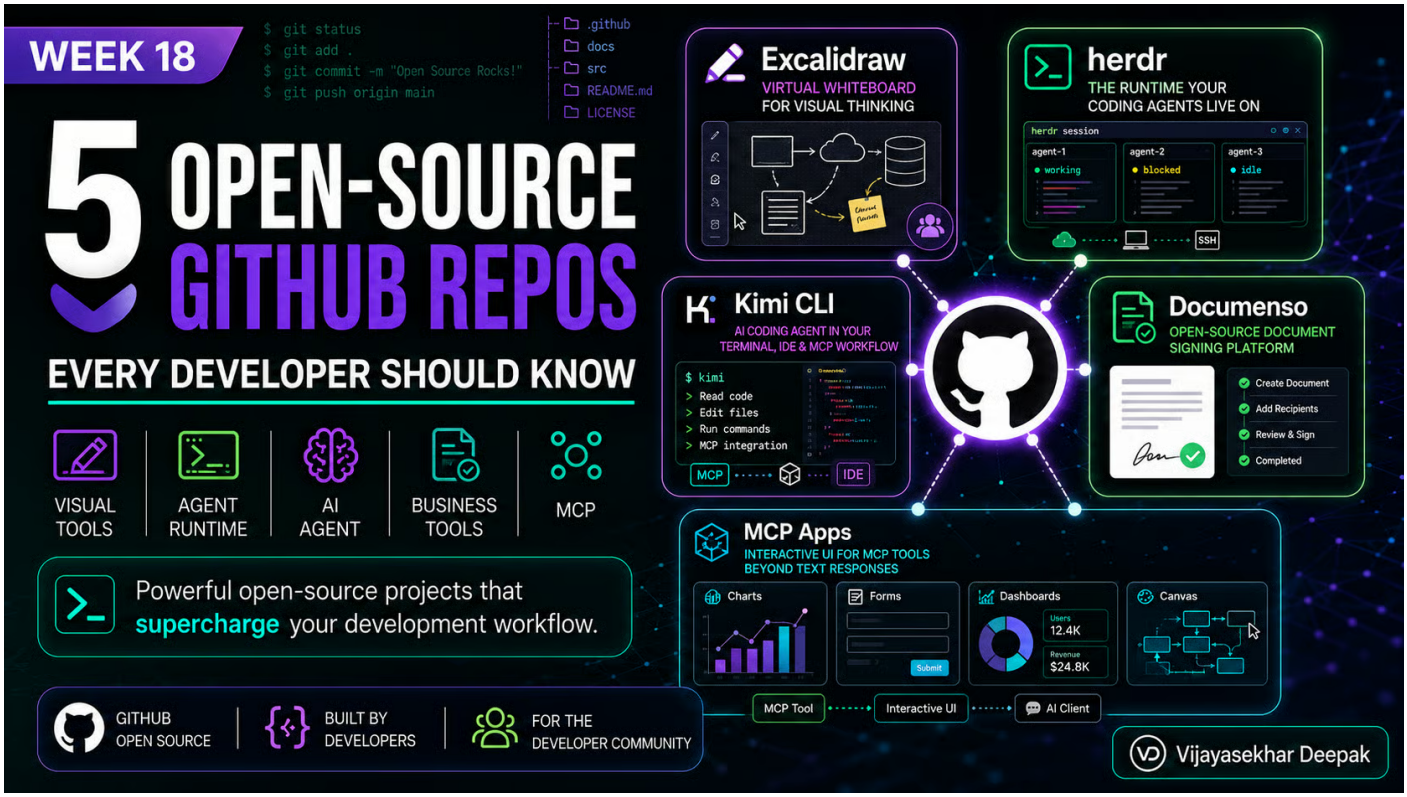
Week 18: 5 Open-Source GitHub Repositories Every Developer Should Know in 2026



Vijayasekhar Deepak
Sep 30, 2026



Developers are increasingly working with AI agents that run for long periods, collaborative visual interfaces, MCP servers, browser-based tools, document workflows, and applications that need to interact with AI in much richer ways than plain text.

And that creates a new problem.

For **Week 18 of my Open Source GitHub Repository Series**, I explored five repositories that solve very different problems:

- **Excalidraw** → Build visual collaboration and diagramming directly into your applications
- **herdr** → Give coding agents a persistent runtime where they can keep working
- **Kimi CLI** → Bring an AI coding agent into your terminal, IDE, and MCP workflow
- **Documenso** → Build document-signing workflows without handing your infrastructure to a proprietary platform
- **MCP Apps** → Turn MCP tools from text-only experiences into interactive UIs

What makes this list especially interesting is that these projects represent something bigger.

AI-native development isn't only about better models. It's about building better environments around those models.

Let's look at each one.

1. Excalidraw: Turn Your Application Into a Collaborative Whiteboard

Repository: *excalidraw/excalidraw*

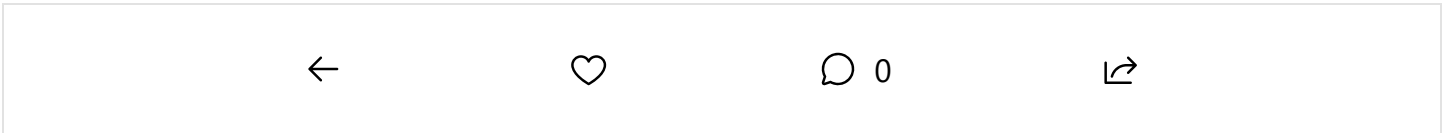
Sometimes the fastest way to explain a system isn't another 500-line document.

It's drawing a simple box with an arrow or a database cylinder and few notes. And suddenly everyone understands the architecture.

That is the problem Excalidraw solves remarkably well.

What is Excalidraw?

Excalidraw is an open-source virtual whiteboard for creating hand-drawn-style diagrams, wireframes, and visual ideas.



The hosted application also provides capabilities such as real-time collaboration, end-to-end encryption, offline/PWA support, local-first autosaving, and shareable read-only links.

But the part developers should pay attention to is this **Excalidraw isn't only a website. It's also something you can embed into your own product.**

The README provides a straightforward npm integration:

```
npm install react react-dom @excalidraw/excalidraw
```

That means you can bring a collaborative, sketch-style canvas into an existing React application rather than building a drawing engine from scratch.

Why Developers Should Care

Most applications eventually need some kind of visual interaction.

Think about:

- system design tools
- architecture diagrams
- workflow builders
- brainstorming applications
- educational platforms
- whiteboard products
- diagramming tools
- planning interfaces

Building canvas interactions yourself sounds easy until you start dealing with **selection + dragging + resizing + arrows + bindings + undo/redo + zoom + export**

0

Excalidraw gives you a large portion of that foundation.

And its integrations are another signal of how reusable the project has become. The repository lists integrations including VS Code, Obsidian, Notion, Replit, CodeSandbox, Meta, and others.

Real Development Use Cases

Architecture Design: Create a visual system-design canvas inside an internal engineering portal.

Product Planning: Allow product teams to sketch user journeys and workflows.

Education: Build interactive technical diagrams into developer-learning platforms.

Collaborative Design: Let multiple users work on the same visual canvas in real time.

Developer Tools: Embed diagrams directly into documentation or engineering workflows.

Productivity Impact

The biggest productivity win is simple: **You don’t have to reinvent the canvas.**

Instead of spending weeks building drawing primitives and interaction behavior, you can focus on the actual product experience.

And there is another advantage.

Because Excalidraw uses an open drawing format, your visual data doesn’t have to disappear inside an opaque proprietary format.

Sometimes productivity isn’t about writing code faster. It’s about deciding which code you shouldn’t write at all.

   0 

Here's a problem that becomes obvious once you start using coding agents seriously.

- What happens when your AI agent is working on something that takes longer than your current terminal session?
- What happens when you close the laptop?
- What happens when your network disappears?
- What happens when you have three agents running at the same time?

Traditional terminal workflows aren't really designed for this.

herdr is.

What is herdr?

herdr describes itself as **"the runtime your coding agents live on."** It runs as a background server where agent terminals live inside persistent sessions.

According to the README, you can close the laptop, lose the network, or restart the machine and the agents can continue working, with sessions available to reattach later, including over SSH.

That changes the mental model.

Instead of: `Terminal → Agent → Done`you start thinking: `Runtime → Agent Sessions → Persistent Work`

herdr also tracks whether panes are **working, blocked, or idle**, helping you identify when an agent needs attention.

And importantly, herdr isn't trying to replace your coding agent.

The README explicitly says it can run tools you already use including Claude Code, Codex, Cursor, OpenCode, Grok, and others while managing the terminals where they run.







AI coding agents are moving from **“answer this coding question”** to **“work on this task.”**

That difference matters. A task-oriented agent may need to:

1. inspect the repository
2. understand the issue
3. modify code
4. run tests
5. discover failures
6. fix them
7. continue iterating

That isn’t always a five-minute interaction. The environment itself becomes part of the agent architecture.

herdr treats that environment as infrastructure.

Real Development Use Cases

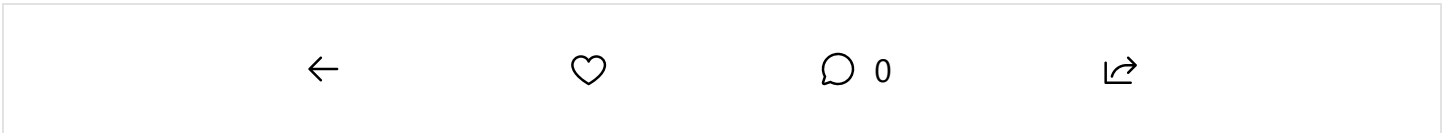
Long-Running Coding Tasks: Keep coding agents alive while they work through larger tasks.

Multiple Agents: Run several agents in different panes and monitor what each one is doing.

Remote Development: Reattach to agent sessions over SSH.

Autonomous Workflows: Let agents continue operating while you’re away from the machine.

Agent Orchestration: The project provides CLI and socket APIs that agents can use to spawn panes, interact with other agents, and wait for another agent when necessary.



The runtime itself becomes agent-aware.

Productivity Impact

herdr reduces a frustrating class of AI-development problems:

- losing long-running sessions
- manually reconnecting to agents
- losing track of which agent is blocked
- constantly managing terminal windows
- treating every agent task as an interactive session

The bigger shift is conceptual.

Coding agents need infrastructure just like applications do.

Once agents become persistent workers, the terminal is no longer just a place to type commands.

It becomes their runtime.

3. Kimi CODE: An AI Coding Agent That Connects Your Terminal, IDE, and MCP Tools

Repository: *MoonshotAI/kimi-code*

The terminal is already where developers have enormous power.

Git.

Docker.

Node.

Python.







Tests.

Deployment scripts.

Now imagine adding an AI agent that can operate inside that same environment.

That is the idea behind Kimi Code.

What is Kimi Code?

Kimi Code is a terminal-based AI agent designed to help developers perform software-development tasks and terminal operations.

The README describes capabilities including:

- reading code
- editing code
- executing shell commands
- searching and fetching web pages
- planning actions
- adjusting its approach during execution

So this isn't simply a chatbot exposed through a terminal.

It's an **agentic development environment**.

IDE Integration

Kimi Code supports the **Agent Client Protocol (ACP)**.

That means it can work with ACP-compatible editors and IDEs, including Zed and JetBrains tooling. The README shows how Kimi Code can run as an ACP agent server and appear inside an IDE's agent panel.

Now the workflow becomes much more interesting. You aren't forced to choose

MCP Support

Kimi Code also supports Model Context Protocol tools.

The repository documents commands for adding MCP servers through HTTP or stdio transports, managing authentication, listing servers, and supplying MCP configuration files.

That means the agent can move beyond **“Understand my code”** toward **“Understand my code and interact with external tools.”**

And that’s where agent workflows become genuinely powerful.

Real Development Use Cases

Repository Understanding: Ask the agent to investigate unfamiliar codebases.

Implementation: Let the agent modify files and execute development commands.

Debugging: Combine code investigation with terminal execution.

IDE Workflows: Bring the same agent into ACP-compatible environments.

Tool-Augmented Agents: Connect external systems through MCP.

Shell Productivity: Kimi Code can also operate as a shell, allowing developers to switch into shell command mode from inside the environment.

Productivity Impact

The real benefit isn’t just AI-generated code. It’s **context continuity**.

- Your terminal has the repository.
- Your IDE has the files.
- Your MCP servers have tools.

0

That creates a much more natural development loop **Understand → Execute → Inspect → Modify → Verify**

instead of constantly jumping between unrelated interfaces.

💡 ***Enjoying this article?***
Every week day, I publish practical, production-ready deep dives covering Web development, System Design, Open source projects, Tech industry trends and AI Engineering and tools.

Subscribe to Newsletter

4. Documenso: Build Your Own Digital Signature Infrastructure

Repository: *documenso/documenso*

Not every useful open-source repository is about AI and that's exactly why Documenso belongs on this list.

Because software development isn't only about building features. It's also about building the business workflows around those features.

One of the most common? **Getting documents signed.**

What is Documenso?

Documenso describes itself as an **open-source DocuSign alternative**.

Its goal is to provide a document-signing platform that organizations can self-host, inspect, run, and fork. The project frames this as part of building more open "trust infrastructure" around digital signatures.

That's a very different philosophy from simply consuming a hosted SaaS API.

Instead of `Your App → Third-Party Signing Platform`

you can explore `Your App → Your Signing Infrastructure`

Imagine you're building:

- HR software
- procurement software
- contract management
- SaaS onboarding
- freelancer platforms
- real-estate applications
- legal workflows
- enterprise approval systems

At some point, someone needs to sign something. You could build a small signature workflow yourself or you could buy a SaaS service or you could use an open-source platform that you can inspect and self-host.

Documenso gives you that third option.

The Developer Stack

One thing I particularly like about this repository is that its stack is familiar to modern JavaScript developers.

The project currently lists:

- TypeScript
- React Router v7
- Hono
- Prisma
- Tailwind CSS
- shadcn/ui
- Radix UI

   0 

- Playwright
- Stripe
- PDF tooling

For a developer already working in the TypeScript ecosystem, this makes the project particularly interesting to study.

You're not just consuming a product, you're looking at a real-world application architecture.

Real Development Use Cases

SaaS Applications: Add document-signing workflows to your own product.

Internal Enterprise Tools: Self-host signing infrastructure for internal workflows.

Contract Management: Create approval and signature pipelines.

HR Platforms: Handle offer letters and employment documents.

Procurement: Integrate document execution into purchasing workflows.

Learning and Architecture: Study how a production-oriented open-source TypeScript application combines frontend, backend, database, PDF tooling, authentication, payments, and testing.

Productivity Impact

The biggest benefit is **control**.

- You can inspect the implementation.
- You can self-host it.
- You can integrate it with your own infrastructure.

And because it's open source, you have more freedom to understand how the



source, but external pull requests are currently paused except for a small group of trusted contributors; detailed issues are the preferred external contribution path.

Open source doesn't always mean "fork it and modify anything tomorrow." It means you can inspect, run, understand, and build on the software while still respecting the project's current contribution model.

5. MCP Apps: The Missing UI Layer for AI Tools

Repository: *modelcontextprotocol/ext-apps*

This is probably the repository in this week's list that points most directly toward where AI interfaces are heading.

Because we have spent years teaching computers to return **text,** then structured JSON, then tool results.

But sometimes text isn't enough.

Try asking an AI agent to:

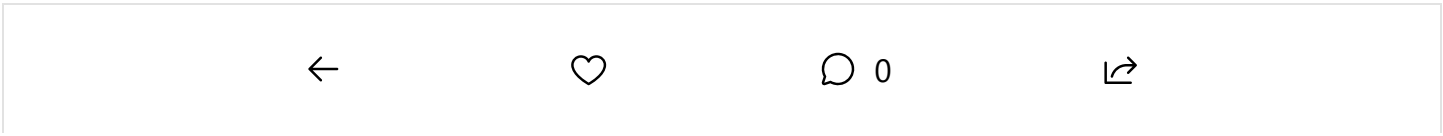
- edit a diagram
- configure a complex form
- explore a chart
- manipulate a visual workflow
- watch a video
- work with a dashboard

Suddenly, a text response feels like the wrong interface.

That's where MCP Apps comes in.

What is MCP Apps?

The official Model Context Protocol repository describes MCP Apps as a



examples such as Claude and ChatGPT. The project describes use cases including charts, forms, dashboards, design canvases, and video players.

This is a big conceptual change.

Traditional MCP `LLM → Tool → Text / Structured Data`

MCP Apps: `LLM → Tool → Interactive UI`

The tool can now become something the user actually interacts with.

How It Works

The architecture is surprisingly elegant.

1. The tool declares a UI resource: A tool can expose a `ui://` resource containing its interface.

2. The model calls the tool: The LLM invokes the MCP server as usual.

3. The host renders the UI: The host fetches the declared resource and places it inside a sandboxed iframe.

4. The UI communicates back: The host can pass data to the UI, while the UI can invoke other tools through the host.

That creates a much richer interaction model.

Real Development Use Cases

Data Visualization: An MCP tool can return an interactive chart instead of a paragraph of numbers.

Forms: The agent can ask the user to fill structured inputs through a real UI.

Design Tools: A tool can render an interactive design surface.

0

The repository includes an approach for converting an existing web application into a hybrid web + MCP App.

And the repository ships Agent Skills that can help coding agents:

- scaffold a new MCP App
- migrate an OpenAI App
- add UI to an existing MCP server
- convert a web app into an MCP App

That's particularly interesting for AI-assisted development. You can increasingly ask your coding agent to build the integration for you.

Productivity Impact

MCP Apps reduces a major limitation of tool-using AI, **tools can finally have an interface.**

Instead of forcing every interaction through natural language, developers can give users the right interaction for the task.

- Text for explanation.
- Forms for input.
- Charts for analysis.
- Canvases for visual work.
- Dashboards for monitoring.

That is much closer to how human-computer interaction already works.

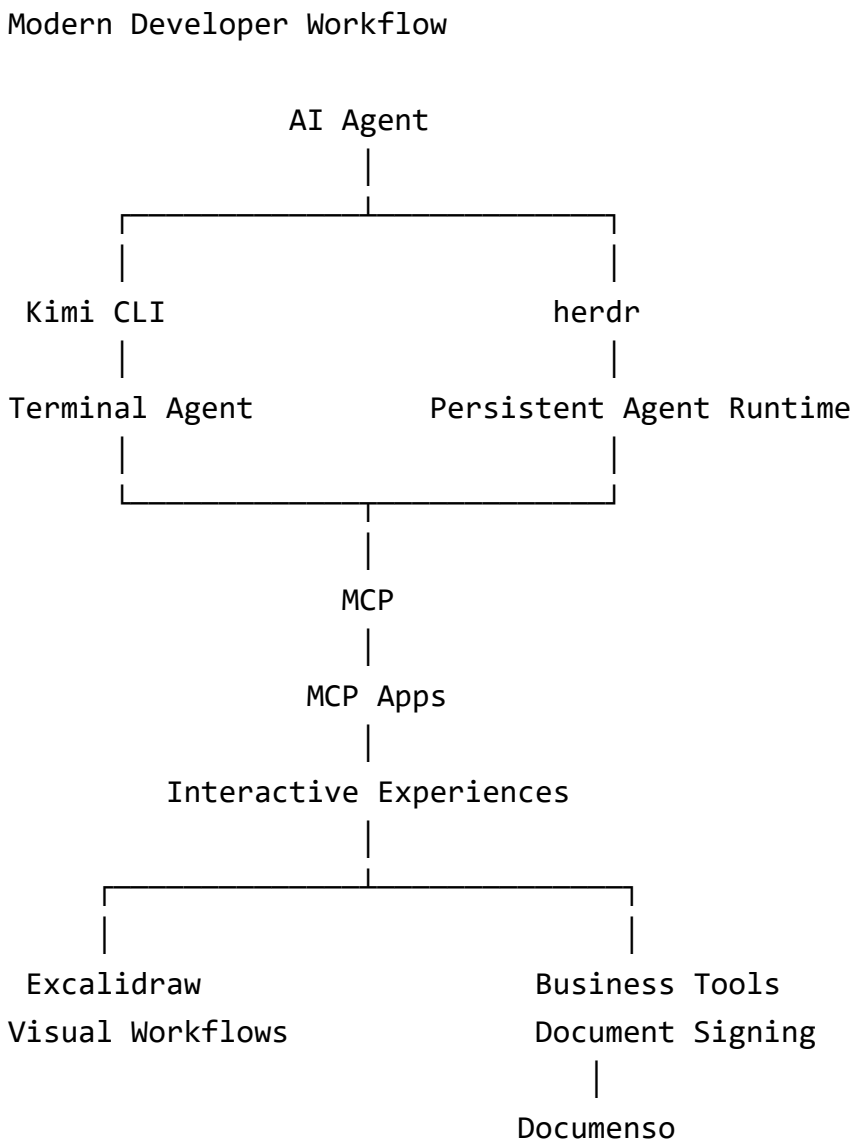
The future of AI interfaces probably isn't "chat replaces UI." It's "chat orchestrates the right UI."

The Bigger Picture: Five Repositories, One Developer Workflow

0

- One draws diagrams.
- One manages coding-agent terminals.
- One runs an AI coding agent.
- One handles digital signatures.
- One adds interactive UIs to MCP tools.

But look at the broader workflow.



Modern AI applications are becoming **multimodal workflows**, not simply chat interfaces.

The agent may need to **reason → execute → visualize → request input → interact with external systems → complete a business workflow**

Each repository in this week's list handles a piece of that puzzle.

Which Repository Should You Try First?

If you're building applications with visual collaboration or diagrams, start with **Excalidraw**.

If you're running multiple coding agents or long-running agent sessions, explore **herdr**.

If you want an AI coding agent that works across terminal, IDE, and MCP workflows, investigate **Kimi CLI** and keep an eye on its evolution toward Kimi Code CLI.

If you're building SaaS or enterprise workflows involving signatures, take a serious look at **Documenso**.

And if you're building MCP-based applications and wondering how to move beyond text-only tools, **MCP Apps** is probably the most important repository in this list to explore.

Final Thoughts

The developer experience is changing.

First, AI helped us **write code**. Then AI started helping us **understand code**. Now AI is beginning to **operate inside our development environments**.

And the next step is bigger **AI will interact with the same rich interfaces humans use.**

   0 

- persistent agent runtimes
- terminal-based agents
- visual canvases
- interactive tool UIs
- business workflows
- open infrastructure

These five repositories are a good snapshot of that transition.

And there is one lesson I keep coming back to **The future developer toolbox won't be defined by a single AI assistant. It will be an ecosystem of open-source building blocks that work together.**

That's what makes repositories like these worth watching, not because every developer needs to install all five, because each one shows us a different piece of where software development is heading.

Thank You for Reading!

I hope you found it helpful and informative. If you have any questions or feedback, feel free to leave a comment below. Your support and engagement mean a lot to me.

Happy Coding!

Reply

Newest first

Add your comment...

Login or Subscribe to participate

Keep Reading

0



8 Open-Source Alternatives to the Jev Model You Can ...

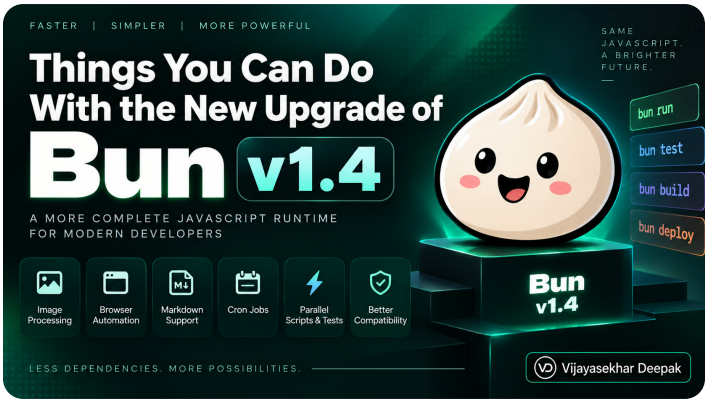
Vijayasekhar Deepak



Sep 28, 2026 • 7 min read

13 Practical Jev Use Cases: Where Fast AI Decisions ...

Vijayasekhar Deepak



Sep 22, 2026 • 5 min read

Things You Can Do With Bun 1.4: The JavaScript ...

Vijayasekhar Deepak

View more

Vijayasekhar Deepak | FullStack JS & AI Newsletter

Get practical Next.js architecture breakdowns, learn to build with

Email Subscribe

0

practical read. Join 600+ tech professionals building the future of the web with Vijayasekhar Deepak's Newsletter.

© 2026 Vijayasekhar Deepak | FullStack JS & AI Newsletter.

Powered by beehiiv




0
