# **M1 — Core Domain & Minimal UC-4 Path (≤300 steps)**

## **A. Lock requirements and design contracts (Solidity)**

1. Create contracts/src/interfaces/IMultiStepAdapter.sol with NatSpec: quote(bytes calldata), execute(bytes calldata); both return standardized AdapterResult.
2. Define AdapterResult struct in contracts/src/common/Types.sol with fields: success, spent, received, gasUsed, auxData.
3. Add StepKind enum in Types.sol (e.g., UNIV3\_SWAP, NOOP), and Step struct {kind, adapter, payload, tokenIn, tokenOut, amountIn, minAmountOut}.
4. Create contracts/src/policy/RouteRegistry.sol with mappings: routeId => Step[], routeId => owner, and events RouteRegistered, RouteUpdated, RouteDisabled.
5. Implement registerRoute(Step[] calldata, bytes32 salt) returning routeId = keccak256(abi.encode(steps, msg.sender, salt)); store owner.
6. Implement updateRoute(routeId, Step[] calldata) only callable by owner; emit RouteUpdated.
7. Implement disableRoute(routeId) only owner; emit RouteDisabled.
8. Create contracts/src/policy/PauseGuardian.sol with roles: owner, guardian; functions pause(), unpause(), and modifier whenNotPaused.
9. Create contracts/src/policy/PolicyGuards.sol with a minimal guard validateStep(Step calldata) enforcing non-zero addresses, non-zero amountIn when required, and allowed adapters set.
10. Add setAllowedAdapter(address adapter, bool allowed) onlyOwner; mapping used in validation.
11. Create contracts/src/settlement/SettlementVault.sol: custodial ERC20 vault with deposit(token, amount), withdraw(to, token, amount) onlyOwner, and transfer(to, token, amount) onlyExecutor.
12. Create contracts/src/execution/BundleExecutor.sol with state: RouteRegistry, PolicyGuards, PauseGuardian, SettlementVault.
13. In BundleExecutor, add execute(bytes32 routeId, uint256 amountIn, uint256 minTotalOut, address tokenIn, address tokenOut, address recipient); returns totalOut.
14. In execute, load steps from RouteRegistry, require whenNotPaused, iterate steps, call adapters via IMultiStepAdapter.execute(step.payload) after performing transfer approvals as needed.
15. Track totalSpent/totalReceived; enforce slippage totalReceived >= minTotalOut.
16. Emit BundleExecuted(routeId, msg.sender, amountIn, totalReceived, block.timestamp).
17. Add onlyOwner/onlyGuardian errors to Errors.sol and use custom errors throughout.
18. Use OZ ReentrancyGuard in BundleExecutor and SettlementVault.
19. Add contracts/src/venues/adapters/NoOpAdapter.sol implementing IMultiStepAdapter returning AdapterResult(true,0,0,0,"").
20. Add contracts/src/venues/adapters/UniswapV3Adapter.sol with immutable IUniV3Router/IQuoterV2 addresses (passed in ctor).
21. Define UniV3SwapParams struct (tokenIn, tokenOut, fee, recipient, amountIn, amountOutMin, sqrtPriceLimitX96, deadline).
22. Implement quote(payload) parsing UniV3SwapParams and invoking QuoterV2 to return expected received.
23. Implement execute(payload) approving router for tokenIn and calling exactInputSingle or exactInput based on hop count; return AdapterResult with gasleft() delta.
24. Add events AdapterQuoted, AdapterExecuted in adapters for observability.
25. Implement recoverERC20(token, to, amount) onlyOwner in adapters and vault.
26. Add contracts/src/execution/MultiSwapRouter.sol (thin) to precompute step payloads off-chain parity; exposes helper pure/ view functions (optional but keeps parity clear).

## **B. Contract configuration & deployment scripts**

1. Extend contracts/script/Deploy.s.sol to deploy PauseGuardian, PolicyGuards, RouteRegistry, SettlementVault, BundleExecutor, NoOpAdapter, UniswapV3Adapter with env addresses for router/quoter.
2. Wire constructor dependencies: pass contract addresses to BundleExecutor.
3. Extend contracts/script/Configure.s.sol to setAllowedAdapter(UniswapV3Adapter,true) and setAllowedAdapter(NoOpAdapter,true).
4. In Configure.s.sol, set SettlementVault owner = ops multisig (from env) and executor = BundleExecutor.
5. Update deployments/local.json writer to include these contracts and ABI paths.

## **C. Foundry tests — unit (Solidity)**

1. Create contracts/test/policy/RouteRegistry.t.sol testing register/update/disable ownership checks and event emissions.
2. Create contracts/test/policy/PolicyGuards.t.sol testing allowed adapter gating and validation failures.
3. Create contracts/test/policy/PauseGuardian.t.sol testing pause/unpause gating BundleExecutor.
4. Create contracts/test/settlement/SettlementVault.t.sol testing deposit/withdraw/transfer constraints and onlyOwner/executor.
5. Create contracts/test/venues/NoOpAdapter.t.sol covering trivial quote/execute.
6. Create contracts/test/venues/UniswapV3Adapter.unit.t.sol with mocks (router/quoter) ensuring payload encoding/decoding and approvals (no external calls).
7. Create contracts/test/execution/BundleExecutor.unit.t.sol using a stub adapter to assert step iteration, minTotalOut enforcement, events, and reentrancy guard.
8. Add invariant test skeleton ensuring executor doesn’t lose tokens from vault without accounting (forge-std invariant framework).
9. Run forge test -vvv locally; make red tests pass before moving on.

## **D. Foundry tests — integration on Anvil (Solidity)**

1. Create contracts/test/integration/Univ3.singleHop.t.sol that forks Sepolia/Mainnet (env) or uses a local deployment of WETH + MockERC20 + local Uni v3; performs a real exactInputSingle.
2. Create contracts/test/integration/BundleExecutor.univ3Route.t.sol that registers a route (token A → token B) and executes end-to-end with minTotalOut checks.
3. Seed test accounts with tokens via mint faucets or local mocks; abstract into BaseIntegration.t.sol.
4. Assert gas upper bounds (set generous caps) to catch regressions.

## **E. Cross-contract events & indexing**

1. Standardize event topics: route events in RouteRegistry, execution events in BundleExecutor, adapter events in their contracts.
2. Ensure all events include routeId or txHash anchors for later telemetry.

## **F. TypeScript domain types & ABI sync**

1. Export ABIs from Foundry out/ to packages/core-exec/src/abi/\*.json via a small copy script.
2. Add packages/core-exec/src/BundleTypes.ts with TS mirrors of Step, StepKind, AdapterResult, and RouteSpec.
3. Implement packages/core-exec/src/encode.ts to encode UniV3SwapParams into adapter payloads (using viem / ethers v6).
4. Implement packages/core-exec/src/decoders.ts for debugging (decode payload → human-readable).
5. Add a narrow PolicyClient in packages/core-exec/src/policyClient.ts (read/write RouteRegistry & PolicyGuards).
6. Add ExecutorClient in packages/core-exec/src/executorClient.ts (call execute, read events).
7. Add VaultClient in packages/core-exec/src/vaultClient.ts (deposit/withdraw for test flows).

## **G. TS Uniswap v3 adapter client (off-chain parity)**

1. In packages/adapters-evm/src/UniswapV3AdapterClient.ts, implement buildSingleHop({tokenIn, tokenOut, fee, amountIn, minOut}) → Step.
2. Implement quoteSingleHop(provider, params) → bigint using QuoterV2 ABI; compare with on-chain quote.
3. Implement buildMultiHop(path[], amountIn, minOut) → Step encoding exactInput path.
4. Add slippage helper: applyBps(value, bps).

## **H. Route builder (TS)**

1. Implement packages/core-exec/src/RouteBuilder.ts with buildRoute(steps: Step[]) → RouteSpec and register(policyClient, routeSpec).
2. Add simulate(policyClient, executorClient, routeSpec, amountIn) that calls adapter client’s quote per step and estimates aggregate minOut.
3. Add validation: ensure each step tokenOut == next step tokenIn, adapters allowed, and amounts sane.

## **I. CLI wiring (TS)**

1. Extend packages/cli/src/commands/route.ts with subcommands:

* build --single-hop ... → prints JSON RouteSpec.
* register --file route.json → registers, prints routeId.
* quote --route-id --amount-in → aggregates per-step quotes.
* execute --route-id --amount-in --min-total-out --recipient.

1. Implement --network and --rpc-url flags with defaults from .env.
2. Ensure CLI exits non-zero on failed validations or tx reverts.

## **J. Config files (policies & venues)**

1. Fill config/venues.json with entries for uniswapV3 (router/quoter per chain), and noop.
2. Fill config/policies.json with allowedAdapters and slippageDefaultBps.
3. Add scripts/dev/sync-config.ts writing on-chain allowlist to policies.json after Configure.s.sol runs.

## **K. End-to-end local flow (scripts)**

1. Implement packages/core-exec/src/run/sepoliaExecute.ts:

* Loads config; builds a single-hop route; registers; executes with minTotalOut from off-chain quote minus slippage.

1. Add root script "demo:singlehop": "tsx packages/core-exec/src/run/sepoliaExecute.ts".
2. Verify on Anvil first, then Sepolia fork (read-only or funded throwaway key).

## **L. Observability stubs**

1. Emit console logs in TS clients at debug level for: routeId creation, quotes, and receipts.
2. Ensure BundleExecuted and adapter events are consumed by TS to print human-readable summaries.

## **M. Hardening (minimum viable)**

1. Add checks in BundleExecutor.execute: forbid unknown StepKind, forbid zero steps, cap steps length (e.g., ≤8 for MVP).
2. Require recipient non-zero and not address(this) unless explicitly allowed.
3. Zero approvals after swap if non-router approvals remain (best-effort cleanup).
4. Use SafeERC20 for all token transfers/approvals.
5. In RouteRegistry, prevent re-registering identical route by the same owner unless updateRoute is used; document expected behavior.
6. In UniswapV3Adapter, enforce deadline >= block.timestamp.

## **N. Gas & sizing**

1. Add forge snapshot and a test asserting gas upper bound for execute with 1–2 steps.
2. Document gas expectations in docs/dev/gas.md.

## **O. Security & safety rails (M1 scope)**

1. Add onlyOwner to sensitive mutators; set owners to deployer in local; placeholder for multisig in higher milestones.
2. Add pause() in BundleExecutor via PauseGuardian modifier to cover execute.
3. Add nonReentrant to execute and sensitive vault paths.
4. Add a static-analysis job in CI (slither if available or Foundry’s forge fmt && forge inspect checks) as informational.

## **P. Docs & examples**

1. Update docs/content/uc4-multiswap-permissions.mdx with the M1 architecture diagram and a “happy-path” walkthrough.
2. Update README Quickstart to include: build route → register → quote → execute (with example command lines).
3. Add examples/route.singlehop.json and examples/route.multihop.json.

## **Q. CI extensions (fast but meaningful)**

1. Extend CI to run forge test --match-path contracts/test/integration/\* -vv on push to PR (label integration job).
2. Add a minimal pnpm -w test job using vitest for TS unit tests (encoders/decoders/clients).
3. Gate PR merge on unit tests + typecheck + linters; allow integration as non-blocking until M2.

## **R. TS unit tests (Vitest)**

1. packages/core-exec/test/encode.spec.ts: encodes payload equals Solidity-side abi encoding (use known vectors).
2. packages/adapters-evm/test/univ3.client.spec.ts: quote matches on-chain quoter (delta within tolerance).
3. packages/core-exec/test/routeBuilder.multistep.spec.ts: validation failures for broken token chains; success for valid chains.
4. packages/cli/test/route.cli.spec.ts: build/register/quote dry-run (mocked provider).

## **S. Address and artifact sanity**

1. Ensure scripts/dev/print-addresses.ts also prints adapter allowlist and guardian/owner addresses.
2. Add a scripts/dev/check-allowlist.ts that compares on-chain allowed adapters vs policies.json, exits non-zero on drift.

## **T. Failure-path coverage**

1. Foundry test: executing a disabled route must revert with custom RouteDisabled error.
2. Foundry test: when paused, execute reverts with Paused.
3. Foundry test: minTotalOut slippage too strict → revert InsufficientOutput().
4. TS test: CLI execute with missing recipient → CLI validation error (no tx sent).
5. TS test: adapter not allowed → PolicyGuards validation failure surfaced to CLI.

## **U. Migration & compatibility notes (documented)**

1. Create docs/dev/migrations/m1.md describing data model, routeId derivation, and any breaking assumptions from prototypes.
2. Note that route owners are EOA for M1; multisig planned later.

## **V. Minimal permissioning flows**

1. In Configure.s.sol, set guardian = OWNER initially; document upgrade path to separate guardian in later milestones.
2. Provide transferOwnership flow examples for RouteRegistry, PolicyGuards, SettlementVault.

## **W. Safety on approvals**

1. In adapter, before approve(router, amount), set to zero if current allowance > 0 (some ERC20s require this).
2. After swap, best-effort approve(router, 0) to reduce residual risk; ignore failure if token non-standard (documented).

## **X. Token handling edge cases**

1. Ensure WETH unwrap only when tokenOut == native is explicitly requested (M1: skip unwrap; keep ERC20 only).
2. Add test to prevent native ETH exposure through adapter in M1.

## **Y. Deterministic routeId parity (TS/Solidity)**

1. Implement computeRouteId(routeSpec, owner, salt) in TS exactly matching Solidity keccak256; add cross-tests comparing both.
2. Ensure JSON serialization order is deterministic (sort fields) before hashing.

## **Z. Minimal telemetry hooks (stubs)**

1. Add TS hook interface TelemetrySink with onRouteRegistered, onBundleExecuted; default sink logs to console.
2. Wire CLI to call sink after successful operations.

## **AA. Error taxonomy surfaced to CLI**

1. Map common reverts (Paused, InsufficientOutput, AdapterNotAllowed) to friendly CLI messages with remediation tips.
2. Add --json flag to CLI to output machine-readable results for agents.

## **AB. Developer ergonomics**

1. Add pnpm demo:singlehop to README with expected console output and a note on funding the deployer on Sepolia.
2. Provide a one-liner script scripts/dev/reset-local.ts to stop anvil, wipe caches, redeploy, reseed mocks.

## **AC. Finalization gates for M1**

1. Run pnpm doctor (from M0) and ensure green.
2. Ensure forge test (unit + integration) passes locally and in CI.
3. Perform a manual E2E:  
    - Start Anvil.  
    - Deploy + Configure.  
    - Build route via CLI.  
    - Register route; capture routeId.  
    - Quote; verify minTotalOut.  
    - Execute; verify balances changed as expected.  
    - Pause; verify execute reverts.  
    - Unpause; execute succeeds.
4. Tag the repo m1-core-mvp and generate release notes summarizing artifacts, ABIs, and known limitations.