Bijlage C: Het Minimum Viable Product (MVP) Canvas

Visie: Het "Walking Skeleton" bouwen: de kleinst mogelijke, end-to-end functionerende versie van de S1mpleTrader V2 applicatie die de kern van de event-gedreven architectuur valideert.

1. MVP Definitie & Scope

Wat is het MVP?

> Een **samenwerking van Service-laag componenten** die, in reactie op events op een simpele EventBus, de **herbruikbare Backend-laag** aansturen om **één enkele strategie**, end-to-end, in een **backtest-omgeving** uit te voeren en de resultaten te rapporteren via de **command-line**.

Focus: De Event-Gedreven Kernflow

De focus ligt 100% op het valideren van de fundamentele, event-gedreven datastroom voor een enkele, geïsoleerde run. We beantwoorden de vraag: "Kunnen onze gespecialiseerde componenten via events met elkaar praten om een brok data te transformeren naar een eindresultaat?"

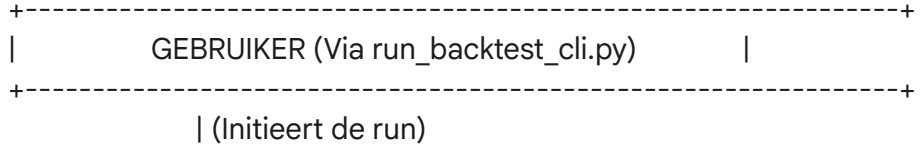
Bewust Buiten Scope (Post-MVP)

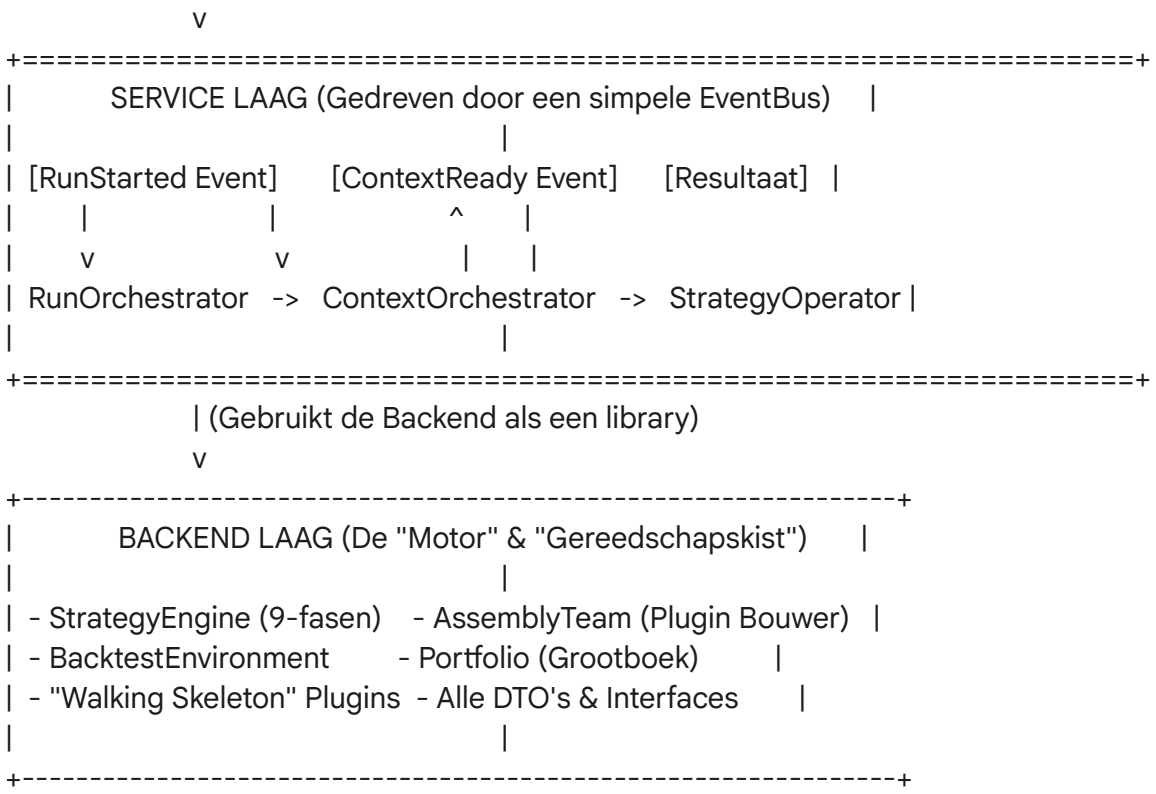
- **PortfolioSupervisor**: Er is geen overkoepelend beheer van meerdere strategieën.
- **Web UI / API's**: De interactie gebeurt uitsluitend via een command-line entrypoint.
- **Live / Paper Trading**: Enkel de BacktestEnvironment wordt geïmplementeerd.
- **Geavanceerde Meta Workflows**: Geen OptimizationService of VariantTestService.
- **Persistentie & Robuustheid**: Geen state-saving of crash-recovery.

2. MVP Architectuur: De Eenzame Specialist-Workflow

Het MVP implementeert de "happy path" workflow voor één enkele run. Een run_blueprint.yaml wordt ingeladen, wat een keten van events triggert die de Backend-componenten als een gereedschapskist gebruiken.

Visueel Diagram van de MVP-Flow:





3. Componenten Breakdown

De Fundering (Bestaand & Herbruikbaar - Backend Laag)

De volledige Backend-laag is in essentie voltooid en wordt als een "gereedschapskist" gebruikt door de Service-laag.

- **De Analytische Motor (StrategyEngine)**: De stateless, procedurele 9-fasen pijplijn die de kernlogica van een strategie uitvoert, is geïmplementeerd en getest.
- **Het Grootboek (Portfolio)**: De "domme" boekhouder die de financiële staat van de run beheert, is robuust en getest.
- **De Wereld (BacktestEnvironment)**: De omgeving voor het uitvoeren van een strategie tegen historische CSV-data is operationeel.
- **De Bouwploeg (AssemblyTeam)**: De specialisten voor het ontdekken, bouwen en voorbereiden van plugins zijn functioneel.
- **De Contracten (DTO's & Interfaces)**: De volledige set van Pydantic data-contracten die de communicatie regelt, is scherp gedefinieerd.

De Lijm & Brandstof (Te Bouwen - Service Laag & Overig)

De volgende componenten vormen de kern van de MVP-implementatie.

- **Service Laag "Glue Code"**: De minimale, script-achtige implementatie van de kern-services (RunOrchestrator, ContextOrchestrator, StrategyOperator) die nodig zijn om de event-gedreven flow voor een enkele backtest te faciliteren. Ze fungeren als de "lijm" tussen de EventBus en de Backend-gereedschappen. Voor het MVP zal de `StrategyOperator` zijn eigen voorstellen direct doorzetten naar de `ExecutionHandler` zonder een aparte goedkeuringsstap, om de kern-flow te valideren.
- **Simpele EventBus**: Een basis in-memory event-systeem dat publish en subscribe functionaliteit biedt voor de duur van de run.
- **"Walking Skeleton" Plugin Set**: Een minimale set van functionerende plugins om de StrategyEngine van een daadwerkelijke, zij het simpele, logica te voorzien. Dit is de "brandstof" voor de motor.
- **CLI Entrypoint (run_backtest_cli.py)**: De "startknop" waarmee de gebruiker vanaf de command-line een backtest kan initiëren door een run_blueprint.yaml mee te geven.
- **Rapportage Specialisten**: Simpele TickReporter & ResultReporter klassen die, aangestuurd door de Service-laag, de voortgang en het eindresultaat naar de console en een CSV-bestand kunnen schrijven.

4. MVP Product Backlog (User Stories)

Hieronder volgt de backlog, geherstructureerd rondom de nieuwe architectuur, die de concrete taken definieert om het MVP te realiseren.

| Categorie | User Story | Component(en) | Type |
| SERVICE LAAG & EVENTS |
Als ontwikkelaar wil ik een RunOrchestrator die luistert naar een CLI-trigger en een RunStarted event publiceert. | RunOrchestrator | BE |
| |
Als ontwikkelaar wil ik een ContextOrchestrator die op RunStarted reageert, de BacktestEnvironment & ContextBuilder aanroept, en een ContextReady event publiceert voor elke tick. | ContextOrchestrator, AssemblyTeam | BE |
| |
Als ontwikkelaar wil ik een StrategyOperator die op ContextReady reageert, de StrategyEngine.run() methode aanroept, en het EngineCycleResult verwerkt. | StrategyOperator, StrategyEngine | BE |
| |
Als ontwikkelaar wil ik een ExecutionHandler (geïntegreerd in de StrategyOperator voor het MVP) die ExecutionDirectives verwerkt en het Portfolio-object muteert. | StrategyOperator, Portfolio | BE |
| PLUGIN DEVELOPMENT |
Ontwikkel een simpele EMADetector plugin (type: structural_context) die een EMA-kolom toevoegt. | Plugin: EMADetector | DEV |
| |
Ontwikkel een simpele CrossOverSignal plugin (type: signal_generator) die een Signal DTO

genereert bij een EMA crossover. | Plugin: CrossOverSignal | DEV |
| |
Ontwikkel een "pass-through" FixedRiskExitPlanner (type: exit_planner) die een vaste stop-loss en take-profit berekent. | Plugin: FixedRiskExitPlanner | DEV |
| |
Schrijf de verplichte unit tests voor alle "Walking Skeleton" plugins om hun correcte werking te valideren. | pytest | DEV |
| ENTRYPOINT (CLI) |
Implementeer run_backtest_cli.py om een run_blueprint.yaml in te lezen en de initiële trigger voor de RunOrchestrator te geven. | run_backtest_cli.py | BE |
| RAPPORTAGE |
Bouw een simpele ConsoleTickReporter die zich abonneert op ContextReady en de P&L per tick print. | ConsoleTickReporter | BE |
| |
Bouw een CsvResultReporter die, na afloop van de run, de ClosedTrades uit het Portfolio naar een bestand schrijft. | CsvResultReporter, Portfolio | BE |
| END-TO-END VALIDATIE |
Schrijf een integratietest die run_backtest_cli aanroept met een vaste run.yaml en valideert of de output CSV exact overeenkomt met een verwacht resultaat. | pytest | TEST |