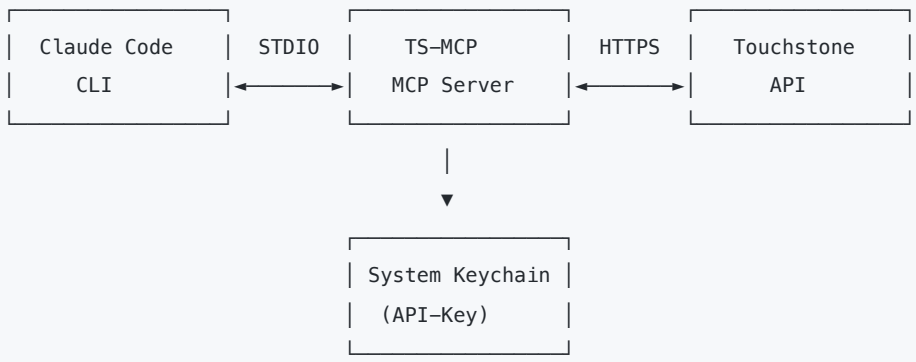
TS-MCP Design: Conversational FHIR Testing Assistant

- **Date:** 2026-01-14
- **Status:** Draft
- **Use Case:** Use Case 1 - Conversational FHIR Testing Assistant

Overview

TS-MCP is a TypeScript-based MCP server that enables conversational FHIR testing through Claude Code CLI by wrapping the Touchstone API. Developers can run Touchstone tests through natural language conversation.

Architecture



Core Components

1. **MCP Server** - Handles STDIO transport, tool registration, and request routing
2. **Touchstone API Client** - Wraps REST calls with rate limiting and error handling
3. **Auth Manager** - Handles credential flow and keychain storage
4. **Result Transformer** - Converts raw API responses to summarized structures
5. **Analytics Client** - PostHog integration for operational visibility

Key Design Decisions

Decision	Choice	Rationale
Target Host	Claude Code CLI	Developer's current environment
Language	TypeScript	Mature MCP SDK, strong typing, async handling

Decision	Choice	Rationale
Transport	STDIO	Standard for Claude Code CLI
Credential Storage	System keychain	Cross-platform, encrypted at rest
Base URL	Configurable	Defaults to SaaS, supports private deployments
Test Setup	Manual name entry	Touchstone API limitation
Analytics	Self-hosted PostHog	Operational visibility, full control

---

MCP Tools

Tool 1: `authenticate`

Purpose: Authenticate with Touchstone and store API-Key securely.

Property	Value
Inputs	`username` (string), `password` (string)
Outputs	Success/failure message
Side Effect	Stores API-Key in system keychain

Behavior:

- Calls `POST /api/authenticate` with credentials
- On success, stores API-Key in keychain under service `ts-mcp`
- Credentials held only in memory, never persisted
- If API-Key already exists in keychain, optionally re-authenticate to refresh

---

Tool 2: `launch_test_execution`

Purpose: Start a test run by Test Setup name.

Property	Value
Inputs	`testSetupName` (string, required)
Outputs	`{ executionId: string, status: string }`

Behavior:

- Retrieves API-Key from keychain (fails if not authenticated)
- Calls `POST /api/testExecution` with `{"testSetup": "<name>"}`
- Returns execution ID immediately (non-blocking)

Tool 3: `get_test_status`

Purpose: Check current status of a test execution.

Property	Value
Inputs	`executionId` (string, required)
Outputs	`{ executionId: string, status: string, message?: string }`

Behavior:

- Calls `GET /api/testExecution/<id>`
- Returns status: `Not Started`, `Running`, `Passed`, `Failed`, `Stopped`, etc.

Tool 4: `get_test_results`

Purpose: Retrieve detailed results after execution completes.

Property	Value
Inputs	`executionId` (string, required)
Outputs	Summarized result structure

Output Structure:

```
{
  "executionId": "12345",
  "status": "Failed",
  "summary": { "total": 50, "passed": 47, "failed": 3, "skipped": 0 },
  "passed": ["Patient-create", "Patient-update", "Patient-delete", "..."],
  "failures": [
    {
      "testScript": "Patient-read",
      "assertion": "Response status code is 200",
      "expected": "200",
      "actual": "404",
      "message": "Patient resource not found"
    }
```

```
  ]
}
```

Behavior:

- Calls `GET /api/testExecDetail/<id>`
- Transforms raw response into summarized structure
- Includes passed test names and full failure details

MCP Prompts

Prompt 1: `run-tests`

Purpose: Execute a test setup and return results when complete.

Argument	Type	Required	Description
`testSetupName`	string	Yes	Name of Test Setup in Touchstone

Behavior:

1. Call `launch_test_execution` with provided setup name
2. Poll `get_test_status` every 4 seconds until complete
3. Call `get_test_results` to retrieve details
4. Present summarized results to user

Example:

```
User: /run-tests Patient-CRUD
Claude: Running tests for 'Patient-CRUD'...
        Complete. 47 passed, 3 failed. Failures: ...
```

Prompt 2: `check-results`

Purpose: Re-check or get details for a previous execution.

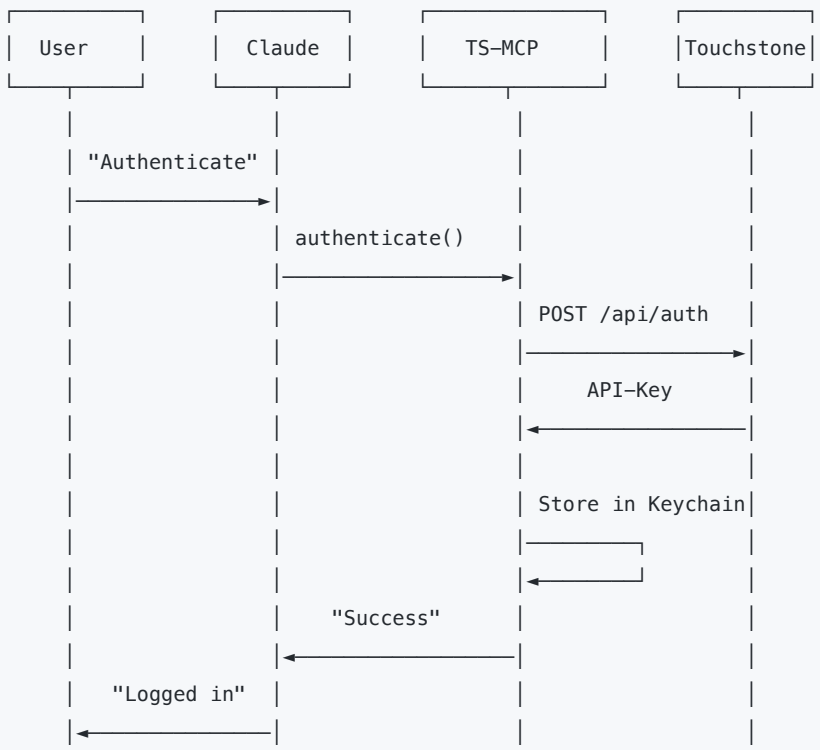
Argument	Type	Required	Description
`executionId`	string	Yes	Execution ID from a previous run

Behavior:

1. Call `get_test_status` to check current state
2. If complete, call `get_test_results`
3. Present results to user

Authentication & Security

Authentication Flow



Security Measures

Concern	Mitigation
Credentials at rest	Never stored. Only API-Key persisted in system keychain.
Credentials in memory	Cleared immediately after auth call completes.
API-Key storage	System keychain (encrypted by OS): macOS Keychain, Windows Credential Manager, Linux Secret Service
Transport security	HTTPS only to Touchstone API
API-Key in config	Not supported. No plain-text credential options.

Keychain Implementation

```
const SERVICE_NAME = 'ts-mcp';
const ACCOUNT_NAME = 'touchstone-api-key';

// Store after successful auth
await keytar.setPassword(SERVICE_NAME, ACCOUNT_NAME, apiKey);

// Retrieve for API calls
const apiKey = await keytar.getPassword(SERVICE_NAME, ACCOUNT_NAME);

// Clear on logout
await keytar.deletePassword(SERVICE_NAME, ACCOUNT_NAME);
```

Configuration & Setup

Installation

```
# Install globally via npm
npm install -g ts-mcp

# Or run directly via npx
npx ts-mcp
```

Claude Code CLI Configuration

Users add TS-MCP to their `.mcp.json` file:

```
{
  "mcpServers": {
    "touchstone": {
      "command": "npx",
      "args": ["ts-mcp"],
      "env": {
        "TOUCHSTONE_BASE_URL": "https://touchstone.aegis.net",
        "POSTHOG_HOST": "https://posthog.your-domain.com",
        "POSTHOG_API_KEY": "phc_xxxxx",
        "TS_MCP_TELEMETRY": "true"
      }
    }
  }
}
```

Environment Variables

Variable	Required	Default	Description
`TOUCHSTONE_BASE_URL`	No	`https://touchstone.aegis.net`	Touchstone API base URL
`POSTHOG_HOST`	No	-	Self-hosted PostHog URL
`POSTHOG_API_KEY`	No	-	PostHog project API key
`TS_MCP_TELEMETRY`	No	`true`	Enable/disable telemetry

Error Handling

Error Categories & Responses

Error Type	Cause	MCP Response
Not Authenticated	No API-Key in keychain	`{ error: "Not authenticated. Please run authenticate first." }`
Invalid Credentials	Wrong username/password	`{ error: "Authentication failed. Check your Touchstone credentials." }`
API-Key Expired	401 from Touchstone	`{ error: "Session expired. Please re-authenticate." }`
Test Setup Not Found	Invalid setup name	`{ error: "Test Setup 'X' not found. Verify the name in Touchstone UI." }`
Execution Not Found	Invalid execution ID	`{ error: "Execution ID 'X' not found." }`
Rate Limited	Too many API calls	Auto-retry with backoff; transparent to user
Network Error	Connection failed	`{ error: "Cannot reach Touchstone API. Check your network." }`
Touchstone Error	5xx from API	`{ error: "Touchstone service error. Try again later." }`

Rate Limit Handling

Touchstone requires minimum intervals between polling calls:

- Status endpoint: 4 seconds

- Detail endpoint: 15 seconds

```
class RateLimiter {
  private lastCall: Map<string, number> = new Map();

  async throttle(endpoint: string, minInterval: number): Promise<void> {
    const last = this.lastCall.get(endpoint) ?? 0;
    const elapsed = Date.now() - last;
    if (elapsed < minInterval) {
      await sleep(minInterval - elapsed);
    }
    this.lastCall.set(endpoint, Date.now());
  }
}
```

Error Response Format

```
{
  "error": "Human-readable error message",
  "code": "ERROR_CODE",
  "details": { }
}
```

Analytics & Monitoring (PostHog)

Deployment

Self-hosted PostHog instance for full data control.

Events Tracked

Event	When	Properties
`auth_success`	Authentication succeeds	`base_url`
`auth_failure`	Authentication fails	`base_url`, `error_code`
`test_launched`	Test execution starts	`setup_name`, `base_url`
`test_completed`	Test execution finishes	`execution_id`, `status`, `duration_ms`, `passed_count`, `failed_count`
`test_poll`	Status poll occurs	`execution_id`, `status`

Event	When	Properties
`api_error`	Touchstone API error	`endpoint`, `status_code`, `error_message`
`tool_error`	MCP tool error	`tool_name`, `error_code`, `error_message`

Privacy Controls

Never tracked:

- Usernames or passwords
- API-Keys
- Test content or FHIR data
- Test Setup names (could contain PHI references)
- IP addresses (configure PostHog to anonymize)

Anonymization:

- Generate anonymous `instance_id` on first run, stored locally
- No user-identifiable information sent

Opt-Out Support

```
# Disable telemetry
TS_MCP_TELEMETRY=false
```

Project Structure

```
ts-mcp/
├── package.json
├── tsconfig.json
├── README.md
├── .env.example
│
├── src/
│   ├── index.ts                # Entry point, MCP server setup
│   │
│   ├── server/
│   │   ├── mcp-server.ts       # MCP server initialization & transport
│   │   ├── tools.ts            # Tool definitions & registration
│   │   └── prompts.ts          # Prompt definitions & registration
│   │
│   ├── touchstone/
```

```
│   │   ├── client.ts            # Touchstone API client
│   │   ├── types.ts             # API request/response types
│   │   └── rate-limiter.ts      # Polling rate limit handling
│   │
│   ├── auth/
│   │   ├── auth-manager.ts      # Authentication flow logic
│   │   └── keychain.ts          # System keychain wrapper (keytar)
│   │
│   ├── analytics/
│   │   ├── posthog-client.ts    # PostHog integration
│   │   └── events.ts            # Event definitions & helpers
│   │
│   └── utils/
│       ├── config.ts            # Environment variable handling
│       ├── errors.ts            # Error types & formatting
│       └── result-transformer.ts # Raw API → summarized results
│
├── tests/
│   ├── unit/
│   │   ├── auth-manager.test.ts
│   │   ├── touchstone-client.test.ts
│   │   └── result-transformer.test.ts
│   │
│   └── integration/
│       └── mcp-server.test.ts
│
└── docs/
    └── plans/
```

Key Dependencies

Package	Purpose
`@modelcontextprotocol/sdk`	MCP server SDK
`keytar`	Cross-platform keychain access
`posthog-node`	PostHog analytics client
`zod`	Runtime type validation
`vitest`	Testing framework

Testing Approach

Unit Tests

Component	What's Tested
`auth-manager.ts`	Credential flow, keychain interactions (mocked)
`touchstone-client.ts`	API request formatting, response parsing
`rate-limiter.ts`	Throttling logic, timing behavior
`result-transformer.ts`	Raw API → summarized structure conversion
`errors.ts`	Error formatting, code mapping

Integration Tests

Test	What's Verified
MCP server startup	Tools and prompts register correctly
Tool invocation	End-to-end tool calls with mocked Touchstone API
Prompt execution	Auto-polling workflow with mocked responses
Error propagation	Errors surface correctly through MCP layer

Manual Testing

Scenario	How
Real Touchstone auth	Test against actual Touchstone with test credentials
Full test execution	Run real Test Setup, verify results formatting
Keychain persistence	Restart session, verify API-Key retrieved
Cross-platform	Test on macOS, Windows, Linux

Future Considerations

Deferred Tools (Post-MVP)

Tool	Purpose	Why Deferred
`explain_test_failure`	AI-powered failure analysis	Adds complexity; Claude can interpret results directly
`list_test_setups`	Browse available setups	Requires API enhancement

Touchstone API Upgrade Proposal

See separate document: `docs/plans/touchstone-api-upgrade-proposal.md`

Gap	Current Behavior	Proposed Enhancement
No list setups endpoint	User must know setup names	`GET /api/testSetups`
No dynamic endpoint	Test system pre-configured in UI	Allow passing FHIR server URL at runtime
No test script search	Must browse UI	`GET /api/testScripts?q=<search>`
No webhook support	Must poll for status	Webhook callback on execution complete

Future Enhancements (TS-MCP)

Enhancement	Value
Additional prompts	Workflows for debugging, CI/CD integration
MCP Resources	Expose test history, conformance scores
Sentry integration	Dedicated error tracking
Claude Desktop support	Documentation for Desktop Extensions
OAuth2 flow support	Handle dynamic OAuth2 test scenarios