTeam ↔ User

- One SubTeam has many Users
- User may or may not belong to a SubTeam

Relationship: `SubTeam 1 – N User (optional)`

Team/SubTeam ↔ Project

- Team-level projects: linked to Team
- SubTeam-level projects: linked to SubTeam

Relationship:

- `Team 1 – N Project`
- `SubTeam 1 – N Project (optional)`

Project ↔ Module

- A Project has multiple Modules

Relationship: `Project 1 – N Module`

Module ↔ Experiment

- Each Module contains many Experiments

Relationship: `Module 1 – N Experiment`

Experiment ↔ ExperimentLog

- Each Experiment has multiple Logs
- Logs are **append-only**

Relationship: `Experiment 1 – N ExperimentLog`

ExperimentLog ↔ Parameter

- Each log has multiple parameters

Relationship: `ExperimentLog 1 – N Parameter`

ExperimentLog ↔ Attachment

- Each log can have multiple attachments

Relationship: `ExperimentLog 1 – N Attachment`

User ↔ Role ↔ Project

- Users are assigned Roles **per project**
- Roles map to Permissions

This is a **many-to-many** relationship:

- User ↔ Project via Role

ER Diagram Justification

- Clear separation of **organizational structure** (Team, SubTeam)
- Project-level isolation ensures scalability
- Append-only logs enforce auditability
- Role–Permission abstraction supports AWS-style IAM
- Schema supports both hardware and software workflows

---

2 USE CASE DIAGRAM

Purpose

To show **who can do what** in the system.

This diagram answers:

> *Which actors interact with the system and how?*

Actors to Include

- Team Lead

- SubTeam Lead
- Project Lead
- Team/SubTeam Member
- Reviewer (read-only)

Core Use Cases (Group them logically)

🔹 Team Management

- Create Team
- Create SubTeam
- Assign SubTeam Lead
- Add Members

Actor: Team Lead

🔹 Project Management

- Create Project
- Assign Project Lead
- Approve Access Requests
- Define Project Roles

Actor: Team Lead / SubTeam Lead / Project Lead

🔹 Experiment Management

- Create Experiment

- Define Parameters
- Add Experiment Log
- Upload Attachments

Actor: Contributor / Project Lead

🔹 Review & Learning

- View Logs
- Comment on Logs
- Export Experiment Data

Actor: Reviewer / Viewer

Associations to Draw

- Team Lead → all management use cases
- SubTeam Lead → subteam project use cases
- Project Lead → project-specific role & access control
- Members → experiment & logging use cases
- Reviewer → view-only use cases

Use Case Diagram Justification

- Clear role separation avoids privilege confusion
- Supports least-privilege principle
- Reflects real engineering hierarchy

- Aligns with permission matrix previously defined

---

3 ACTIVITY DIAGRAMS

You should include **at least 2 activity diagrams**.

Activity Diagram 1: Experiment Logging Workflow

Flow (step-by-step)

1. Start
2. User logs in
3. Select Project
4. Select Module
5. Create / Select Experiment
6. Enter Procedure
7. Enter Observations
8. Add Parameters
9. Upload Files
10. Submit Log
11. System locks log
12. End

Control Flow Notes

- **Decision node:** “Is user authorized?”

- **No loop back** after submission (immutability)
- Clear linear progression

Justification

- Models real lab workflow
- Ensures accountability
- Prevents data tampering
- Supports audit trail

Activity Diagram 2: Access Request & Approval Flow

Flow

1. Start
2. User requests access
3. System notifies Project Lead
4. Decision:
    - Approve → Grant Read-only
    - Reject → Notify user
5. End

Fork / Join (Optional)

- Notification + logging can happen in parallel

Justification

- Explicit approval prevents unauthorized edits
- Supports transparency across departments
- Models real-world engineering governance

TECH STACK

Design Principle for Stack Selection

> **The stack must serve the system design, not the other way around.**

Your system needs:

- Strong RBAC (permissions)
- Clean REST APIs
- Structured + flexible data
- Cloud deployment
- Future ML integration
- Easy demo & grading

✅ PRIMARY STACK

🧠 Backend

Node.js + Express.js

Why

- Excellent for REST APIs
- Huge ecosystem for auth & RBAC
- Easy JWT handling
- Easy ML add-ons later (Python microservice can plug in)
- Fast to develop

Key libraries

- express

- jsonwebtoken (JWT)
- bcrypt (password hashing)
- multer (file uploads)
- cors
- dotenv

🗄️ Database

PostgreSQL

Why

- Strong relational modeling (perfect for Team → SubTeam → Project hierarchy)
- ACID compliance (important for logs)
- JSONB support → perfect for **user-defined parameters**
- Industry-standard, prof-approved

Key design choice

- Use relational tables for core entities
- Use `JSONB` column for:
    - Custom parameters
    - Role permission mappings

This is a *very smart hybrid approach*.

🌐 Frontend

React (basic, not fancy)

Why

- Component-based → role-based UI rendering
- Industry relevance
- Easy to keep simple
- Prof won’t penalize minimal UI

UI focus

- Functional, not aesthetic
- Role-aware pages
- Forms for experiments & logs

You can even say:

> “UI is intentionally minimal; focus is system design.”

🔐 Authentication & Authorization

JWT + Role-Based Access Control

Why

- Stateless
- Scales well
- Clean permission checks at API layer

Flow

1. Login
2. JWT issued
3. Role + permissions encoded
4. Backend validates every request

📁 File Storage

Cloud Object Storage

Choose ONE:

- AWS S3 (best industry value)
- Supabase Storage (simpler)
- Firebase Storage (easy)

Why

- Attachments don’t belong in DB
- Scales naturally
- Clean separation of concerns

☁️ Deployment

Recommended combo (simple + powerful)

Component	Platform
Backend	Render / Railway
Frontend	Vercel
Database	Supabase / Railway PostgreSQL
Storage	Same provider or AWS

This gives you:

- Public URLs
- Environment variables
- Logs
- Easy redeploy

🧩 Architecture Diagram (Textual)

```
React Frontend
   ↓ HTTPS
Express Backend (RBAC + APIs)
   ↓
PostgreSQL (Core Data + JSONB)
   ↓
Cloud Storage (Files)
```

Simple, clean, defensible.

🔮 ML EXTENSION STRATEGY (IMPORTANT)

You do **not** add ML now.

But your stack allows this later:

Option A — Python Microservice

- Flask / FastAPI
- Reads logs from DB
- Generates summaries / insights

- Exposed via REST API

Option B — Rule-based Engine (MVP)

- Detect missing logs
- Flag repeated failures
- Simple analytics

You can proudly say:

> "ML hooks are architecturally supported but out of MVP scope."

That's a **mature engineering statement**.

⚖️ ALTERNATIVE STACKS

Option 2 — Java Stack (If prof prefers)

- Backend: Spring Boot
- DB: PostgreSQL
- Auth: Spring Security + JWT

More boilerplate, but very academic-friendly.

Option 3 — Python Stack

- Backend: Flask / FastAPI
- DB: PostgreSQL
- Frontend: Simple React or Jinja

Fast to build, slightly less enterprise-looking.

🚫 What We Are NOT Using (On Purpose)

❌ Microservices
❌ Kubernetes
❌ GraphQL
❌ MongoDB-only design
❌ Serverless overkill

Avoids red flags.

LabLedger — FINAL MVP FEATURES & CUTS

🎯 MVP OBJECTIVE (LOCKED)

> Build a cloud-deployed, experiment-first project management system for hardware teams, where **engineering decisions, failures, and iterations are preserved through append-only logs with role-based access control**.

✅ FINAL MVP FEATURES (MUST WORK END-TO-END)

1️⃣ Authentication & Access

- Login with email + password
- JWT-based authentication
- **Seeded users** (no invite/signup flow):
    - Team Lead
    - Contributor
    - Viewer
- Backend-enforced authorization (403 Forbidden on violation)

2️⃣ Team & SubTeam Structure

- Single Team (seeded)
- SubTeams:
    - Create SubTeam
    - Assign users to SubTeam
- SubTeam exists mainly for **project ownership context**

❌ No multi-team support in MVP

3️⃣ Project Dashboard (Context Dashboard)

- Landing page after login
- Lists **projects the user can access**
- Shows:
    - Project name
    - Project type (TEAM / SUBTEAM)
    - User role in project
- Button: **Open Project**

❌ No analytics
❌ No metrics
❌ No activity feed

4️⃣ Projects

- Create Project
- Project types:
    - TEAM project
    - SUBTEAM project
- Assign Project Lead (seeded logic)
- Project visibility controlled by role

5️⃣ Breadcrumb Navigation (MANDATORY)

Displayed on all pages below Project level.

Example:

`IGNITION > PROPULSION > HYBRID ENGINE > INJECTOR TESTS`

Purpose:

- Navigation clarity
- Demo safety
- Zero UX confusion

6️⃣ Modules / Subsystems

- Create modules inside a project
- Simple list view
- No hierarchy inside modules

Examples:

- Combustion
- Avionics
- Testing

7️⃣ Experiments

- Create experiment inside a module
- Required fields:
    - Title
    - Objective
- Linked to exactly one module

❌ No tags
❌ No hypothesis (optional, can exist but ignored)

8️⃣ Experiment Logs (CORE FEATURE — MUST BE PERFECT)

Each experiment supports **append-only logs**.

Log fields:

- Procedure
- Observations
- Outcome
- Parameters (key–value list)
- Timestamp (auto-generated)

Rules (NON-NEGOTIABLE):

- ❌ No edit log endpoint
- ❌ No delete log endpoint
- ✅ Logs are append-only
- ✅ Immutability enforced at API layer

This is the heart of LabLedger.

9️⃣ Parameters (SAFE IMPLEMENTATION)

- UI uses:
    - Key input
    - Value input
    - “Add Parameter” button
- Backend converts list → JSONB

- Stored per log

❌ No raw JSON input
❌ No validation rules
❌ No templates UI

🔟 File Uploads (Evidence-Based Logging)

- Upload **one file per log**
- Any file type
- Stored in cloud object storage
- Only file URL saved in DB

❌ No preview
❌ No delete
❌ No versioning

1️⃣1️⃣ Role-Based Access Control (RBAC)

Hardcoded roles (no UI):

Role	Permissions
Project Lead	Full access
Contributor	Create experiments, add logs
Viewer	Read-only

Enforced in backend middleware
Frontend may show disabled buttons, but backend is source of truth.

1️⃣2️⃣ Authorization Error Visibility (DEMO-CRITICAL)

- Unauthorized actions return **403 Forbidden**
- Error shown via:
    - Toast notification OR
    - Network tab (demo)

This proves backend security, not UI hiding.

1️⃣3️⃣ Cloud Deployment

- Backend deployed (Render / Railway)
- Frontend deployed (Vercel)
- PostgreSQL (cloud)
- Object storage for files

Public URL must work end-to-end.

❌ FINAL CUTS (NOT IN MVP — DO NOT BUILD)

These features **do not exist at all** in MVP:

- ❌ Custom role creation UI
- ❌ Permission editor
- ❌ Notifications
- ❌ Activity feed
- ❌ Analytics / charts
- ❌ Search

- ❌ Experiment templates UI
- ❌ CSV / export
- ❌ Comments on logs
- ❌ Multi-team support
- ❌ User invite system
- ❌ Real-time features
- ❌ Audit dashboards

If it’s not listed under MVP features — it doesn’t exist.

🧊 FROZEN FEATURES (EXIST BUT MINIMAL)

These exist only to support the core flow:

Feature	MVP State
Roles	Seeded only
Permissions	Hardcoded
Parameters	Key–value only
SubTeams	Flat, no hierarchy
UI Styling	Minimal, ugly but clear
ML	Mentioned in README only

🧠 README / DEFENSE STATEMENTS (IMPORTANT)

You explicitly state:

> “LabLedger is an experiment-first system.
> Logs are append-only by design to preserve engineering truth.
> Advanced features (ML, analytics, notifications) are architecturally supported but intentionally excluded from MVP to maintain scope discipline.”

This turns cuts into **design decisions**.

✅ FINAL MVP SUCCESS CRITERIA

Your MVP is **DONE** if you can:

- Login
- Open dashboard
- Open project
- Create experiment
- Add append-only log
- Upload evidence
- Login as Viewer
- Attempt edit → blocked with 403
- Show breadcrumb navigation
- Access public deployed URL

If all of that works → **full marks**.

IMPLEMENTATION ROADMAP — LabLedger (MVP)

CORE PRINCIPLE FOR EXECUTION

> **Backend first → API stable → Frontend thin → Deploy early**

If backend is solid, everything else is easy.

🗓️ OVERALL PHASE SPLIT

Phase	Goal	Outcome
Phase A	Foundation	Auth + DB ready
Phase B	Core Data Model	Projects → Experiments
Phase C	Logging Engine	Append-only logs
Phase D	File Uploads	Evidence attached
Phase E	Permissions	RBAC enforced
Phase F	Frontend	Minimal but complete

Phase G	Deployment	Public working app
Phase H	Polish & Docs	Submission-ready

🔹 PHASE A — FOUNDATION (Day 1–2)

A1. Repo & Project Setup

- Create GitHub repo
- Setup:
    - `backend/`
    - `frontend/`
- Setup `.env` handling

Checkpoint

- Repo exists
- Server starts without errors

A2. Backend Skeleton

- Node.js + Express
- Base folders:
    - routes
    - controllers

    - services
    - middleware
    - models

Checkpoint

- `/health` endpoint returns OK

A3. Database Setup

- PostgreSQL (cloud or local)
- Tables (initial):
    - users
    - teams
    - subteams

No relations yet — just schema.

Checkpoint

- DB connected
- Can insert & fetch users

🔹 PHASE B — AUTH + ORG STRUCTURE (Day 3–4)

B1. Authentication

- Register
- Login
- Password hashing

- JWT issuance

Checkpoint

- Login returns JWT
- Protected route rejects unauth users

B2. Team & SubTeam Logic

- Create Team (once)
- Create SubTeams
- Assign users to SubTeams

MVP simplification

- One team only
- Team Lead created manually (seed data)

Checkpoint

- Users linked to subteams
- Data visible via API

🔷 PHASE C — PROJECT & MODULES (Day 5–6)

C1. Project Entity

- Create Project
- Project type:
    - TEAM

    - SUBTEAM
- Assign Project Lead

Checkpoint

- Project created with correct ownership

C2. Modules / Subsystems

- Create modules per project
- Simple CRUD

Checkpoint

- Project → modules relationship works

🔹 PHASE D — EXPERIMENT ENGINE (Day 7–8)

⚠️ This is the **heart of the system**.

D1. Experiments

- Create experiment
- Link to module
- Objective mandatory

Checkpoint

- Experiments list under modules

D2. Append-Only Logs (CRITICAL)

- Add experiment log
- Fields:
    - procedure
    - observations
    - outcome
    - parameters (JSON)
- Auto timestamp
- No edit / delete endpoints

Checkpoint

- Logs append correctly
- Older logs untouched

This alone makes your project strong.

🔹 PHASE E — FILE UPLOADS (Day 9)

E1. Cloud Storage Setup

- AWS S3 / Supabase / Firebase
- Secure credentials via env

E2. Upload per Log

- Upload file
- Store:
    - file name

- type
- URL
- log_id

Checkpoint

- File uploaded
- URL accessible
- Linked to correct log

🔹 PHASE F — PERMISSIONS (Day 10)

F1. Backend RBAC Middleware

- Check role from JWT
- Enforce:
  - who can create projects
  - who can log experiments
  - who is read-only

Hardcode permissions (no UI).

Checkpoint

- Unauthorized actions blocked
- Viewer cannot modify anything

🔹 PHASE G — FRONTEND (Day 11–13)

⚠️ Keep UI **ugly but functional**.

G1. Auth Pages

- Login
- Register

G2. Dashboard

- List projects user can see
- Click → project details

G3. Experiment Flow UI

- Select module
- Create experiment
- Add log
- Upload file

Forms only. No fancy UI.

Checkpoint

- Full workflow doable from browser

🔹 PHASE H — DEPLOYMENT (Day 14)

H1. Backend Deployment

- Render / Railway
- Env vars set
- DB connected

H2. Frontend Deployment

- Vercel
- Backend URL wired

Checkpoint

- Public URL works
- Prof can open & test

🔹 PHASE I — POLISH & DOCUMENTATION (Day 15)

Final Touches

- Seed demo data
- Add README
- Screenshots
- Architecture explanation
- Known limitations
- Future scope

✅ FINAL SUBMISSION CHECKLIST

You must be able to show:

- Live URL
- Login works
- Create project

- Log experiment
- Upload evidence
- Read-only access enforced
- Logs immutable

If yes → **project is DONE**.