Slice – Phase 5 Guide (Dev B)

Slice – Phase 5 Implementation Guide (Dev B)

Role: Dev B owns the memory wrapper implementation, unit tests, API tests, and documentation. You provide stability and verification around the interfaces that Dev A defines and implements.

High-level principles:
- Do not modify core schemas from earlier phases (Thesis, Observation, Trade, Scenario, RiskReport, BacktestResult, embeddings).
- Do not bypass repositories or ingestion pipelines to write directly to the DB.
- Treat docs/phase5/phase5_interfaces.md as the source of truth for Phase 5 public interfaces.

Git conventions:
- Main branch: main
- Dev B feature branch: feature/phase5-devB (or similar)
- Always pull main (or Dev A's branch if merged) before starting a new block of work.
- After finishing each major step, run tests, commit, and push your branch, then inform Dev A so they can sync and build on top of your changes.

Step 0 – Initial Setup and Waiting for Interface Spec
(You should not implement any Phase 5 logic until Dev A publishes the interface spec.)

1. Pull the latest main:
   - git checkout main
   - git pull origin main
2. Create your feature branch:
   - git checkout -b feature/phase5-devB
3. Wait for Dev A to finish Step 1 on their side: creation of docs/phase5/phase5_interfaces.md.
4. Once Dev A confirms the spec is ready, pull again:
   - git checkout main
   - git pull origin main (or pull their branch if you temporarily depend on it)
   - git checkout feature/phase5-devB
   - git merge main (or rebase, depending on workflow)
5. Open docs/phase5/phase5_interfaces.md and read it carefully. All of your work must respect the function signatures and model shapes defined there.

Step 1 – Repository Audit and Safe Entry Points
(You can do this as soon as the interface spec is in place.)

1. Scan existing code to understand official entrypoints:
   - Observation ingestion pipeline.
   - Memory context builder (from Phase 4).
   - Risk/portfolio repositories (you will not implement them, but you should know how they are accessed).
2. Create or update a short note inside docs/phase5/phase5_interfaces.md (or an adjacent file) listing:
   - All allowed read/write paths to the DB for Phase 5.
   - Any functions that must not be called from the new Phase 5 code (e.g., deprecated direct DB access).
3. Run tests to ensure nothing is broken by your documentation changes.
4. Commit and push:
   - git add docs/phase5/phase5_interfaces.md (and any new docs)

- git commit -m "Document Phase 5 allowed DB entrypoints"
- git push origin feature/phase5-devB

5. Inform Dev A that the DB entrypoint documentation is ready.

Step 2 – Memory Wrapper Skeletons (must match Dev A's spec)
(Dev A will have added stubs; your job is to implement the internals.)

Precondition:
- Dev A has created src/slice/memory/interface.py signatures in the interface spec.

1. Pull latest main (or Dev A's branch once merged):
   - git checkout main
   - git pull origin main
   - git checkout feature/phase5-devB
   - git merge main
2. Open src/slice/memory/interface.py and ensure it matches the spec:
   - get_memory_context_for_text(text: str, k: int) -> MemoryContext
   - search_similar_theses(text: str, k: int) -> list[ThesisSummary] (stub for future)
   - search_similar_trades(text: str, k: int) -> list[TradeSummary] (stub for future)
3. If any signatures differ from docs/phase5/phase5_interfaces.md, coordinate with Dev A before editing; the spec must be updated in lockstep.
4. Commit and push only if you changed placeholder code to match the spec:
   - git add src/slice/memory/interface.py
   - git commit -m "Align memory interface signatures with Phase 5 spec"
   - git push origin feature/phase5-devB

Step 3 – Implement get_memory_context_for_text
(This is your first major implementation piece; Dev A's orchestrator will depend on it.)

1. Implement get_memory_context_for_text(text: str, k: int) such that it:
   - Uses the existing observation ingestion pipeline to:
     * Normalize and validate the incoming text.
     * Persist a new Observation if required by the agreed pattern.
     * Obtain observation_id.
   - Uses the existing MemoryContextBuilder (or equivalent Phase 4 class) to:
     * Query embeddings via pgvector or other vector store.
     * Retrieve the top-k similar Observations.
     * Construct a memory context block as plaintext (in the format expected by Dev A's prompt builder).
2. Define MemoryContext (if not already done) with fields like:
   - observation_id: str
   - context_block: str
   - source_observation_ids: list[str]
3. Write unit tests for get_memory_context_for_text:
   - Use fixtures or a test DB with a few Observations pre-seeded.
   - Assert that:
     * k=3 returns three ids when available.
     * The context_block contains the expected observation texts.
     * Edge cases (no similar items) are handled gracefully.
4. Run the test suite (or at least the relevant test module).
5. Commit and push:
   - git add src/slice/memory/interface.py tests/...
   - git commit -m "Implement get_memory_context_for_text and tests"
   - git push origin feature/phase5-devB
6. Notify Dev A: they must pull your branch (or the merged main) before they fully

implement SessionOrchestrator.run_session.

Step 4 – Stubs for search_similar_theses and search_similar_trades
(These are forward-looking; they must not break anything.)

1. Implement search_similar_theses and search_similar_trades as stubs that:
   - Accept text and k parameters.
   - For now, return an empty list or a NotImplemented placeholder, but do not raise errors during normal operation.
2. Add brief unit tests confirming they return empty lists and do not crash.
3. Commit and push:
   - git add src/slice/memory/interface.py tests/...
   - git commit -m "Add placeholder implementations for thesis/trade similarity"
   - git push origin feature/phase5-devB

Step 5 – Prompt Tests and Wording Support
(Dev A owns the structure of prompts; you support testing and wording.)

Precondition:
- Dev A has implemented src/slice/session/prompts.py with build_prompt(...) function.

1. Pull latest main (or Dev A's branch if not yet merged):
   - git checkout main
   - git pull origin main
   - git checkout feature/phase5-devB
   - git merge main
2. Add tests for build_prompt:
   - Using fake MemoryContext and RiskSnapshot objects, assert that:
     * The system prompt is always first.
     * Memory context, if provided, appears before risk context.
     * User query always appears last.
     * No blocks are missing when options.use_memory/use_risk are True.
3. If minor wording changes to system or guardrail prompts are needed, propose them to Dev A and only change them after agreement.
4. Commit and push:
   - git add tests/... (and prompts file only if wording changes were agreed)
   - git commit -m "Add tests for Phase 5 prompt builder"
   - git push origin feature/phase5-devB
5. Inform Dev A so they can verify the tests and adjust orchestrator logic if needed.

Step 6 – Risk Accessor Tests
(Dev A implements risk.get_snapshot; you verify correctness with tests.)

Precondition:
- Dev A has implemented RiskSnapshot, get_snapshot, and render_risk_snapshot_text in src/slice/risk/interface.py.

1. Pull latest main:
   - git checkout main
   - git pull origin main
   - git checkout feature/phase5-devB
   - git merge main
2. Write tests for get_snapshot and render_risk_snapshot_text:
   - Use fixtures with synthetic RiskReport/BacktestResult data or a pre-populated test DB.

- Assert that:
    * get_snapshot returns a RiskSnapshot with all expected fields populated.
    * render_risk_snapshot_text includes the key metrics and rails names and values.
    * Behavior is deterministic (no randomness, no missing keys).
3. Run tests.
4. Commit and push:
  - git add tests/...
  - git commit -m "Add tests for RiskSnapshot access and text rendering"
  - git push origin feature/phase5-devB
5. Notify Dev A if any assumptions in their implementation conflict with the spec or tests.

Step 7 – Orchestrator Tests with Mocks
(Dev A implements SessionOrchestrator; you validate behavior with mocked dependencies.)

Precondition:
- Dev A has implemented SessionOrchestrator.run_session in
  src/slice/session/orchestrator.py.

1. Pull latest main:
  - git checkout main
  - git pull origin main
  - git checkout feature/phase5-devB
  - git merge main
2. Add tests for SessionOrchestrator.run_session:
  - Mock get_memory_context_for_text, get_snapshot, and the LLM client:
    * Ensure memory is only called when options.use_memory is True.
    * Ensure risk is only called when options.use_risk is True.
    * Ensure LLM is always called, with messages built via build_prompt.
  - Assert that SessionResponse includes:
    * observation_id (from mocked ingestion).
    * memory_context and risk_context (when enabled).
    * llm_response and timing info.
3. Run tests.
4. Commit and push:
  - git add tests/...
  - git commit -m "Add tests for SessionOrchestrator.run_session"
  - git push origin feature/phase5-devB
5. Inform Dev A about any behavioral mismatches your tests reveal.

Step 8 – API Endpoint Tests
(Dev A implements the endpoint; you verify it.)

Precondition:
- Dev A has implemented POST /api/v1/session/step in the API layer.

1. Pull latest main:
  - git checkout main
  - git pull origin main
  - git checkout feature/phase5-devB
  - git merge main
2. Add API tests using your framework's test client:
  - Test minimal payload (default options).
  - Test payloads with use_risk=True and use_risk=False.
  - Mock the orchestrator so that these tests isolate only the HTTP layer and
    serialization/deserialization logic.

3. Assert that the response matches SessionResponse schema exactly.
4. Run tests.
5. Commit and push:
   - git add tests/...
   - git commit -m "Add API tests for /api/v1/session/step"
   - git push origin feature/phase5-devB
6. Notify Dev A that the endpoint is covered; any future changes should update both implementation and tests.

Step 9 – Logging Tests and Documentation

Precondition:
- Dev A has implemented the logging helper and wired it into the orchestrator.

1. Pull latest main:
   - git checkout main
   - git pull origin main
   - git checkout feature/phase5-devB
   - git merge main
2. Add minimal tests for the logging helper:
   - Ensure log_session_event can be called with a fake SessionResponse and SessionMeta without throwing exceptions.
3. Create or update documentation:
   - docs/phase5/phase5_integration.md: high-level description of Phase 5 behavior, orchestrator flow, and how memory/risk are wired in.
   - docs/phase5/prompts.md: details of system prompts, guardrails, and the structure of memory/risk blocks.
   - Document which logging fields are stored and where (DB table vs log file).
4. Run tests and spell-check docs if applicable.
5. Commit and push:
   - git add docs/phase5/*.md tests/...
   - git commit -m "Add Phase 5 integration docs and logging tests"
   - git push origin feature/phase5-devB
6. Inform Dev A that docs are ready and ask them to review for correctness.

Step 10 – Final Sync and Merge Strategy (with Dev A)

1. Coordinate with Dev A to ensure both feature branches are up to date with main:
   - git checkout main
   - git pull origin main
   - git checkout feature/phase5-devB
   - git merge main (resolve any conflicts, especially in docs and tests).
2. Work with Dev A to merge both feature branches into main via PRs:
   - Resolve conflicts in favor of the agreed spec in docs/phase5/phase5_interfaces.md.
3. After merging:
   - Ensure all tests pass on main.
   - Confirm that:
     * SessionOptions and SessionResponse are treated as stable public interfaces.
     * SessionOrchestrator.run_session, get_memory_context_for_text, and risk.get_snapshot signatures match the spec and tests.
4. Once confirmed, treat Phase 5 as frozen from your perspective, and prepare for Phase 6 work to build on these stable interfaces.

Follow this document in order. Do not change public interfaces on your own; any change must be coordinated with Dev A and reflected in tests and documentation.