Slice – Phase 5 Guide (Dev A)

Slice – Phase 5 Implementation Guide (Dev A)

Role: Dev A owns the core computational and orchestration work: SessionOrchestrator, risk snapshot integration, LLM prompt structure, API endpoint, and logging.

High-level principles:
- Never change core schemas from earlier phases (Thesis, Observation, Trade, Scenario, RiskReport, BacktestResult, embeddings).
- Never bypass repositories or ingestion pipelines to write directly to the DB.
- Let Dev B handle memory wrappers, unit tests, and docs, but you must define the interfaces they target.

Git conventions:
- Main branch: main
- Dev A feature branch: feature/phase5-devA (or similar)
- Dev B feature branch: feature/phase5-devB
- Always pull main before starting a new block of work.
- After finishing each major step below, run tests, commit, and push your branch, then tell Dev B to pull.

Step 0 – Initial Setup (both devs, but you drive it)
1. Pull the latest main:
   - git checkout main
   - git pull origin main
2. Create your feature branch:
   - git checkout -b feature/phase5-devA
3. Confirm tests pass (or at least that the test runner starts) before Phase 5 work.

Step 1 – Freeze Interfaces and Create the Phase 5 Interface Spec
(This step must be completed before Dev B does any serious implementation.)

1. Inspect the existing models and schemas from Phase 4:
   - Thesis, Observation, Trade, Scenario Pydantic models and DB tables.
   - RiskReport, BacktestResult models and tables.
   - Embedding configuration (model name, dimension).
2. Verify there are no Phase 5 schema changes required. If you find any, stop and resolve them before proceeding; Phase 5 is not allowed to mutate these schemas.
3. Create docs/phase5/phase5_interfaces.md with at least:
   - SessionOptions model: fields, types, default values.
   - SessionResponse model: fields, types.
   - SessionMode enum (if used).
   - SessionOrchestrator.run_session(text, options) signature and behavior summary.
   - risk.get_snapshot(thesis_id=None, portfolio_id=None) signature and return type.
   - memory.get_memory_context_for_text(text, k) signature and return type (even though Dev B will implement it).
   - Error/exception expectations: what raises vs what returns None.
4. Run tests (or at least lint/format).
5. Commit and push:
   - git add docs/phase5/phase5_interfaces.md
   - git commit -m "Add Phase 5 interface contract"
   - git push origin feature/phase5-devA
6. Notify Dev B: they must pull this before starting their work.

Step 2 – Add Core Session and Risk Skeletons (no real logic yet)
(Safe for both devs to work in parallel once this is pushed.)

1. Implement basic structures with NotImplemented logic:
   - src/slice/session/models.py:
      * SessionMode (enum)
      * SessionOptions
      * SessionResponse
   - src/slice/session/orchestrator.py:
      * class SessionOrchestrator with run_session(self, text, options) -> SessionResponse
        that currently raises NotImplementedError.
   - src/slice/risk/interface.py:
      * get_snapshot(thesis_id=None, portfolio_id=None) -> RiskSnapshot (stub,
        NotImplementedError).
      * RiskSnapshot model placeholder (fields left minimal for now).
2. Ensure imports are correct and the application starts.
3. Run tests to confirm nothing is broken by imports.
4. Commit and push:
   - git add src/slice/session/models.py src/slice/session/orchestrator.py src/slice/risk/interface.py
   - git commit -m "Add Phase 5 session and risk skeletons"
   - git push origin feature/phase5-devA
5. Tell Dev B to pull your branch into theirs (or to rebase their feature branch on main
   after your branch is merged).

Step 3 – Prompt Structure and LLM Template Functions
(You can do this while Dev B implements the memory wrapper internals.)

1. Create src/slice/session/prompts.py and define:
   - A canonical system prompt string for the Slice Macro Copilot.
   - Guardrail instructions (no price forecasting, no invented data, etc.).
   - Function build_prompt(memory_ctx, risk_ctx, user_text, options) -> list of messages
     that:
      * Orders blocks as: system → memory → risk → user query.
      * Handles None cases for memory_ctx and risk_ctx cleanly.
2. Ensure the prompt structure is deterministic and documented in comments.
3. Add unit tests for prompt order if you want; Dev B will likely add more tests later.
4. Commit and push:
   - git add src/slice/session/prompts.py
   - git commit -m "Add Phase 5 prompt builder"
   - git push origin feature/phase5-devA
5. Tell Dev B to pull so their tests and docs match the actual prompt structure.

Step 4 – Implement RiskSnapshot and risk.get_snapshot
(Dev B will write tests after you define this behavior; do not change the API afterwards.)

1. Finalize RiskSnapshot model in src/slice/risk/interface.py:
   - Define fields for metrics, rails, scenarios, etc., mapped from existing RiskReport/
     BacktestResult models.
2. Implement get_snapshot(thesis_id=None, portfolio_id=None):
   - Read from existing risk/portfolio repositories created in earlier phases.
   - Return a fully populated RiskSnapshot or raise a clear exception if data is missing.
3. Implement render_risk_snapshot_text(snapshot: RiskSnapshot) -> str:
   - Produce a compact, deterministic textual summary (no LLM usage).
4. Run focused tests (or at minimum quick scripts) to ensure this works end-to-end with
   real or fixture data.

5. Commit and push:
   - git add src/slice/risk/interface.py
   - git commit -m "Implement RiskSnapshot and risk.get_snapshot"
   - git push origin feature/phase5-devA
6. Notify Dev B explicitly: they should pull now and start writing unit tests for get_snapshot and render_risk_snapshot_text.

Step 5 – Implement SessionOrchestrator.run_session
(This step depends on Dev B having implemented get_memory_context_for_text.)

Precondition:
- Dev B has implemented src/slice/memory/interface.py::get_memory_context_for_text and pushed it.
- You have pulled their changes into your local feature branch or main.

1. Sync code:
   - git checkout feature/phase5-devA
   - git pull origin main   (if you are merging via PRs)
   - or rebase your branch on main if that's the team convention.
2. Implement run_session(self, text, options):
   - OBSERVE: Use the existing observation ingestion pipeline to:
     * Validate and persist a new Observation.
     * Obtain observation_id.
   - RECALL:
     * If options.use_memory is True: call get_memory_context_for_text(text, options.k).
     * Else: set memory_ctx=None.
   - RISK:
     * If options.use_risk is True: call get_snapshot(...) and render_risk_snapshot_text.
     * Else: set risk_ctx=None.
   - PROMPT:
     * Call build_prompt(memory_ctx, risk_ctx, text, options).
   - LLM:
     * Call the LLM client with the assembled messages, using model and temperature from config (temperature=0).
   - RESPONSE:
     * Assemble SessionResponse containing observation_id, the memory context string, the risk snapshot (as dict), the LLM answer, and timing data.
3. Add minimal internal logging/timing, but leave full logging to the next step.
4. Run tests with a mocked LLM client and mocked memory/risk calls if necessary.
5. Commit and push:
   - git add src/slice/session/orchestrator.py
   - git commit -m "Implement SessionOrchestrator.run_session"
   - git push origin feature/phase5-devA
6. Notify Dev B to pull; they will write orchestrator-level tests using mocks.

Step 6 – API Endpoint /api/v1/session/step
(Dev B will test this; you own the actual FastAPI (or equivalent) implementation.)

1. Add a new route in the API layer, e.g. src/slice/api/session_routes.py:
   - Define SessionStepRequest model (mirrors what client sends).
   - Implement POST /api/v1/session/step that:
     * Creates SessionOptions from the payload.
     * Invokes SessionOrchestrator.run_session(...).
     * Returns SessionResponse (or maps it into the HTTP response model).
2. Wire the router into the main FastAPI app if not already done.

3. Run API tests manually via curl or a test client if you want a quick sanity check.
4. Commit and push:
   - git add src/slice/api/session_routes.py (and any registration changes)
   - git commit -m "Add session_step API endpoint"
   - git push origin feature/phase5-devA
5. Tell Dev B to pull and implement endpoint tests against this route.

Step 7 – Logging / Session Events

1. Implement a logging helper, e.g. src/slice/session/logging.py:
   - log_session_event(response: SessionResponse, meta: SessionMeta) -> None.
   - SessionMeta should include model name, tokens in/out, latency, and hashes of memory and risk blocks.
2. Decide on storage:
   - DB table (session_events) or structured log file.
3. Hook log_session_event into SessionOrchestrator.run_session at the very end, wrapped in try/except so logging failures never break user responses.
4. Add a small test to ensure the logger can be called without raising.
5. Commit and push:
   - git add src/slice/session/logging.py src/slice/session/orchestrator.py
   - git commit -m "Add session logging for Phase 5"
   - git push origin feature/phase5-devA
6. Inform Dev B that logging fields and behavior are stable so they can document them.

Step 8 – End-to-End Integration Test (Primary Owner: Dev A)

1. Coordinate with Dev B to ensure:
   - Memory wrapper is fully implemented and tested.
   - Risk accessor is implemented and tested.
   - Orchestrator and API endpoint are wired correctly.
2. Add an integration test that:
   - Seeds the DB with minimal observation/thesis/risk data.
   - Mocks only the LLM client (everything else real).
   - Hits /api/v1/session/step with a realistic payload.
   - Asserts a new Observation is created, memory/risk paths are hit as expected, and a SessionResponse is returned.
3. Run the full test suite.
4. Commit and push:
   - git add tests/...
   - git commit -m "Add Phase 5 end-to-end integration test"
   - git push origin feature/phase5-devA

Step 9 – Final Sync and Merge Strategy

1. Ensure both your and Dev B's branches are up to date with main:
   - git checkout main
   - git pull origin main
2. Merge or rebase feature/phase5-devA and feature/phase5-devB into main via PRs:
   - Resolve conflicts in favor of the Phase 5 interface spec and orchestrator/ wrapper structure you agreed on in docs/phase5/phase5_interfaces.md.
3. Once merged:
   - Tag a release (e.g., v0.5.0).
   - Explicitly freeze:
     * SessionOptions and SessionResponse schemas.
     * SessionOrchestrator.run_session signature.

    * risk.get_snapshot and get_memory_context_for_text signatures.
4. Communicate clearly that any Phase 6 work must treat those interfaces as stable.

Follow this document in order. Do not change public interfaces after you publish them, unless both devs coordinate the change and update the docs and tests together.