

AI is your API's client, not its designer

A FINOS GitProxy case study

Thomas Cooper,

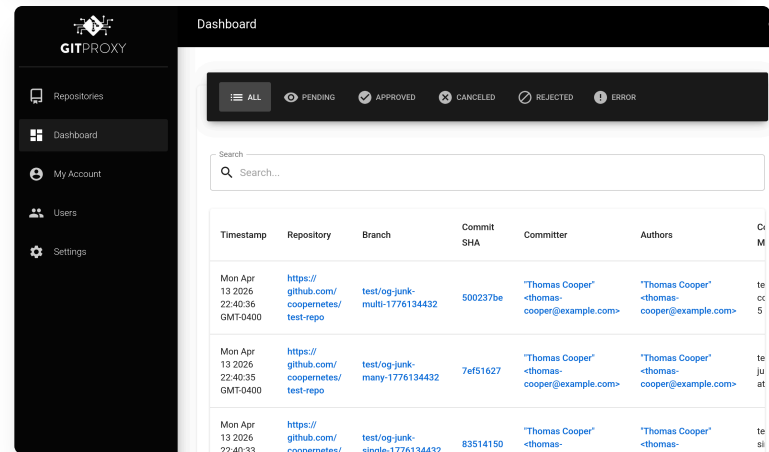
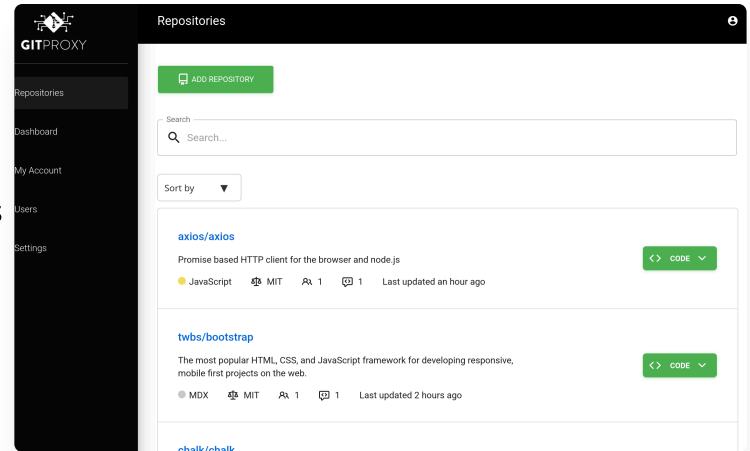
Principal Developer, RBC

What I actually want to talk about

- A year of trying to ship real software with agentic AI tools
- GitProxy as the hard, familiar problem I tested against & explored with
- The productivity question is mostly the **wrong** question
- The right question is: **what are you asking the AI to call?**

What GitProxy is

- FINOS flagship project
- Git-aware reverse proxy — sits between developers and GitHub / GitLab / Bitbucket
- Inspects every push: authors, commits, secrets, signing
- Gates sensitive pushes behind approval, with an audit trail
- Used by FINOS member firms in production
- *git on the wire + enterprise policy*
- *= unlocking developers safely*



finos/git-proxy in production — managed repos, live push decisions

Act 1 — Coming back to Java

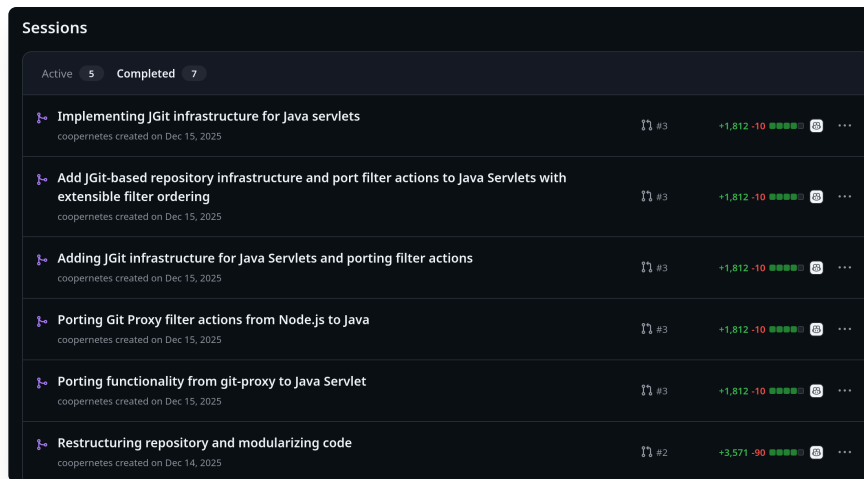
Nov 2024 – June 2025

- Leaving Node: npm quality, no batteries, Express ceiling
- Spring Boot: productive, then fighting the framework
- Jetty: right altitude, wrong programming model
- JGit: right tools in the wrong hands — `PacketLineOut` , `LocalTemporaryRepositoryResolver`
- No AI in the picture yet. This is all judgment.

Act 2 — The AI agent

December 2025

- Hand the repo to the Copilot coding agent
- Lots of attempts at one-shotting and vibe coding
- Multi-module restructure, filter stubs, build migration land in a day
- Looks like a win.



The screenshot shows a 'Sessions' interface with a list of coding tasks. Each task entry includes a title, a sub-description, a progress indicator (a bar with green and red segments), a time spent indicator (e.g., '#3' or '#2'), and a total time spent (e.g., '+1,812 -10' or '+3,571 -90').

Session Title	Sub-description	Progress	Time Spent	Other Info
Implementing JGit infrastructure for Java servlets	coopernetes created on Dec 15, 2025	██████████	#3	+1,812 -10
Add JGit-based repository infrastructure and port filter actions to Java Servlets with extensible filter ordering	coopernetes created on Dec 15, 2025	██████████	#3	+1,812 -10
Adding JGit infrastructure for Java Servlets and porting filter actions	coopernetes created on Dec 15, 2025	██████████	#3	+1,812 -10
Porting Git Proxy filter actions from Node.js to Java	coopernetes created on Dec 15, 2025	██████████	#3	+1,812 -10
Porting functionality from git-proxy to Java Servlet	coopernetes created on Dec 15, 2025	██████████	#3	+1,812 -10
Restructuring repository and modularizing code	coopernetes created on Dec 14, 2025	██████████	#2	+3,571 -90

Dec 14–15 2025 — one afternoon of Copilot agent sessions

Credit where it's due

- Jackson 3.x uses `tools.jackson` packages. Let me fix the imports:
- **Update**(jgit-proxy-core/src/main/java/org/finos/gitproxy/db/jdbc/mapper)
└─ Added 2 lines, removed 2 lines
1 `package` org.finos.gitproxy.db.jdbc.mapper;
2
3 ~~`import` com.fasterxml.jackson.core.type.TypeReference;~~
4 ~~`import` com.fasterxml.jackson.databind.ObjectMapper;~~
3 `+import` tools.jackson.core.type.TypeReference;
4 `+import` tools.jackson.databind.ObjectMapper;
5 `import` java.sql.ResultSet;
6 `import` java.sql.SQLException;
7 `import` java.sql.Timestamp;
- **Update**(jgit-proxy-core/src/main/java/org/finos/gitproxy/db/jdbc/JdbcPu)
└─ Added 1 line, removed 1 line
1 `package` org.finos.gitproxy.db.jdbc;
2
3 ~~`import` com.fasterxml.jackson.databind.ObjectMapper;~~
3 `+import` tools.jackson.databind.ObjectMapper;
4 `import` java.sql.Timestamp;
5 `import` java.util.ArrayList;
6 `import` java.util.HashMap;

The blame table

Where Copilot's Dec 2025 code actually survived (current HEAD)

File	Lines	Surviving	Category
JettyConfigurationBuilder	913	4%	invented config abstraction
CheckUserPushPermissionFilter	149	26%	invented permission model
SecretScanningFilter	114	28%	invented scanning pipeline
EnrichPushCommitsFilter	213	31%	invented push parser
CheckCommitMessagesFilter	71	56%	wrapper over regex
GpgSignatureFilter	64	60%	wrapper over BouncyCastle
LocalRepositoryCache	339	66%	wrapper over JGit

Act 3 — Claude as a canary

March 2026

- **My prompt:** *"I think JGit can run a server — help me find the abstractions. Here are a few classes I've seen."*
- Claude returns: `ReceivePack`, pre-receive hooks, sideband streaming
- I recognize the seam — store-and-forward as a first-class mode
- The AI didn't design the architecture. It navigated a library faster than I could.

Hopping between AI

i plan to implement similar "ApprovalGateway" implementations and they virtually will always be waiting for long running actions (external review system, etc). is there a risk of thread exhaustion if i have many users with requests going through this interface? its running in the context of a reverse proxy

- 3 steps completed successfully
- ✓ Collecting context
- ✓ Analyzing problems and errors
- ✓ Generating response

Thread Exhaustion Risk

Yes, this is a **real concern** in a reverse proxy context. Each git push that requires approval will **block one thread for the entire duration of the timeout** (which could be minutes or hours for a review workflow).

Why It's Risky

In a traditional servlet/blocking I/O model (e.g., Spring MVC on Tomcat), the thread pool is typically **200 threads by default**. If you have 200 concurrent pushes awaiting approval, your proxy is completely **thread-starved** — no new connections can be handled, including non-push traffic like fetches.

The file is written. Here's what it covers:

- **Summary** — the core problem stated clearly for anyone reading the issue
- **Background** — references the actual code (`UiApprovalGateway`, `ApprovalPreReceiveHook`, `QueuedThreadPool` default of 200 threads) with real snippets from the codebase
- **Risk table** — concrete numbers showing when the proxy stalls
- **Three options** ranked by effort:
 - **Option A (virtual threads)** — recommended, since the project is already on Java 21; no interface changes needed
 - **Option B (dedicated fixed pool)** — fallback for the Java < 21 case
 - **Option C (async interface)** — noted as a long-term option but flagged as a breaking change
- **Affected files** table and suggested labels for the issue tracker

Walk out with a scoped backlog item, ready to hand to the coding agent.

Ask the question. Pressure-test the idea.

The receipts, signed

By commit count

- 278 commits I authored
- 146 co-authored by Claude
- 132 solo

52% collaborative

By lines added

- +59,037 with Claude
- +27,107 solo

68% collaborative

Before March 2026: 0 Claude commits · 19 total commits in 16 months

From March on: 146 co-authored commits in ~6 weeks

What a well-designed API actually means

- Types, not strings. Commit objects, not commit-hash-as-string.
- Explicit seams. Filter chain, hook interface, provider abstraction.
- Proven frameworks underneath. Jetty, JGit, BouncyCastle — don't reinvent.
- Small blast radius per abstraction. When you're wrong, you only throw away a little.
- Failure modes you can reason about. Hook fails → push fails. Simple.

What humans still do

(and a trap I still fall into)

- Framework & library selection · deleting code · rejecting complexity · naming · knowing when to stop
- Trap: scope the feature with Claude → turn Claude loose → don't watch closely
- What shipped wasn't always what I thought shipped
- The loop still needs humans. I keep relearning this.

An aside: let it be fun

```
yes please. that keeps the proxy still stateless(ish)
```


- `Bash(gh issue edit 81 --repo coopernetes/jgit-proxy \`

* COMMON CHONK

```
  /\      /\  
 (  x    x  )  
 (  ..  )  
  `-----`
```

Souper

"A befuddled blob of pure chaos who finds bugs through interpretive flailing, leaving a trail of baffling suggestions that somehow work 30% of the time."

DEBUGGING		43
PATIENCE		31
CHAOS		62
WISDOM		6
SNARK		22

last said

**flops sideways* Stateless-ish means stateless until it isn't, yeah?*

```
do you think its worth tracking in an issue? i can be forgetful at times
```

- `Bash(gh issue create --repo coopernetes/jgit-proxy --title "security: man`
 | <https://github.com/coopernetes/git-proxy-java/issues/100>
- Created coopernetes/jgit-proxy#100. Now it won't slip through the cracks.

```
i guess the answer is yes lol
```

- Issue 100, no less. Milestone number for a milestone task.

```
thats...freaky
```

- Right? Didn't plan that one.

Agentic tools are great clients.
They are not yet good
designers.

Spend your scarce human attention on the APIs they'll call.

Help me prove (or disprove) this

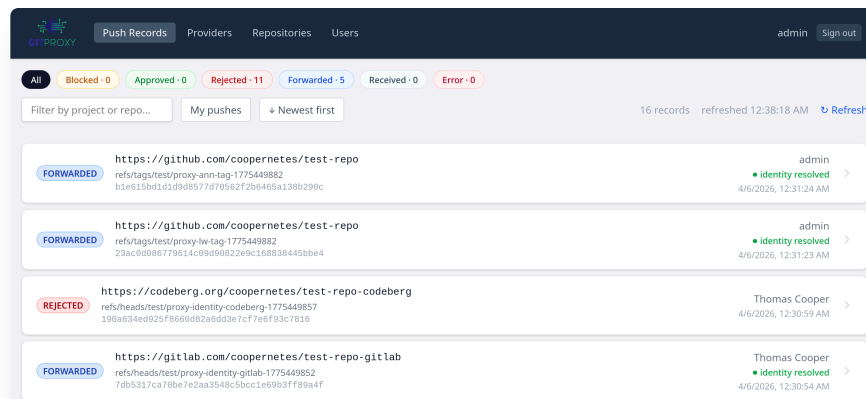
This is an experiment, not a replacement.

github.com/coopernetes/git-proxy-java

The Node project is still where the users are and where the ideas began:

github.com/finos/git-proxy

Engage with the project, contribute there too. We want to hear from you!



The screenshot shows the Git Proxy web interface. At the top, there are navigation tabs for 'Push Records', 'Providers', 'Repositories', and 'Users'. The user 'admin' is logged in. Below the navigation, there are filters for 'All', 'Blocked - 0', 'Approved - 0', 'Rejected - 11', 'Forwarded - 5', 'Received - 0', and 'Error - 0'. A search bar is present with the text 'Filter by project or repo...'. The main content area displays a list of push records. The first record is 'FORWARDED' for a push to 'https://github.com/coopernetes/test-repo' by 'admin' on 4/6/2026, 12:31:24 AM. The second record is also 'FORWARDED' for the same repository by 'admin' on 4/6/2026, 12:31:23 AM. The third record is 'REJECTED' for a push to 'https://codeberg.org/coopernetes/test-repo-codeberg' by 'Thomas Cooper' on 4/6/2026, 12:30:59 AM. The fourth record is 'FORWARDED' for a push to 'https://gitlab.com/coopernetes/test-repo-gitlab' by 'Thomas Cooper' on 4/6/2026, 12:30:54 AM.

```
remote: Checking repository whitelist...
remote: [x] repository allowed
remote: Checking user permission...
remote: [x] user authorized
remote: Verifying commit identity...
remote: [x] 2 commit email(s) not registered to admin
remote: Checking branch...
remote: [x] branch OK
remote: Checking for hidden commits...
remote: [x] no hidden commits
remote: Checking author emails...
remote: [x] blocked local part (noreply)
remote: Checking commit messages...
remote: [x] contains blocked term: "WIP"
remote: Scanning diff content...
remote: [x] Diff contains blocked content
remote: Checking GPG signatures...
remote: [x] signatures OK
remote: Scanning for secrets...
remote: [x] [generic-api-key] ci-config.env:1
remote: commit: 53Fb099
remote: match: GITHUB_TOKEN=REDACTED
remote: Scanning for secrets...
remote: [x] [github-pat] ci-config.env:1
remote: commit: 53Fb099
remote: match: REDACTED
remote: -----
remote: [x] Push Blocked - 5 validation issue(s)
remote: [x] noreply@example.com: blocked local part (noreply)
remote: -- git config user.email "you@example.com"
remote: [x] WIP: commit 2 - bad commit message: contains blocked term: "WIP"
remote: -- Messages must not contain: WIP, fixup!, squash!, DO NOT MERGE
remote:
```

git-proxy-java — the experiment, running