AIRMAN Full Stack Developer Technical Assessment

72-Hour Technical Assessment

Objective

- Ship **end-to-end features** (UI + API + DB)
- Write **clean, scalable architecture**
- Implement **security fundamentals** (Auth, RBAC, Multi-tenant boundaries)
- Demonstrate **performance awareness** (pagination, caching, indexing)
- Show **CI/CD maturity & deployment discipline**
- Exhibit **product thinking & time prioritization**

Submission Requirements (Mandatory)

All submissions must include:

Public GitHub Repository

- Clean commit history
- Proper folder structure
- No broken builds
- Repository must be **PUBLIC**

The GitHub repo link must be shared in the provided Google Form.

Documentation Files

README.md must include:

- Setup steps
- Architecture diagram (simple is fine)
- Key technical decisions & tradeoffs
- API documentation
- Sample requests
- Demo credentials

PLAN.md must include:

- 72-hour schedule breakdown
- What was shipped
- What was intentionally cut
- Why certain features were deprioritized

CUTS.md

- List of features intentionally not built
- Clear reasoning behind each decision

POSTMORTEM.md

- What went wrong
- Technical challenges faced
- What would be improved with one more week

Running Demo

- Docker Compose must run the entire stack using **one command**
- Optional: Cloud deployment link
- Short demo video (≤ 6 minutes) covering:
  - Key flows
  - RBAC enforcement
  - Tests running
  - CI passing

LEVEL 1

"Maverick + Skynet Core: Auth, Learning, Scheduling"

Problem Statement

Build a minimal, production-minded **AIRMAN Core** system with:

- Authentication + RBAC

- Learning module (Maverick-style)
- Scheduling module (Skynet-style)
- Clean UI + API + DB integration

A) Authentication & RBAC

Roles Required

- Student
- Instructor
- Admin

Permissions

Admin

- Create instructors
- Approve students

Instructor

- Create content
- Assign quizzes

Student

- View content
- Attempt quizzes

Mandatory Requirements

- Hashed passwords (bcrypt or argon2)
- Refresh tokens or session-based auth strategy
- Route guards on frontend
- RBAC enforced at backend API level

B) Learning Module (Maverick-lite)

Structure

- Course → Module → Lesson hierarchy
- Lesson types:
  - Text lesson
  - MCQ quiz

Quiz Flow

- Store attempts
- Calculate score
- Show incorrect questions

Mandatory Requirements

- Pagination for content lists
- Search by course/module title

C) Scheduling Module (Skynet-lite)

Features

- Instructor availability management
- Student booking requests
- Admin approval and instructor assignment
- Weekly calendar list view

Booking Logic

- Conflict detection (no double-booking instructor)
- Status states:
  - requested
  - approved
  - completed
  - cancelled

Mandatory Tech Requirements (Level 1)

Backend

- REST API

  (Node + Express / NestJS OR Python + FastAPI)

- PostgreSQL (Required)
- ORM preferred (Prisma / TypeORM / Sequelize / SQLAlchemy)
- Input validation (Zod / Joi / Pydantic)
- Structured error handling

Frontend

- React / Next.js (preferred) or equivalent
- Form validation
- Clean UX
- Role-aware navigation

DevOps

- Docker Compose:
  - frontend
  - backend
  - postgres
- CI pipeline (GitHub Actions or equivalent) must run:
  - lint
  - unit tests
  - build

Testing

Minimum:

- Backend unit tests for:
  - Auth
  - Booking conflict detection
- 2 integration tests hitting real DB

(Test containers acceptable)

Level 1 Acceptance Criteria

Must pass:

- RBAC enforced at API level
- Booking conflict detection works
- Quiz scoring works
- Docker Compose runs on a clean machine
- CI passes
- README is complete and clear

LEVEL 2

“Skynet Multi-Tenant Ops + Audit-Grade Workflow”

Extends Level 1 into a production-grade, institutional system.

A) Multi-Tenancy (Hard Requirement)

Implement **two flight schools as tenants**.

Choose one approach (must document decision):

- Shared DB + tenant_id on every row (recommended)
- Separate schema per tenant
- Separate DB per tenant

Mandatory Enforcement

- All queries scoped to tenant_id
- School A cannot access School B data
- Backend rejects cross-tenant access (even if frontend tampered)

B) Audit Logs (Aviation-Grade Discipline)

Log all critical actions:

- User login
- Course creation/update
- Schedule creation/approval/cancel
- Role changes

Log Must Include

- user_id
- tenant_id
- before/after state (JSON diff acceptable)
- timestamp
- correlation_id (request trace ID)

C) Workflow Engine

Implement scheduling workflow:

requested → approved → assigned → completed

Automation Required

- If instructor not assigned within X hours → escalate to Admin
- Email notification stub (console logger acceptable)

Technical Requirements

- Background job runner (BullMQ / Agenda / cron / etc.)
- Safe retry handling

D) Performance & Scalability

Mandatory:

- Caching for read-heavy endpoints

  (Redis preferred or in-memory with TTL)

- DB indexes (document which and why)
- Pagination on all list endpoints
- Rate limiting on:
  - Auth endpoints
  - Booking endpoints

E) Deployment & Release Discipline

Provide a cloud deployment option:

Examples:

- Render
- Fly.io
- Vercel + Railway
- Supabase
- AWS / GCP

Must Include

- Environment separation (dev / staging / prod)
- Secrets management approach
- Basic rollback strategy (documented)

F) CI/CD & Quality Gates

CI must include:

- Lint
- Unit tests
- Integration test with DB container
- Migration check
- Build artifacts for frontend + backend

At least one quality gate:

- Fail if test coverage drops below threshold

  OR

- Fail if performance metric below threshold

Keep it practical.

Level 2 Acceptance Criteria

- Tenant isolation verified with test cases
- Audit logs contain correct metadata
- Background jobs retry safely
- Rate limiting prevents abuse
- Cloud deployment works (or documented proof)
- CI blocks regressions

Bonus (High Signal — Optional)

Choose one:

- Offline-first quiz attempts (sync later)
- Telemetry ingestion stub (accept JSON flight event logs)
- Role-based feature flags

Approach & Time Management (Mandatory)

Candidate must submit:

- PLAN.md (time blocks & prioritization)
- CUTS.md (what was intentionally not built)
- POSTMORTEM.md (reflection + improvements)