M1 Hierarchy

A. Solidity contracts — scaffolds & ownership/guards

A1. File tree (under contracts/)

contracts/

src/common/Types.sol

src/common/Errors.sol

src/interfaces/IMultiStepAdapter.sol

src/policy/RouteRegistry.sol

src/policy/PolicyGuards.sol

src/policy/PauseGuardian.sol

src/settlement/SettlementVault.sol

src/execution/BundleExecutor.sol

src/venues/adapters/NoOpAdapter.sol

src/venues/adapters/UniswapV3Adapter.sol

script/Deploy.s.sol

script/Configure.s.sol

test/...

A2. Common types & errors

src/common/Types.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

library Types {

enum StepKind { NOOP, UNIV3\_SWAP }

struct AdapterResult {

bool success;

uint256 spent;

uint256 received;

uint256 gasUsed;

bytes auxData;

}

struct Step {

StepKind kind;

address adapter;

bytes payload;

address tokenIn;

address tokenOut;

uint256 amountIn;

uint256 minAmountOut;

}

}

src/common/Errors.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

library Errors {

error Unauthorized();

error Paused();

error InvalidInput();

error InvalidAdapter();

error RouteDisabled();

error InsufficientOutput(uint256 want, uint256 got);

}

A3. Adapter interface

src/interfaces/IMultiStepAdapter.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

import {Types} from "../common/Types.sol";

interface IMultiStepAdapter {

event AdapterQuoted(bytes32 indexed routeId, uint256 expectedOut);

event AdapterExecuted(bytes32 indexed routeId, uint256 spent, uint256 received);

function quote(bytes calldata payload) external view returns (Types.AdapterResult memory);

function execute(bytes calldata payload) external returns (Types.AdapterResult memory);

}

A4. Policy: Pause guardian

src/policy/PauseGuardian.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

import {Errors} from "../common/Errors.sol";

contract PauseGuardian {

address public owner;

address public guardian;

bool public paused;

event Paused(address indexed by);

event Unpaused(address indexed by);

event GuardianUpdated(address indexed newGuardian);

event OwnerUpdated(address indexed newOwner);

modifier onlyOwner() { if (msg.sender != owner) revert Errors.Unauthorized(); \_; }

modifier onlyGuardianOrOwner() { if (msg.sender != guardian && msg.sender != owner) revert Errors.Unauthorized(); \_; }

modifier whenNotPaused() { if (paused) revert Errors.Paused(); \_; }

constructor(address \_owner, address \_guardian) {

owner = \_owner; guardian = \_guardian;

}

function pause() external onlyGuardianOrOwner { paused = true; emit Paused(msg.sender); }

function unpause() external onlyOwner { paused = false; emit Unpaused(msg.sender); }

function setGuardian(address g) external onlyOwner { guardian = g; emit GuardianUpdated(g); }

function setOwner(address o) external onlyOwner { owner = o; emit OwnerUpdated(o); }

}

A5. Policy: Route registry

src/policy/RouteRegistry.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

import {Types} from "../common/Types.sol";

import {Errors} from "../common/Errors.sol";

contract RouteRegistry {

struct RouteMeta { address owner; bool disabled; }

mapping(bytes32 => Types.Step[]) private \_steps;

mapping(bytes32 => RouteMeta) public routes;

event RouteRegistered(bytes32 indexed routeId, address indexed owner, uint256 steps);

event RouteUpdated(bytes32 indexed routeId, uint256 steps);

event RouteDisabled(bytes32 indexed routeId, bool disabled);

function computeRouteId(

Types.Step[] memory steps, address owner, bytes32 salt

) public pure returns (bytes32) {

return keccak256(abi.encode(steps, owner, salt));

}

function stepsOf(bytes32 routeId) external view returns (Types.Step[] memory) {

return \_steps[routeId];

}

function registerRoute(Types.Step[] memory steps, bytes32 salt) external returns (bytes32 routeId) {

routeId = computeRouteId(steps, msg.sender, salt);

if (routes[routeId].owner != address(0)) revert Errors.InvalidInput(); // already exists

routes[routeId] = RouteMeta({owner: msg.sender, disabled: false});

for (uint i; i < steps.length; ++i) { \_steps[routeId].push(steps[i]); }

emit RouteRegistered(routeId, msg.sender, steps.length);

}

function updateRoute(bytes32 routeId, Types.Step[] memory steps) external {

if (routes[routeId].owner != msg.sender) revert Errors.Unauthorized();

delete \_steps[routeId];

for (uint i; i < steps.length; ++i) { \_steps[routeId].push(steps[i]); }

emit RouteUpdated(routeId, steps.length);

}

function setDisabled(bytes32 routeId, bool disabled) external {

if (routes[routeId].owner != msg.sender) revert Errors.Unauthorized();

routes[routeId].disabled = disabled;

emit RouteDisabled(routeId, disabled);

}

}

A6. Policy: Adapter allowlist

src/policy/PolicyGuards.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

import {Types} from "../common/Types.sol";

import {Errors} from "../common/Errors.sol";

contract PolicyGuards {

address public owner;

mapping(address => bool) public allowedAdapter;

event AllowedAdapterSet(address indexed adapter, bool allowed);

event OwnerUpdated(address indexed owner);

modifier onlyOwner() { if (msg.sender != owner) revert Errors.Unauthorized(); \_; }

constructor(address \_owner) { owner = \_owner; }

function setOwner(address o) external onlyOwner { owner = o; emit OwnerUpdated(o); }

function setAllowedAdapter(address a, bool allowed) external onlyOwner {

allowedAdapter[a] = allowed; emit AllowedAdapterSet(a, allowed);

}

function validateStep(Types.Step calldata s) external view {

if (!allowedAdapter[s.adapter]) revert Errors.InvalidAdapter();

if (s.tokenIn == address(0) || s.tokenOut == address(0)) revert Errors.InvalidInput();

if (s.amountIn == 0) revert Errors.InvalidInput();

}

}

A7. Settlement vault

src/settlement/SettlementVault.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

import {IERC20} from "openzeppelin-contracts/contracts/token/ERC20/IERC20.sol";

import {SafeERC20} from "openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol";

import {Errors} from "../common/Errors.sol";

contract SettlementVault {

using SafeERC20 for IERC20;

address public owner;

address public executor;

event OwnerUpdated(address indexed owner);

event ExecutorUpdated(address indexed executor);

event Deposited(address indexed token, address indexed from, uint256 amount);

event Transferred(address indexed token, address indexed to, uint256 amount);

modifier onlyOwner(){ if (msg.sender != owner) revert Errors.Unauthorized(); \_; }

modifier onlyExecutor(){ if (msg.sender != executor) revert Errors.Unauthorized(); \_; }

constructor(address \_owner){ owner = \_owner; }

function setOwner(address o) external onlyOwner { owner = o; emit OwnerUpdated(o); }

function setExecutor(address e) external onlyOwner { executor = e; emit ExecutorUpdated(e); }

function deposit(address token, uint256 amount) external {

IERC20(token).safeTransferFrom(msg.sender, address(this), amount);

emit Deposited(token, msg.sender, amount);

}

function transfer(address to, address token, uint256 amount) external onlyExecutor {

IERC20(token).safeTransfer(to, amount);

emit Transferred(token, to, amount);

}

function withdraw(address to, address token, uint256 amount) external onlyOwner {

IERC20(token).safeTransfer(to, amount);

}

}

A8. Bundle executor

src/execution/BundleExecutor.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

import {Types} from "../common/Types.sol";

import {Errors} from "../common/Errors.sol";

import {IMultiStepAdapter} from "../interfaces/IMultiStepAdapter.sol";

import {PolicyGuards} from "../policy/PolicyGuards.sol";

import {RouteRegistry} from "../policy/RouteRegistry.sol";

import {PauseGuardian} from "../policy/PauseGuardian.sol";

import {SettlementVault} from "../settlement/SettlementVault.sol";

import {IERC20} from "openzeppelin-contracts/contracts/token/ERC20/IERC20.sol";

import {SafeERC20} from "openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol";

contract BundleExecutor {

using SafeERC20 for IERC20;

RouteRegistry public registry;

PolicyGuards public guards;

PauseGuardian public pause;

SettlementVault public vault;

uint256 public constant MAX\_STEPS = 8;

event BundleExecuted(bytes32 indexed routeId, address indexed caller, uint256 amountIn, uint256 totalReceived);

constructor(RouteRegistry \_r, PolicyGuards \_g, PauseGuardian \_p, SettlementVault \_v) {

registry = \_r; guards = \_g; pause = \_p; vault = \_v;

}

modifier whenNotPaused() { pause.whenNotPaused(); \_; } // delegate to PauseGuardian

function execute(

bytes32 routeId,

uint256 amountIn,

uint256 minTotalOut,

address tokenIn,

address tokenOut,

address recipient

) external whenNotPaused returns (uint256 totalOut) {

if (recipient == address(0)) revert Errors.InvalidInput();

// pull funds from vault to the first adapter (router pulls via adapter)

// for MVP we transfer to adapter or approve downstream router inside adapter

Types.Step[] memory steps = registry.stepsOf(routeId);

RouteRegistry.RouteMeta memory meta = registry.routes(routeId);

if (meta.disabled) revert Errors.RouteDisabled();

if (steps.length == 0 || steps.length > MAX\_STEPS) revert Errors.InvalidInput();

// Seed first token into adapter execution context: send to adapter if needed

IERC20(tokenIn).safeIncreaseAllowance(steps[0].adapter, amountIn);

uint256 remaining = amountIn;

for (uint i; i < steps.length; ++i) {

guards.validateStep(steps[i]);

Types.AdapterResult memory res = IMultiStepAdapter(steps[i].adapter).execute(steps[i].payload);

if (!res.success) revert Errors.InvalidInput();

if (res.received < steps[i].minAmountOut) revert Errors.InsufficientOutput(steps[i].minAmountOut, res.received);

remaining = res.received;

if (i < steps.length - 1) {

// approve next adapter to spend current tokenOut (== next tokenIn)

IERC20(steps[i].tokenOut).safeApprove(steps[i+1].adapter, 0);

IERC20(steps[i].tokenOut).safeIncreaseAllowance(steps[i+1].adapter, remaining);

}

}

totalOut = remaining;

if (totalOut < minTotalOut) revert Errors.InsufficientOutput(minTotalOut, totalOut);

// pay recipient

IERC20(tokenOut).safeTransfer(recipient, totalOut);

emit BundleExecuted(routeId, msg.sender, amountIn, totalOut);

}

}

A9. Adapters

src/venues/adapters/NoOpAdapter.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

import {IMultiStepAdapter} from "../../interfaces/IMultiStepAdapter.sol";

import {Types} from "../../common/Types.sol";

contract NoOpAdapter is IMultiStepAdapter {

function quote(bytes calldata) external pure returns (Types.AdapterResult memory r) {

return Types.AdapterResult(true, 0, 0, 0, "");

}

function execute(bytes calldata) external pure returns (Types.AdapterResult memory r) {

return Types.AdapterResult(true, 0, 0, 0, "");

}

}

src/venues/adapters/UniswapV3Adapter.sol (MVP: exactInputSingle only; exactInput path optional later)

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

import {IMultiStepAdapter} from "../../interfaces/IMultiStepAdapter.sol";

import {Types} from "../../common/Types.sol";

import {IERC20} from "openzeppelin-contracts/contracts/token/ERC20/IERC20.sol";

import {SafeERC20} from "openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol";

interface IQuoterV2 {

function quoteExactInputSingle(

address tokenIn, address tokenOut, uint24 fee, uint256 amountIn, uint160 sqrtPriceLimitX96

) external returns (uint256 amountOut);

}

interface ISwapRouter {

struct ExactInputSingleParams {

address tokenIn; address tokenOut; uint24 fee; address recipient; uint256 deadline;

uint256 amountIn; uint256 amountOutMinimum; uint160 sqrtPriceLimitX96;

}

function exactInputSingle(ExactInputSingleParams calldata params) external payable returns (uint256 amountOut);

}

contract UniswapV3Adapter is IMultiStepAdapter {

using SafeERC20 for IERC20;

ISwapRouter public immutable router;

IQuoterV2 public immutable quoter;

struct UniV3SwapParams {

address tokenIn; address tokenOut; uint24 fee; address recipient;

uint256 amountIn; uint256 minOut; uint160 sqrtPriceLimitX96; uint256 deadline;

}

constructor(ISwapRouter \_router, IQuoterV2 \_quoter) { router = \_router; quoter = \_quoter; }

function \_decode(bytes calldata p) internal pure returns (UniV3SwapParams memory sp) {

return abi.decode(p, (UniV3SwapParams));

}

function quote(bytes calldata payload) external returns (Types.AdapterResult memory r) {

UniV3SwapParams memory sp = \_decode(payload);

uint256 outAmt = quoter.quoteExactInputSingle(

sp.tokenIn, sp.tokenOut, sp.fee, sp.amountIn, sp.sqrtPriceLimitX96

);

return Types.AdapterResult(true, sp.amountIn, outAmt, 0, "");

}

function execute(bytes calldata payload) external returns (Types.AdapterResult memory r) {

uint256 g0 = gasleft();

UniV3SwapParams memory sp = \_decode(payload);

IERC20(sp.tokenIn).safeApprove(address(router), 0);

IERC20(sp.tokenIn).safeIncreaseAllowance(address(router), sp.amountIn);

ISwapRouter.ExactInputSingleParams memory params = ISwapRouter.ExactInputSingleParams({

tokenIn: sp.tokenIn, tokenOut: sp.tokenOut, fee: sp.fee, recipient: sp.recipient,

deadline: sp.deadline, amountIn: sp.amountIn, amountOutMinimum: sp.minOut, sqrtPriceLimitX96: sp.sqrtPriceLimitX96

});

uint256 received = router.exactInputSingle(params);

return Types.AdapterResult(true, sp.amountIn, received, g0 - gasleft(), "");

}

}

B. Deployment scripts (Foundry, script/)

script/Deploy.s.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

import "forge-std/Script.sol";

import {PauseGuardian} from "../src/policy/PauseGuardian.sol";

import {PolicyGuards} from "../src/policy/PolicyGuards.sol";

import {RouteRegistry} from "../src/policy/RouteRegistry.sol";

import {SettlementVault} from "../src/settlement/SettlementVault.sol";

import {BundleExecutor} from "../src/execution/BundleExecutor.sol";

import {UniswapV3Adapter} from "../src/venues/adapters/UniswapV3Adapter.sol";

interface IQuoterV2 { }

interface ISwapRouter { }

contract Deploy is Script {

function run() external {

address owner = vm.envAddress("OWNER");

address guardian = vm.envAddress("GUARDIAN");

address router = vm.envAddress("UNIV3\_ROUTER");

address quoter = vm.envAddress("UNIV3\_QUOTER");

vm.startBroadcast();

PauseGuardian pg = new PauseGuardian(owner, guardian);

PolicyGuards pol = new PolicyGuards(owner);

RouteRegistry rr = new RouteRegistry();

SettlementVault sv = new SettlementVault(owner);

BundleExecutor ex = new BundleExecutor(rr, pol, pg, sv);

UniswapV3Adapter ua = new UniswapV3Adapter(ISwapRouter(router), IQuoterV2(quoter));

sv.setExecutor(address(ex));

vm.stopBroadcast();

console2.log("PauseGuardian", address(pg));

console2.log("PolicyGuards ", address(pol));

console2.log("RouteRegistry", address(rr));

console2.log("SettlementVault", address(sv));

console2.log("BundleExecutor", address(ex));

console2.log("UniswapV3Adapter", address(ua));

}

}

script/Configure.s.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

import "forge-std/Script.sol";

import {PolicyGuards} from "../src/policy/PolicyGuards.sol";

contract Configure is Script {

function run() external {

address guards = vm.envAddress("GUARDS");

address univ3 = vm.envAddress("UNIV3\_ADAPTER");

address noop = vm.envAddress("NOOP\_ADAPTER");

vm.startBroadcast();

PolicyGuards(guards).setAllowedAdapter(univ3, true);

PolicyGuards(guards).setAllowedAdapter(noop, true);

vm.stopBroadcast();

}

}

C. Foundry tests — skeletons you can extend

test/policy/RouteRegistry.t.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

import "forge-std/Test.sol";

import "../../src/policy/RouteRegistry.sol";

import "../../src/common/Types.sol";

contract RouteRegistryTest is Test {

RouteRegistry rr;

function setUp() public { rr = new RouteRegistry(); }

function test\_registerUpdateDisable() public {

Types.Step;

steps[0] = Types.Step(Types.StepKind.NOOP, address(0xBEEF), "", address(1), address(2), 1e18, 1);

bytes32 salt = keccak256("salt");

bytes32 id = rr.registerRoute(steps, salt);

assertEq(rr.routes(id).owner, address(this));

steps[0].minAmountOut = 2;

rr.updateRoute(id, steps);

assertTrue(rr.routes(id).owner == address(this));

rr.setDisabled(id, true);

assertTrue(rr.routes(id).disabled());

}

}

test/execution/BundleExecutor.unit.t.sol (uses a stub adapter or NoOp)

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

import "forge-std/Test.sol";

import "../../src/execution/BundleExecutor.sol";

import "../../src/policy/RouteRegistry.sol";

import "../../src/policy/PolicyGuards.sol";

import "../../src/policy/PauseGuardian.sol";

import "../../src/settlement/SettlementVault.sol";

import "../../src/common/Types.sol";

contract BundleExecutorUnit is Test {

RouteRegistry rr; PolicyGuards pol; PauseGuardian pg; SettlementVault sv; BundleExecutor ex;

function setUp() public {

rr = new RouteRegistry(); pol = new PolicyGuards(address(this)); pg = new PauseGuardian(address(this), address(this));

sv = new SettlementVault(address(this)); ex = new BundleExecutor(rr, pol, pg, sv);

}

function test\_rejectWhenNoSteps() public {

vm.expectRevert();

ex.execute(bytes32(uint256(1)), 0, 0, address(1), address(2), address(this));

}

}

Add Uniswap fork/integration tests similarly (omitted here for brevity).

D. TypeScript domain & clients — scaffolds

D1. File tree (TS)

packages/

core-exec/src/BundleTypes.ts

core-exec/src/encode.ts

core-exec/src/decoders.ts

core-exec/src/policyClient.ts

core-exec/src/executorClient.ts

core-exec/src/vaultClient.ts

core-exec/src/RouteBuilder.ts

adapters-evm/src/UniswapV3AdapterClient.ts

cli/src/commands/route.ts

D2. Types

packages/core-exec/src/BundleTypes.ts

export enum StepKind { NOOP = 0, UNIV3\_SWAP = 1 }

export type Step = {

kind: StepKind;

adapter: `0x${string}`;

payload: `0x${string}`;

tokenIn: `0x${string}`;

tokenOut: `0x${string}`;

amountIn: bigint;

minAmountOut: bigint;

};

export type RouteSpec = { steps: Step[]; salt: `0x${string}` };

D3. Encoders

packages/core-exec/src/encode.ts

import { encodeAbiParameters, parseAbiParameters } from "viem";

export type UniV3SwapParams = {

tokenIn: `0x${string}`; tokenOut: `0x${string}`;

fee: number; recipient: `0x${string}`;

amountIn: bigint; minOut: bigint; sqrtPriceLimitX96: bigint; deadline: bigint;

};

export const encodeUniV3SwapParams = (p: UniV3SwapParams): `0x${string}` =>

encodeAbiParameters(

parseAbiParameters("address,address,uint24,address,uint256,uint256,uint160,uint256"),

[p.tokenIn, p.tokenOut, p.fee, p.recipient, p.amountIn, p.minOut, p.sqrtPriceLimitX96, p.deadline]

) as `0x${string}`;

D4. Adapter client

packages/adapters-evm/src/UniswapV3AdapterClient.ts

import { StepKind, Step } from "@project/core-exec/src/BundleTypes";

import { encodeUniV3SwapParams, UniV3SwapParams } from "@project/core-exec/src/encode";

export function buildSingleHop(params: UniV3SwapParams, adapter: `0x${string}`): Step {

return {

kind: StepKind.UNIV3\_SWAP,

adapter,

payload: encodeUniV3SwapParams(params),

tokenIn: params.tokenIn,

tokenOut: params.tokenOut,

amountIn: params.amountIn,

minAmountOut: params.minOut,

};

}

D5. Policy & executor clients

packages/core-exec/src/policyClient.ts

import { PublicClient, WalletClient, getAbiItem, encodeFunctionData } from "viem";

import routeRegistryAbi from "./abi/RouteRegistry.json";

import policyAbi from "./abi/PolicyGuards.json";

export class PolicyClient {

constructor(private pub: PublicClient, private wallet: WalletClient, private addresses: { registry: `0x${string}`, guards: `0x${string}` }){}

async registerRoute(owner: `0x${string}`, stepsEncoded: `0x${string}`[], salt: `0x${string}`) {

// Example: call registerRoute with ABI encoding of Step[]; you’ll need to mirror ABI for FULL payload

return this.wallet.writeContract({

address: this.addresses.registry, abi: routeRegistryAbi, functionName: "registerRoute",

args: [/\* Types.Step[] \*/, salt]

});

}

}

packages/core-exec/src/executorClient.ts

import executorAbi from "./abi/BundleExecutor.json";

import { WalletClient } from "viem";

export class ExecutorClient {

constructor(private wallet: WalletClient, private address: `0x${string}` ) {}

execute(args: {

routeId: `0x${string}`; amountIn: bigint; minTotalOut: bigint;

tokenIn: `0x${string}`; tokenOut: `0x${string}`; recipient: `0x${string}`;

}) {

return this.wallet.writeContract({

address: this.address, abi: executorAbi, functionName: "execute",

args: [args.routeId, args.amountIn, args.minTotalOut, args.tokenIn, args.tokenOut, args.recipient],

});

}

}

D6. Route builder

packages/core-exec/src/RouteBuilder.ts

import { Step, RouteSpec } from "./BundleTypes";

import { keccak256, encodeAbiParameters, parseAbiParameters } from "viem";

export function buildRoute(steps: Step[], salt: `0x${string}`): RouteSpec { return { steps, salt }; }

export function computeRouteId(spec: RouteSpec, owner: `0x${string}`): `0x${string}` {

// Must mirror Solidity: keccak256(abi.encode(steps, owner, salt))

// For MVP you may ABI-encode each Step in the same order as Solidity struct.

const encoded = encodeAbiParameters(

parseAbiParameters("(uint8,address,bytes,address,address,uint256,uint256)[] steps,address owner,bytes32 salt"),

[spec.steps.map(s => [s.kind, s.adapter, s.payload, s.tokenIn, s.tokenOut, s.amountIn, s.minAmountOut]), owner, spec.salt]

);

return keccak256(encoded);

}

E. CLI — command skeleton

packages/cli/src/commands/route.ts

import { Command } from "commander";

import { buildRoute, computeRouteId } from "@project/core-exec/src/RouteBuilder";

import { buildSingleHop } from "@project/adapters-evm/src/UniswapV3AdapterClient";

export const routeCmd = new Command("route");

routeCmd

.command("build-singlehop")

.requiredOption("--adapter <addr>")

.requiredOption("--token-in <addr>")

.requiredOption("--token-out <addr>")

.requiredOption("--fee <number>")

.requiredOption("--amount-in <wei>")

.requiredOption("--min-out <wei>")

.requiredOption("--recipient <addr>")

.requiredOption("--deadline <ts>")

.option("--sqrt-limit <x96>", "0")

.requiredOption("--salt <bytes32>")

.action(async (opts) => {

const step = buildSingleHop({

tokenIn: opts.tokenIn, tokenOut: opts.tokenOut, fee: Number(opts.fee),

amountIn: BigInt(opts["amount-in"]), minOut: BigInt(opts["min-out"]),

recipient: opts.recipient, deadline: BigInt(opts.deadline), sqrtPriceLimitX96: BigInt(opts["sqrt-limit"] || 0)

}, opts.adapter as `0x${string}`);

const spec = buildRoute([step], opts.salt as `0x${string}`);

console.log(JSON.stringify(spec, null, 2));

});

export default routeCmd;

Wire this into your CLI index and package.json scripts.

F. Config — minimal entries

config/venues.json

{

"sepolia": {

"uniswapV3": {

"router": "0x<router>",

"quoter": "0x<quoter>"

}

}

}

config/policies.json

{

"allowedAdapters": ["0x<uniswapV3Adapter>","0x<noopAdapter>"],

"slippageDefaultBps": 50

}

G. E2E script — Sepolia demo

packages/core-exec/src/run/sepoliaExecute.ts

import { publicClient, walletClient } from "../util/clients"; // your viem setup

import { ExecutorClient } from "../executorClient";

import { buildSingleHop } from "@project/adapters-evm/src/UniswapV3AdapterClient";

import { buildRoute, computeRouteId } from "../RouteBuilder";

async function main() {

const cfg = require("../../../config/venues.json").sepolia.uniswapV3;

const addresses = require("../../../deployments/sepolia.json");

const owner = (await walletClient.getAddresses())[0];

const step = buildSingleHop({

tokenIn: "0x<tokenIn>", tokenOut: "0x<tokenOut>", fee: 3000,

amountIn: 10n \*\* 18n, minOut: 0n, recipient: owner, deadline: BigInt(Math.floor(Date.now()/1000)+1800), sqrtPriceLimitX96: 0n

}, addresses.UniswapV3Adapter as `0x${string}`);

const spec = buildRoute([step], "0x0000000000000000000000000000000000000000000000000000000000000042");

const routeId = computeRouteId(spec, owner);

console.log({ routeId });

const ex = new ExecutorClient(walletClient, addresses.BundleExecutor as `0x${string}`);

const tx = await ex.execute({

routeId, amountIn: step.amountIn, minTotalOut: step.minAmountOut,

tokenIn: step.tokenIn, tokenOut: step.tokenOut, recipient: owner

});

console.log("execute tx", tx);

}

main().catch((e)=>{ console.error(e); process.exit(1); });

H. Tests — TS (Vitest) skeletons

packages/core-exec/test/encode.spec.ts

import { describe, it, expect } from "vitest";

import { encodeUniV3SwapParams } from "../src/encode";

describe("encode uni v3", () => {

it("encodes deterministically", () => {

const hex = encodeUniV3SwapParams({

tokenIn: "0x0000000000000000000000000000000000000001",

tokenOut:"0x0000000000000000000000000000000000000002",

fee: 3000, recipient:"0x0000000000000000000000000000000000000003",

amountIn: 123n, minOut: 100n, sqrtPriceLimitX96: 0n, deadline: 9999999999n

});

expect(hex.startsWith("0x")).toBe(true);

});

});

packages/core-exec/test/routeBuilder.multistep.spec.ts

import { describe, it, expect } from "vitest";

import { buildRoute, computeRouteId } from "../src/RouteBuilder";

describe("route builder", () => {

it("computes deterministic id", () => {

const spec = buildRoute([], "0x"+ "00".repeat(32) as `0x${string}`);

const id = computeRouteId(spec, "0x0000000000000000000000000000000000000001");

expect(id.startsWith("0x")).toBe(true);

});

});

I. CI additions (concrete)

Add jobs to your existing workflow:

- name: Foundry unit

run: forge test --match-path contracts/test/\*\* -vv

- name: TypeScript unit

run: pnpm -w -r run test

- name: Lint & typecheck

run: pnpm -w -r run lint && pnpm -w -r run typecheck

J. Manual E2E checklist (one sitting)

Anvil: anvil --fork-url $SEPOLIA\_RPC

Deploy: forge script contracts/script/Deploy.s.sol --rpc-url $RPC --broadcast --legacy -vvvv

Record addresses to deployments/sepolia.json.

Configure allowlist: forge script contracts/script/Configure.s.sol --rpc-url $RPC --broadcast

Fund deployer & vault tokens as needed.

CLI build route:

pnpm cli route build-singlehop \

--adapter 0x... --token-in 0x... --token-out 0x... \

--fee 3000 --amount-in 1000000000000000000 \

--min-out 0 --recipient 0xYOU --deadline $(date -v+30M +%s) \

--salt 0x... > /tmp/route.json

Run script: pnpm -w demo:singlehop.

Verify balances and BundleExecuted logs.

K. Known limitations (explicit, so agents don’t overbuild in M1)

ERC20 only (no native unwrap).

exactInputSingle only (path-based exactInput reserved for M2).

No on-chain slippage oracle; CLI sets minTotalOut.

Owners are EOAs (multisig in M2).

Telemetry is console-only (metrics backends in later milestones).