Project Plan & Specification

Clarifying Questions

Before implementing this telemetry system, I would ask the researcher these critical questions:

1. Should we anonymise sensitive details (paths, credentials, PII in commands) automatically, or leave this responsibility to the researcher?

2. Should developer “dead ends” (failed builds/tests, incorrect commands, abandoned edits) be included, or do you only want successful solution paths?

3. Should we store full logs/diffs or only summarised/truncated versions (with option to fetch full artifacts)?

4. How much “chain-of-thought” should be made explicit? Should we prompt developers to write down reasoning steps periodically, or only passively log their actions?

5. Should the system be fully passive (record everything silently), or should we inject active prompts (“Why did you try this fix?”) to enrich reasoning data?

Assumptions Made

1. For MVP, we won't focus on redaction, but if need be, we can run a small NLP model to detect PII in commands/paths.
2. We will capture all events, including failed attempts, as they provide valuable context on the problem-solving process.
3. We will store full logs and diffs, but provide options to summarise or truncate for quick viewing.
4. We will passively log actions, but provide an optional feature for developers to add explicit thoughts.
5. The system will be primarily passive, but we can consider active prompts in future iterations.

Proposed Data Schema

The telemetry trace follows this comprehensive JSON schema:

```
{
  "trace_id": "uuid",
  "created_at": "2024-01-15T10:30:00Z",
  "version": "trace-1.0.0",
  "developer": {
    "name": "string",
    "email": "string",
    "experience_level": "junior|mid|senior",
    "ide": "vscode|intellij|vim|emacs",
    "os": "windows|macos|linux"
  },
  "task": {
    "description": "string",
    "repository": "https://github.com/org/repo",
    "branch": "main",
    "commit": "abc123",
    "test_image": "golang:1.24",
    "test_command": "go test ./...",
    "complexity": "low|medium|high"
  },
  "environment": {
    "os": "linux",
    "editor": "vscode",
    "version": "1.0.0",
    "plugins": ["go", "gitlens"],
    "workspace_size": 42
  },
  "events": [
    {
      "t": "2024-01-15T10:30:15Z",
      "type": "file_opened",
      "session_id": "session-1",
      "file_path": "src/main.go",
      "language": "go",
    },
    {
      "t": "2024-01-15T10:30:20Z",
      "type": "go_to_definition",
      "session_id": "session-1",
      "source": {
        "file_path": "src/main.go",
        "symbol": "calculateTotal",
        "range": {"start": {"line": 10, "col": 5}, "end": {"line": 10, "col": 20}}
      },
      "target": {
        "file_path": "src/calculator.go",
        "line": 25,
        "symbol_kind": "function"
      },
      "latency_ms": 150
    },
    {
      "t": "2024-01-15T10:30:25Z",
      "type": "edit",
      "session_id": "session-1",
      "file_path": "src/calculator.go",
      "op": "replace",
      "range": {"start": {"line": 25, "col": 1}, "end": {"line": 30, "col": 1}},
      "patch_unified": "diff --git a/calculator.go b/calculator.go\n...",
      "before_hash": "abc123",
      "after_hash": "def456"
    },
    {
      "t": "2024-01-15T10:30:30Z",
      "type": "terminal_command",
      "session_id": "session-1",
      "cmd": "go test",
      "args": ["-v", "./..."],
      "cwd": "/workspace",
      "exit_code": 0,
      "duration_ms": 2500,
      "stdout": "PASS: TestCalculator\n",
      "stderr": ""
    },
    {
      "t": "2024-01-15T10:30:35Z",
      "type": "thought",
      "session_id": "session-1",
      "raw": "I need to fix the edge case in the calculation",
      "tags": ["debugging", "edge-case"]
    }
  ],
  "artifacts": {
    "final_patch": "unified diff string",
    "test_results": {
      "passed": 5,
      "failed": 0,
      "duration_ms": 2500
    }
  },
  "qa": {
    "tests": {
      "runner": "docker",
      "image": "golang:1.24",
      "command": "go test ./...",
      "ok": true,
      "stdout": "PASS: TestCalculator\n",
      "stderr": "",
      "exit_code": 0
    },
    "judge": {
      "model": "gpt-4",
      "rubric_version": "v1",
      "scores": {
        "problem_understanding": 4.0,
        "investigation_strategy": 4.0,
        "decision_rationale": 4.0,
        "use_of_evidence": 4.0,
        "reproducibility": 4.0,
        "safety_privacy_awareness": 4.0
      },
      "overall": 4.0,
      "comments": "Excellent problem-solving approach"
    }
  }
}
```

Schema Design Justifications:

- **Structured Events**: Each event type has specific fields relevant to its context, enabling rich analysis
- **Temporal Ordering**: All events include precise timestamps for sequence analysis
- **Session Tracking**: `session_id` groups related events within a coding session
- **Rich Context**: File paths, symbols, ranges provide detailed context for each action
- **Performance Metrics**: Latency, duration, and resource usage enable performance analysis
- **Thought Capture**: Developer reasoning and mental model capture via thought events
- **Patch Integration**: Unified diff format enables precise code change analysis
- **Extensible Design**: Version field and flexible JSON structure allow schema evolution

High-Level Technical Plan

Proposed Tech Stack:

- **API Layer**: Go with Chi router
- **Database**: PostgreSQL with JSONB for flexible event storage
- **Message Queue**: Redis with Asynq for job processing
- **Object Storage**: MinIO (S3-compatible) for large event batches
- **Container Runtime**: Docker-in-Docker for isolated QA testing
- **Monitoring**: Built-in health checks and structured logging

Stack Justification:

- **Go**: Excellent performance, built-in concurrency, strong typing for data processing
- **PostgreSQL**: ACID compliance, JSONB for flexible schemas, excellent query performance
- **Redis**: Fast in-memory operations, reliable job queuing with Asynq
- **MinIO**: S3-compatible API, easy local development, cost-effective
- **Docker**: Industry standard for containerization, excellent isolation for testing

Scope & Trade-offs

MVP Features (Essential):

- □ Basic event ingestion (file ops, edits, commands)
- □ Event storage and retrieval
- □ Code patch extraction and application
- □ Automated test execution in Docker
- □ Basic QA result storage
- □ REST API for trace management
- □ Authentication and security

Phase 2 Features (Important):

- □ Advanced event types (go-to-definition, find-references)
- □ Thought event capture and processing
- □ AI-powered code quality judgment
- □ Event filtering and search capabilities
- □ Performance metrics and analytics

De-scoped Features (Nice-to-have):

- □ Multi-language IDE support beyond Go
- □ Advanced privacy controls and data anonymization

Key Trade-offs Made:

1. **Simplicity vs. Features**: Focused on core functionality over advanced analytics
2. **Performance vs. Flexibility**: Used JSONB for flexibility over optimized relational design
3. **Local Development vs. Production**: Optimized for easy local setup with Docker Compose