



A E G I S

got Claude Code?

Validate your FHIR Implementations with Touchstone directly from Claude Code

- *FHIR faster*

Jeff Helman
March 12, 2026



Agenda

1. Claude Code* workflow tour
2. FHIR Implementation Validation: Then
3. FHIR Implementation Validation : Now
4. FHIR Faster with Touchstone and Claude*
5. Tooling Roadmap



*CoPilot, Codex and other AI CLIs coming soon.



A E G I S

*Powerful Results.
Delivered.*

AI-assisted Programming Workflow

- Let's step back from the headlines and hype
- Examine how these AI coding tools work
- What they are—and are not
- Who can (and should—or should not) use them
- Is there ROI yet?



AI-assisted Programming Workflow

Stepping back from the headlines and hype



TOUCHSTONE

AEGIS.net

www.touchstone.com

AI-assisted Programming Workflow

```
prior-auth — * Claude Code — claude — 133x35
~/DEV/prior-auth — * JwtDecoder Bean Error — claude
~/DEV/prior-auth — %
[jeffhelman@Jeffs-Laptop ~/DEV/prior-auth % claude
Claude Code v2.1.59
Opus 4.6 · Claude Max
~/DEV/prior-auth
>
? for shortcuts
```

From Windows Terminal/WSL or MacOS Terminal, type "claude"



AI-assisted Programming Workflow



The screenshot shows a terminal window with the following content:

```
prior-auth — * Claude Code — claude — 133x35
~/DEV/prior-auth — * JwtDecoder Bean Error — claude
[jeffhelman@Jeffs-Laptop ~/DEV/prior-auth % claude
 Claude Code v2.1.59
Opus 4.6 · Claude Max
~/DEV/prior-auth

> █

? for shortcuts
```

- Command-line interface (CLI) tools from the AI vendors are now serious accelerators.
- Experienced developers become **product orchestrators** @ 10X productivity



AI-assisted Programming Workflow

```
> Use all your superpowers to design and implement the functionality described in the HL7  
DaVinci Prior Authorization FHIR Implementation Guide found here:  
https://hl7.org/fhir/us/davinci-pas/
```

```
• I'm going to use the brainstorming skill to refine the design before jumping into implementation.  
This is a large, complex healthcare interoperability project that needs careful design work first.  
Let me start by understanding the specification and the current state of the project.
```

- Non-trivial **CMS-0057-F Example:**

- Design, plan and implement a solution conformant to the HL7 DaVinci Prior Auth FHIR IG.

- Plain language prompt from the command line, followed by confirmation from the AI

- A set of AI "skills" that focuses the CLI on software development best practices.
 - <https://github.com/obra/superpowers>



AI-assisted Programming Workflow

Which role are you building for?

1. **EHR/Provider Client** – A system that submits prior authorization requests to a payer/intermediary (the requesting side)
2. **Intermediary/Payer Server** – A system that receives prior authorization requests and returns decisions (the responding side)
3. **Both** – A full reference implementation covering both sides of the exchange
4. **Intermediary only** – A translation layer that sits between EHR and payer, handling FHIR-to-X12 conversion

* Churned for 1m 33s

> 2

AI walks us through **business** decisions. **NOTE:** Domain/dev expertise still required.

- Here we select a **Payer Server prior auth solution.**



TOUCHSTONE

AEGIS.net

www.touchstone.com

AI-assisted Programming Workflow

- Good – we're building the Intermediary/Payer Server side that receives and processes prior authorization requests.

What technology stack do you want to use?

1. Node.js/TypeScript with Express – Fast to develop, large FHIR library ecosystem (e.g., `fhir-works-on-aws`), good for REST APIs
2. Java with Spring Boot + HAPI FHIR – Industry standard for FHIR servers, HAPI provides built-in resource validation, profile support, and storage
3. Python with FastAPI – Clean async API framework, good for rapid prototyping, libraries like `fhir.resources` for validation
4. .NET with Firely SDK – Strong FHIR tooling, common in healthcare enterprise environments

My recommendation is option 2 (Java/Spring Boot + HAPI FHIR) – HAPI is the de facto open-source FHIR server library, provides out-of-the-box resource parsing/validation/serialization, profile validation, and is what most production healthcare systems use. It significantly reduces the amount of FHIR plumbing you need to write yourself.

> 2 ■

AI has researched the FHIR landscape, recommending HAPI FHIR for the base.



TOUCHSTONE

AEGIS.net

www.touchstone.com

AI-assisted Programming Workflow

Claude Code **walks me through decisions** about:

- Tech stack (and recommending options)
- Auto-adjudicating (at least some) prior auth determinations (“Yes”)
 - **Note:** This is NOT AI-driven adjudication; it is rules-engine based.
- Data persistence choices
- Message subscription/notification mechanism
- X12 integration options
- Security architecture



AI-assisted Programming Workflow

I think I have enough to present the design. Let me summarize what we've decided so far before I start presenting sections:

- **Role:** Intermediary/Payer Server
- **Stack:** Java + Spring Boot + HAPI FHIR JPA Server
- **Adjudication:** Auto-adjudication with Drools rules engine
- **Storage:** HAPI FHIR JPA Server (bundled persistence, search, versioning)
- **Notifications:** REST-hook subscriptions for pended authorizations
- **X12:** Pluggable conversion interface, ships FHIR-native only
- **Security:** Pluggable auth with OAuth 2.0 support + dev-mode bypass

One last question before I present the design:

What build tool and Java version?

1. **Maven + Java 17** – Maven is the standard in healthcare/HAPI FHIR ecosystem. Java 17 is the current LTS and what HAPI FHIR targets.
2. **Gradle + Java 21** – Gradle is faster and more flexible. Java 21 is the newest LTS with virtual threads (Project Loom), which is great for high-concurrency FHIR servers.
3. **Maven + Java 21** – Stay with the HAPI ecosystem standard build tool but use the latest LTS for language features and virtual threads.

My recommendation is **option 3 (Maven + Java 21)** – Maven keeps us aligned with the HAPI FHIR ecosystem (all their examples and plugins are Maven-based). Java 21 gives us virtual threads for handling concurrent prior auth requests efficiently, plus modern language features like records and pattern matching.

... until it “knows” enough to create the design.



TOUCHSTONE

AEGIS.net

www.touchstone.com

Section 1: High-Level Architecture

The system is a Spring Boot application built on top of the HAPI FHIR JPA Server that implements the DaVinci PAS Intermediary/Payer role. It exposes two custom FHIR operations (`$submit` and `$inquiry`) and layers a Drools rules engine on top for auto-adjudication.

Core components:

- FHIR API Layer – Spring Boot + HAPI FHIR JPA Server providing REST endpoints, resource parsing/validation/serialization, persistence, search, and subscription infrastructure
- PAS Operation Providers – Custom HAPI resource providers implementing `Claim/$submit` and `Claim/$inquiry` operations with PAS-specific Bundle validation
- Adjudication Engine – Drools-based rules engine that evaluates prior authorization requests against configurable business rules and returns approve/deny/pend decisions
- Notification Service – Manages REST-hook subscriptions and dispatches notifications when pended authorizations receive decisions
- X12 Conversion SPI – A Java interface for pluggable FHIR-to-X12 translation, shipped with a no-op FHIR-native implementation
- Security Layer – Spring Security filter chain supporting pluggable OAuth 2.0 token validation with a dev-mode API key bypass

Request flow: EHR submits a PAS Request Bundle → FHIR API validates structure and profiles → Adjudication engine evaluates rules → ClaimResponse is generated → Response Bundle returned (within 15-second SLA). If pended, the request is persisted and the notification service watches for resolution.

Does this high-level architecture look right so far?

Al now walks me through its proposed functional design ...



TOUCHSTONE

AEGIS.net

www.touchstone.com

● Section 9: Capability Statement & Conformance

The server publishes a FHIR `CapabilityStatement` at `[base]/metadata` declaring conformance to the PAS Intermediary role. This is critical for EHR systems discovering what the server supports.

Key declarations in the `CapabilityStatement`:

- FHIR version: 4.0.1
- Implementation Guide: `http://hl7.org/fhir/us/davinci-pas|2.1.0`
- Operations: `Claim/$submit` and `Claim/$inquiry`
- Supported profiles: All 25+ PAS resource profiles listed as supported
- REST interactions: Subscription create/read/delete for pended authorization notifications
- Security: OAuth 2.0 authorization endpoint and token endpoint advertised (when prod profile active)

We implement this as a custom HAPI `ServerCapabilityStatementProvider` that extends the auto-generated `CapabilityStatement` with PAS-specific declarations. This ensures the metadata endpoint always accurately reflects what the server supports.

Conformance testing posture: By loading the PAS IG NPM package into HAPI's validation infrastructure, we get automatic profile validation for both incoming requests and outgoing responses. The server validates its own Response Bundles before returning them – if we generate an invalid response (a bug), it throws an internal error rather than sending a non-conformant response. This "validate on the way out" approach catches implementation errors early.

That covers the full design. Does this final section look right?

> yes

... until we have iterated enough and approve the design.



TOUCHSTONE

AEGIS.net

www.touchstone.com

AI-assisted Programming Workflow

After this “Socratic interview,” we have a *well-documented, domain-specific, architecturally sound, design document.*

DaVinci PAS Intermediary/Payer Server — Design Document

Summary

A FHIR R4 server implementing the HL7 DaVinci Prior Authorization Support (PAS) Implementation Guide v2.1.0 in the Intermediary/Payer role. Built on Java 17, Spring Boot, HAPI FHIR JPA Server, with a Drools rules engine for auto-adjudication.

Decisions

Decision	Choice	Rationale
Role	Intermediary/Payer Server	Receives and processes prior auth requests
Stack	Java 17 + Spring Boot + HAPI FHIR JPA Server	Industry standard, built-in FHIR persistence/search/subscriptions
Adjudication	Drools rules engine	Complex healthcare rules, decision tables for business analysts
Storage	HAPI FHIR JPA Server (bundled)	Turnkey FHIR resource persistence, versioning, search indexing
Notifications	REST-hook only	Mandatory per spec; WebSocket is optional, added later if needed
X12	Pluggable SPI, ships FHIR-native only	Clean architecture without premature complexity
Security	Pluggable OAuth 2.0 + dev-mode bypass	Production-flexible, easy local development
Build	Maven + Java 17	Maximum HAPI FHIR ecosystem compatibility

1. High-Level Architecture

The system is a Spring Boot application built on the HAPI FHIR JPA Server that implements the DaVinci PAS



TOUCHSTONE

AEGIS.net

www.touchstone.com

• Plan complete and saved to docs/plans/2026-02-24-davinci-pas-implementation-plan.md.

15 tasks across 8 phases:

Phase	Tasks	What
1	1-2	Maven multi-module scaffolding
2	3	X12 conversion SPI
3	4-5	Drools adjudication engine + rules
4	6	Security (dev API key + prod OAuth)
5	7	REST-hook notification dispatcher
6	8-11	FHIR operations (\$submit, \$inquire, validators, builders)
7	12-13	Server assembly + CapabilityStatement
8	14-15	Integration tests + verification

Two execution options:

1. **Subagent-Driven (this session)** – I dispatch a fresh subagent per task, review between tasks, fast iteration
2. **Parallel Session (separate)** – Open a new session with `executing-plans`, batch execution with checkpoints

Which approach?

* Worked for 21m 16s

>

... then Claude Code creates a **detailed implementation plan**.



TOUCHSTONE

AEGIS.net

www.touchstone.com

AI-assisted Programming Workflow

Comprehensive Implementation Plan

DaVinci PAS Server Implementation Plan

For Claude: REQUIRED SUB-SKILL: Use superpowers:executing-plans to implement this plan task-by-task.

Goal: Build a FHIR R4 Intermediary/Payer server implementing DaVinci PAS IG v2.1.0 with auto-adjudication.

Architecture: Multi-module Maven project built on HAPI FHIR JPA Server with Drools rules engine. Six modules: pas-server, pas-operations, pas-adjudication, pas-notification, pas-x12, pas-security.

Tech Stack: Java 17, Spring Boot 3.2.5, HAPI FHIR 7.4.0, Drools 9.44.0.Final, PostgreSQL, Maven

Design Document: [docs/plans/2026-02-24-davinci-pas-server-design.md](#)

Phase 1: Project Scaffolding

Task 1: Create Parent POM

Files:

- Create: `pom.xml`

Step 1: Create the parent POM

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.5</version>
    <relativePath/>
```



AI-assisted Programming Workflow

Question for You:

How long do your teams typically spend creating a design and detailed implementation plan (for a typical HL7 FHIR IG like Da Vinci's PAS IG)?



AI-assisted Programming Workflow

Question for You:

How long do your teams typically spend creating a design and detailed implementation plan (for a typical HL7 FHIR IG like Da Vinci's PAS IG)?

A month?



AI-assisted Programming Workflow

Question for You:

How long do your teams typically spend creating a design and detailed implementation plan (for a typical HL7 FHIR IG like Da Vinci's PAS IG)?

- A month?
- 2 weeks?



AI-assisted Programming Workflow

Question for You:

How long do your teams typically spend creating a design and detailed implementation plan (for a typical HL7 FHIR IG like Da Vinci's PAS IG)?

- A month?
- 2 weeks?
- 1 week?
- A day or two?



AI-assisted Programming Workflow

My Metrics:

- **Design** took < 1 hour.
 - I spent another 30 minutes to review.
- The **implementation plan** took < 5 minutes.
 - I spent an hour to review.



AI-assisted Programming Workflow

I approved the implementation plan, and Claude Code **completed this baseline implementation within a couple of hours**.

- A couple of hours because I (a domain/technical expert) am still approving major actions
- Others may have a higher risk tolerance, so they could spend less time on the initial iteration.



AI-assisted Programming Workflow

FHIR IG to Baseline Implementation: ~ 4-5 hours



TOUCHSTONE

AEGIS.net

www.touchstone.com

AI-assisted Programming Workflow

- I think there may be some ROI here.



AI-assisted Programming Workflow

- I think there may be some ROI here.
- However, this is not “VIBING” ...



AI-accelerated DELIVERY Workflow

It is solution delivery acceleration.



TOUCHSTONE

AEGIS.net

www.touchstone.com

~~Vibe Coding~~ AI-accelerated DELIVERY Workflow

It is solution delivery acceleration.
BUT ... it's NOT "Vibe Coding"



TOUCHSTONE

AEGIS.net

www.touchstone.com

~~Vibe Coding~~ AI-accelerated DELIVERY Workflow

It is solution delivery acceleration.
BUT ... it's NOT "Vibe Coding"

- **Domain and technical expertise is a must**



TOUCHSTONE

AEGIS.net

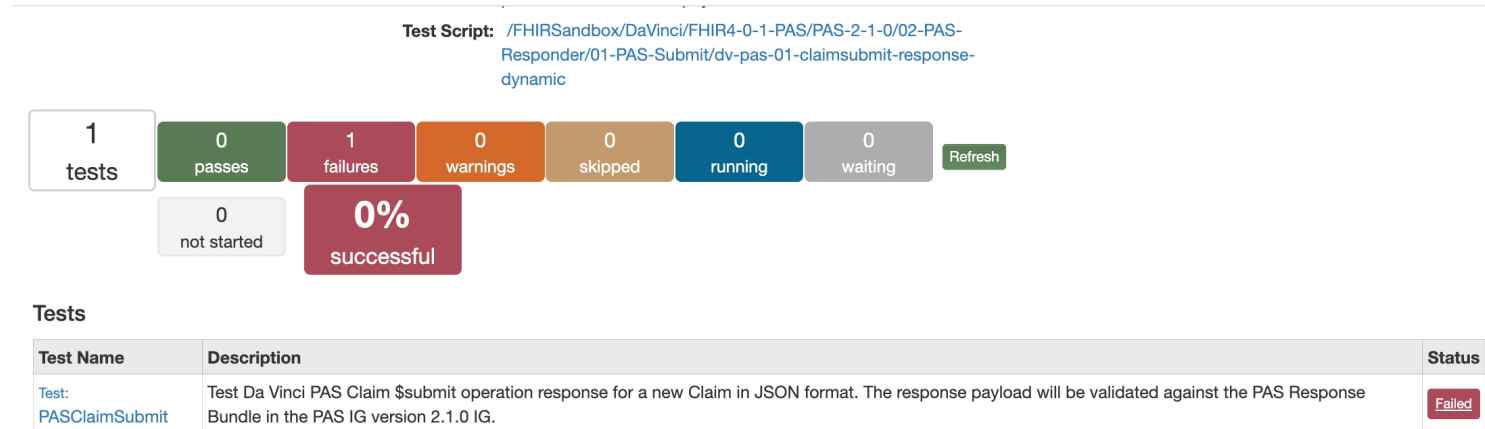
www.touchstone.com

FHIR Implementation Validation Best Practices: Then and Now



Implementation Validation Best Practices - THEN

- Sub-systems pass internal testing.
- Deploy to a QA environment.
- Run a testing scenario in AEGIS Touchstone.
- Investigate errors.
- Developer(s) dig back into that code.
- Fix defect(s) and redeploy
- Iterate until no more defects found.
- Developer(s) **context switch** back into their current code.



NOW: Instant Touchstone Validation via CLI

- Developer finishes individual feature.
- Types "Run my testing scenario in Touchstone"
- "Here are the results."
- Developer types "**fix those bugs**"*
- Then: "Try that again"
- TS-MCP: "Pass."
- Developer momentum preserved.
- **Delivery is accelerated.**

```
• The execution completed with status Failed. Let me pull the detailed results.
• ts-mcp - get_test_results (MCP)(executionId: "202602271154391587442882", verbose: true)
  L {
    "executionId": "202602271154391587442882",
    "status": "Failed",
    ... +43 lines (ctrl+o to expand)
• Here are the results:

Test: PASClaimSubmit – FAILED (0/1 passed)

The $submit operation returned a 503 with an OperationOutcome instead of the expected 200/201 with a PAS Response Bundle. The specific failures:
```

Assertion	Issue
Response code is 200 or 201	Got 503
Resource type is Bundle	Got OperationOutcome
Bundle.type is "collection"	Absent
Bundle contains 1 ClaimResponse	Found 0
Validates against PAS Response Bundle profile	Type mismatch (OperationOutcome vs Bundle)

```
The root issue is that your server returned a 503 Service Unavailable. This suggests the server may be down or encountering an error processing the $submit request. Would you like me to investigate what might be causing the 503?
```

*BUT WAIT! ... It gets *better*.



NOW: Instant Touchstone Validation via CLI

Code quality
and coverage
are surfaced
where and when
remediation is
most effective.

- The execution completed with status **Failed**. Let me pull the detailed results.
- `ts-mcp - get_test_results (MCP)(executionId: "202602271154391587442882", verbose: true)`
└ {
 "executionId": "202602271154391587442882",
 "status": "Failed",
 ... +43 lines (ctrl+o to expand)

- Here are the results:

Test: PASClaimSubmit – FAILED (0/1 passed)

The `$submit` operation returned a **503** with an `OperationOutcome` instead of the expected **200/201** with a PAS Response Bundle. The specific failures:

Assertion	Issue
Response code is 200 or 201	Got 503
Resource type is Bundle	Got OperationOutcome
Bundle.type is "collection"	Absent
Bundle contains 1 ClaimResponse	Found 0
Validates against PAS Response Bundle profile	Type mismatch (OperationOutcome vs Bundle)

The root issue is that your server returned a **503 Service Unavailable**. This suggests the server may be down or encountering an error processing the `$submit` request. Would you like me to investigate what might be causing the 503?



TOUCHSTONE

AEGIS.net

www.touchstone.com

FHIR Faster with Touchstone and Claude Code

How it Works

- Model Context Protocol (**MCP**) is a standard that defines how AI systems safely and predictably interact with business data and tools.
- We call this “TS-MCP”
 - TS-MSP connects Claude Code* to the Touchstone API
 - TS-MCP is freely available.

*CoPilot, Codex and other AI CLIs coming soon.



TOUCHSTONE

AEGIS.net

www.touchstone.com

FHIR Faster with Touchstone and Claude Code

Getting Started with TS-MCP (Beta)

1. Install in local dev environment:

```
npx github:AEGISnetInc/TS-MCP -help
```

2. Add to Claude Code:

```
claude mcp add ts-mcp -- npx github:AEGISnetInc/TS-MCP
```

3. Touchstone Authentication (to your existing Touchstone account):

```
npx github:AEGISnetInc/TS-MCP auth
```



FHIR Faster with Touchstone: Roadmap

got (other) AI-coding CLIs?

- MS/GitHub CoPilot CLI (next up)
- OpenAI Codex?
- Gemini CLI?
- Aider?
- Others?
- **Look for my [LinkedIn Poll](https://www.linkedin.com/in/jeffhelman/) the week of 3/16 for input into priorities**

<https://www.linkedin.com/in/jeffhelman/>



Next TS-MCP Release: [Auto-defect-fix](#) TS-MCP prompt

Soon: Extensive Touchstone API enhancements



TOUCHSTONE

AEGIS.net

www.touchstone.com

Next Steps

Click the left QR Code for this presentation.

Click on the right QR Code to schedule a call to influence our roadmap.

Get This Presentation



<https://touchstone.com/got-claude-code/>

Let's Talk



<https://calendly.com/d/cv9f-t65-7d2/discovery-call-with-the-aegis-team>



TOUCHSTONE

AEGIS.net

www.touchstone.com