# A blue and white logo AI-generated content may be incorrect.

# Architecture Documentation

## SPC Station Health Charts

**Version:** 1.0

**Last Updated:** October 6, 2025

**Architecture Style:** Client-Server (Local)

## 1. System Overview

### High-Level Architecture

┌─────────────────────────────────────────────────────────────┐

│ User's Desktop │

│ │

│ ┌──────────┐ ┌──────────────────┐ │

│ │ User │────Double-Click───→│ Batch Launcher │ │

│ └──────────┘ └──────────────────┘ │

│ ↓ ↓ │

│ │ ┌──────────────────┐ │

│ │ │ Python Server │ │

│ │ │ (localhost:8000)│ │

│ │ └──────────────────┘ │

│ │ ↓ │

│ │ ┌──────────────────┐ │

│ └─────Opens Browser───────→│ Web Dashboard │ │

│ │ (HTML/Canvas) │ │

│ └──────────────────┘ │

│ ↓ │

│ ┌──────────────────┐ │

│ │ SPC Processor │ │

│ │ (Statistics) │ │

│ └──────────────────┘ │

│ ↓ │

│ ┌──────────────────┐ │

│ │ CSV Files │ │

│ │ (input/) │ │

│ └──────────────────┘ │

└─────────────────────────────────────────────────────────────┘

## 2. Component Architecture

### 2.1 Frontend (dashboard\_standalone.html)

**Technology:** Pure HTML5 + JavaScript (No frameworks)

**Key Responsibilities:**

* User interface rendering
* Chart visualization via HTML5 Canvas
* API communication with Python backend
* PNG export via Canvas toDataURL()
* Client-side state management

**Design Patterns:**

* **Event-Driven:** Button clicks trigger async API calls
* **Data-Driven Rendering:** Charts redraw on state change
* **Functional Decomposition:** Separate functions for each chart type

### 2.2 Backend (Python HTTP Server)

**server.py - HTTP Request Handler**

* Extends SimpleHTTPRequestHandler
* Routes requests to appropriate handlers
* Adds CORS headers for local development
* Serves static HTML file
* Processes API requests

**spc\_processor.py - Statistical Engine**

* Parses CSV data
* Calculates control limits (2.66 sigma)
* Detects phases using Wheeler's Rules
* Generates both X and mR charts
* Returns structured JSON

**load\_actual\_data.py - Data Loader**

* Scans input/ directory for CSV files
* Converts format to standardized structure
* Maps station names (e.g., "Dallas" → "DAL")
* Infers measure names from filenames
* Combines multiple files into single dataset

## 3. Data Flow Architecture

### 3.1 Complete Data Flow (Load Test Data)

User clicks "Load Test Data"

↓

Browser sends POST /api/load-actual

↓

Server calls load\_actual\_data.convert\_to\_spc\_format()

↓

Loader reads all CSV files from input/

↓

Loader converts to standard format

↓

Server calls spc\_processor.process\_data()

↓

SPC Processor:

- Parses CSV

- Groups by station/measure

- For each measure:

\* detect\_phases (X chart)

\* generate\_moving\_range\_data

\* detect\_phases (mR chart)

↓

Returns JSON {chartData, stations}

↓

Browser receives JSON response

↓

Browser renders charts on Canvas

↓

User views 24 charts

### 3.2 Phase Detection Algorithm Flow

Start: Raw Data Points

↓

Sort by Date

↓

Initialize: phase\_number=1, current\_start=0

↓

┌──────────────────────┐

│ More data to process?│──No──→ Return phases + augmented points

└──────────────────────┘

↓ Yes

Take next 20 points as baseline

↓

Calculate limits from baseline ONLY: CL, UCL, LCL

↓

Monitor remaining data for Wheeler's Rules

↓

┌────────────────────────┐

│ Point outside UCL/LCL? │──Yes──→ Signal Detected!

└────────────────────────┘ ↓

↓ No End current phase

┌────────────────────────────┐ ↓

│ 8 consecutive on one side? │──Yes──→ Recalculate limits

└────────────────────────────┘ ↓

↓ No Save phase metadata

Continue Phase ↓

↓ Increment phase\_number

Back to "More data to process?" ↓

current\_start = phase\_end + 1

## 4. Technology Stack

### 4.1 Technology Decisions

|  |  |  |
| --- | --- | --- |
| **Layer** | **Technology** | **Rationale** |
| **Frontend UI** | HTML5 + CSS | Universal browser support; no build step |
| **Frontend Logic** | Vanilla JavaScript | Zero dependencies; works offline |
| **Charting** | HTML5 Canvas API | No external libraries; full control; PNG export built-in |
| **Backend** | Python 3.7+ | Already installed enterprise-wide; no admin rights needed |
| **Web Server** | http.server (stdlib) | No installation required; sufficient for localhost |
| **Data Format** | CSV | Universal; easy to generate from any system |
| **Data Transfer** | JSON over HTTP | Standard web protocol; easy parsing |
| **State Management** | Global JS variables | Simple; no framework overhead |

### Critical Design Decision

**❌ Rejected:** React, Recharts, npm, Node.js

**Reason:** External dependencies, CDN issues, corporate firewall blocks

**✅ Chosen:** Vanilla JavaScript + Canvas

**Reason:** Zero dependencies, works anywhere, full control

## 5. API Specification

### 5.1 Endpoint: GET /

**Purpose:** Serve main dashboard HTML

|  |  |
| --- | --- |
| **Request** | **Response** |
| GET / HTTP/1.1 Host: localhost:8000 | HTTP/1.1 200 OK Content-Type: text/html  <!DOCTYPE html>... |

### 5.2 Endpoint: POST /api/load-actual

**Purpose:** Load sample data from input/ folder

|  |  |
| --- | --- |
| **Request** | **Response** |
| POST /api/load-actual HTTP/1.1 Host: localhost:8000 Content-Type: application/json  {} | HTTP/1.1 200 OK Content-Type: application/json  {"success": true, "chartData": {...}, "stations": [...]} |

### 5.3 Endpoint: POST /api/process

**Purpose:** Process uploaded CSV file

|  |  |
| --- | --- |
| **Request** | **Response** |
| POST /api/process HTTP/1.1 Host: localhost:8000 Content-Type: application/json  {"csvData": "...", "filename": "..."} | HTTP/1.1 200 OK Content-Type: application/json  {"success": true, "chartData": {...}, "stations": [...]} |

## 6. Security Architecture

### 6.1 Security Model

**Threat Model:** Low risk (local desktop tool)

**Security Decisions:**

* **No authentication:** Single-user desktop tool; localhost only
* **No HTTPS:** Traffic never leaves local machine
* **Minimal input validation:** Trust user-provided CSVs (graceful error handling)
* **No CORS restrictions:** Local origin only

|  |  |  |
| --- | --- | --- |
| **Threat** | **Status** | **Mitigation/Acceptance** |
| CSV Injection | Accepted | Not applicable (no Excel output) |
| Path Traversal | Mitigated | Only read file content; no path access |
| XSS | Mitigated | Canvas rendering (not innerHTML) |
| DoS (Large CSV) | Accepted | User's own machine; low risk |

## 7. Deployment Architecture

### 7.1 Deployment Model

GitHub Repository

↓

Distribution ZIP

↓

┌─────────────┬─────────────┬─────────────┐

│ User 1 │ User 2 │ User 3 │

│ Desktop │ Desktop │ Desktop │

└─────────────┴─────────────┴─────────────┘

↓ ↓ ↓

┌─────────────┬─────────────┬─────────────┐

│ Python │ Python │ Python │

│ Server │ Server │ Server │

│ Port 8000 │ Port 8000 │ Port 8000 │

└─────────────┴─────────────┴─────────────┘

No central server - each user runs independent instance

### 7.2 Deployment Steps

1. Download ZIP from GitHub
2. Extract to any folder
3. Double-click START\_DASHBOARD.bat (Windows) or ./start\_dashboard.sh (Mac/Linux)
4. Browser opens automatically to http://localhost:8000

### 7.3 File Structure

SPC-Station-Health-Charts/

│

├── dashboard\_standalone.html # Main UI (self-contained)

├── server.py # HTTP server + routing

├── spc\_processor.py # Statistical calculations

├── load\_actual\_data.py # CSV data loader

│

├── START\_DASHBOARD.bat # Windows launcher

├── start\_dashboard.sh # Mac/Linux launcher

│

├── README.md # Technical README

├── README\_DISTRIBUTION.md # User-facing README

├── .gitignore # Git ignore rules

│

├── input/ # Sample CSV data

│ ├── maintenance\_cancels.csv

│ ├── maintenance\_delays.csv

│ ├── scheduled\_maintenance\_findings.csv

│ └── unscheduled\_maintenance.csv

│

└── output/ # Exported PNG charts

## 8. Performance Architecture

### 8.1 Performance Characteristics

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Target** | **Actual** | **Notes** |
| Server startup | < 2s | ~1s | Python startup + socket bind |
| Page load | < 3s | ~1s | Single HTML file, no bundling |
| Data processing (10K rows) | < 2s | ~0.5s | Pure Python, no database |
| Chart rendering (8 charts) | < 2s | ~1s | Canvas API, no DOM manipulation |
| PNG export | Instant | ~100ms | Canvas toDataURL() |

### 8.2 Scalability Limits

**Design Constraints:**

* **Data size:** Tested up to 10,000 rows
* **Chart count:** Tested up to 24 charts per view
* **Concurrent users:** Single user per instance (localhost only)
* **Memory:** ~50MB Python process + ~100MB browser tab

**Horizontal Scaling:** Each user runs their own instance (inherently scalable)

## 9. Architecture Decision Records (ADRs)

### ADR-001: Use HTML5 Canvas Instead of Recharts

**Context:** Initial implementation used Recharts library loaded from CDN

**Decision:** Replaced with native HTML5 Canvas API

**Rationale:**

* Corporate firewall blocks CDN access
* Recharts requires React framework (additional dependency)
* Canvas API provides sufficient charting capability
* Full control over rendering and export

**Consequences:**

* (+) Zero external dependencies
* (+) Works in any restricted environment
* (+) PNG export built-in via toDataURL()
* (-) More code to maintain (manual chart drawing)

**Status:** ✅ Accepted

### ADR-002: Python http.server Instead of Flask/FastAPI

**Context:** Need web server for local dashboard

**Decision:** Use Python's built-in http.server module

**Rationale:**

* Already included in Python standard library
* No pip install required (users lack admin rights)
* Sufficient for localhost single-user scenario
* Simple to understand and modify

**Consequences:**

* (+) Zero installation friction
* (+) Works immediately after extracting ZIP
* (-) Not suitable for production multi-user deployment

**Status:** ✅ Accepted

### ADR-003: Calculate Limits from Baseline, Not Entire Dataset

**Context:** Initial implementation calculated limits from all data, causing erratic behavior

**Decision:** Calculate limits from first 20 points (baseline), then monitor for shifts

**Rationale:**

* Follows Wheeler's methodology exactly
* Prevents limits from being influenced by out-of-control points
* Enables proper detection of process changes

**Consequences:**

* (+) Statistically correct
* (+) Realistic phase boundaries
* (-) Requires minimum 20 data points per phase

**Status:** ✅ Accepted

## 10. Future Architecture Considerations

### Potential Enhancements (V2.0)

**Architectural Evolution:**

1. **Phase 1 (Current):** Desktop tool, zero dependencies ✅
2. **Phase 2 (Future):** Optional web deployment for teams
   * Web Server: Flask/FastAPI
   * Database: PostgreSQL for historical data
   * Authentication: SSO integration
3. **Phase 3 (Future):** Integration with airline ERP systems
   * Real-time Updates: WebSocket
   * ML Predictions: Anomaly detection
   * REST API: External integrations

**Document Version History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Changes** |
| 1.0 | 2025-10-06 | Development Team | Initial architecture documentation |

*Southwest Airlines - Technical Operations Analytics Team*