CARMS PROJECT MASTERY PLAN (INTERVIEW-FOCUSED)

This guide is designed to help you build **deep, end-to-end ownership** of this project across architecture, data, code, runtime behavior, and product/DS extension ideas.

GOAL AND OUTCOMES

By the end of this plan, you should be able to confidently explain:

1. **Why the platform exists** and which user problems it solves.
2. **How data flows** from raw files into API-ready gold tables.
3. **How one API call is served** from route -> query -> schema -> response.
4. **How quality, performance, and security are enforced** in practice.
5. **What you would improve in your first 30 days** as a junior DS/DE.

Target total time: **6?9 hours**.

PHASE A ? CONCEPTUAL UNDERSTANDING (1?2 HOURS)

STEP A1) READ README.MD END-TO-END

Focus on:
? Platform intent and scope.
? Two run modes (UI-only vs full platform).
? Architecture diagram and endpoint surface.
? Security and rate limiting behaviors.

Output artifact: a 1-page note titled ?System in 90 seconds? answering:
? What the system does.
? Why it uses bronze/silver/gold.
? Why Dagster + FastAPI + Postgres were selected.

STEP A2) READ DOCS IN THIS EXACT ORDER

1. ‘docs/architecture.md‘
2. ‘docs/data-dictionary.md‘
3. ‘docs/api-contract.md‘
4. ‘docs/performance.md‘

As you read, maintain a two-column table:
? **Column A: claim** (e.g., ?silver standardizes and validates rows?).

? **Column B: where it is implemented** (file path / function / model).

STEP A3) WRITE YOUR OWN 8?10 ARCHITECTURE BULLETS

Use your own wording. Keep each bullet concise and testable.

Suggested structure for your bullets
? 1 bullet: business objective.
? 2 bullets: ingestion and transformation.
? 2 bullets: serving layer and API.
? 1 bullet: security/rate limit.
? 1 bullet: orchestration/operations.
? 1 bullet: performance strategy.
? 1 bullet: roadmap/future capability.

Interview check: if you cannot explain each bullet without notes, revisit docs.

PHASE B ? CODE TRACING (2?3 HOURS)

STEP B1) START FROM ‘CARMS/PIPELINES/DEFINITIONS.PY‘

Understand the asset/check registration and job wiring:
? Which asset groups are loaded.
? What checks are attached.
? How materialization order is implied through dependencies.

STEP B2) TRACE ONE RECORD BRONZE -> SILVER -> GOLD

Pick one ‘program_stream_id‘ and follow it through:
? Bronze models/assets (‘carms/pipelines/bronze‘, ‘carms/models/bronze.py‘).
? Silver transforms (‘carms/pipelines/silver‘, ‘carms/models/silver.py‘).
? Gold curation (‘carms/pipelines/gold‘, ‘carms/models/gold.py‘).

Document for each stage:
? Input fields.
? Transformation/validation logic.
? Output fields.
? Potential failure or null/edge conditions.

STEP B3) TRACE ONE API ENDPOINT ROUTE -> QUERY/MODEL -> RESPONSE SCHEMA

Recommended endpoint: ‘GET /programs‘.

Trace across:
? Route/controller: ‘carms/api/routes/programs.py‘.
? Data access/session dependency: ‘carms/api/deps.py‘, ‘carms/core/database.py‘.
? SQLModel models: ‘carms/models/gold.py‘.
? Response contracts: ‘carms/api/schemas.py‘.

Then repeat quickly for:
? ‘GET /programs/{program_stream_id}‘
? ‘GET /disciplines‘
? ‘POST /pipeline/run‘

STEP B4) DRAW DATA FLOW ON PAPER (OR DIGITAL WHITEBOARD)

Include exactly three lanes:
1. **Source lane:** files in ‘data/‘.
2. **Transform lane:** Dagster assets/checks.
3. **Serve lane:** FastAPI routes + map/static artifacts.

Also annotate:
? Where schema migrations apply.
? Where indexes matter.
? Where auth/rate limiting intercept requests.

PHASE C ? RUNTIME UNDERSTANDING (1?2 HOURS)

STEP C1) RUN TESTS AND INSPECT FAILURE BEHAVIOR BY BREAKING ONE TRANSFORM LOCALLY

Workflow:
1. Run baseline tests.
2. Intentionally break one silver transform rule (e.g., province parsing or validity flag behavior).
3. Re-run tests.
4. Capture which tests fail and why.
5. Revert your break.

Purpose: develop confidence in what the tests protect vs what they do not.

STEP C2) START FULL STACK WITH DOCKER AND HIT ENDPOINTS

Use full platform mode so Dagster + Postgres + API are all live.

Call at least:
? ‘GET /health‘

? ‘GET /programs?province=ON&limit=5‘
? ‘GET /programs/{program_stream_id}‘
? ‘GET /disciplines‘
? ‘GET /map/data.json‘
? ‘POST /pipeline/run‘

Capture examples of:
? Normal 200 response.
? At least one 4xx validation/auth/rate-limit response.

STEP C3) OBSERVE LOGS AND CONNECT PIPELINE STEPS TO API OUTPUTS

In your notes, create a ?cause/effect? map:
? Which materialized table changed.
? Which endpoint payload changed.
? Which field in response is sourced from which table/column.

PHASE D ? IMPROVEMENT READINESS (1?2 HOURS)

STEP D1) IDENTIFY ONE QUALITY CHECK TO ADD

Examples:
? Assert no duplicate ‘program_stream_id‘ in gold.
? Assert ‘province‘ is in the allowed code set.
? Assert description text is non-empty for active programs.

For interview: explain expected false positives/negatives.

STEP D2) IDENTIFY ONE INDEXING/PERFORMANCE IMPROVEMENT

Options:
? Add trigram index for ‘ILIKE‘ substring filters on discipline/school.
? Add composite index for common combined filters.
? Introduce materialized search view for high-frequency queries.

For interview: include expected query pattern and why b-tree alone may be insufficient for ‘%...%‘.

STEP D3) IDENTIFY ONE DS FEATURE EXTENSION

Choose one and define MVP:
? Program similarity search over descriptions (embedding + pgvector).
? Preference-aware ranking (province + discipline + quota + text relevance).

? Scenario simulation (e.g., discipline demand shifts by province).

For interview: include training/validation data assumptions and fairness caveats.

STEP D4) PREPARE YOUR ?FIRST 30 DAYS? PLAN

Week 1: Environment setup, lineage validation, endpoint smoke tests.

Week 2: Add one quality check + one observability improvement (logging/metrics).

Week 3: Ship one performance improvement with before/after benchmark.

Week 4: Prototype DS extension and present tradeoffs/next steps.

DEEP-UNDERSTANDING CHECKLIST (SELF-EVALUATION)

You are interview-ready when you can answer ?yes? to all:

? I can explain bronze/silver/gold using one real row example.
? I can describe exactly how ‘/programs‘ applies filters and pagination.
? I can name where auth and rate limiting are enforced.
? I can explain one migration and one index from the codebase.
? I can run the stack and troubleshoot a failed endpoint quickly.
? I can propose one quality and one performance enhancement with rationale.
? I can describe one DS extension with realistic implementation scope.

SUGGESTED FINAL DELIVERABLES FOR INTERVIEW PREP FOLDER

Create a personal folder (outside git if preferred) containing:
1. ‘architecture-summary.md‘ (your 8?10 bullets).
2. ‘record-trace.md‘ (one row traced bronze->silver->gold).
3. ‘endpoint-trace.md‘ (route->query->schema walkthrough).
4. ‘runtime-findings.md‘ (tests, logs, failures, fixes).
5. ‘first-30-days.md‘ (execution plan + milestones).

If you can present these five artifacts clearly, you will demonstrate genuine end-to-end mastery rather than surface familiarity.