Member-only story

My Terminal AI Sidekick: Building Djin To Reclaim Coding Time

Building a terminal AI to manage Jira, time tracking, and tasks without context switching

Patrick Kalkman · Following

11 min read · 3 hours ago

6

Reclaim your coding time from admin tasks with Djin, image by author.

Your productivity stack is lying to you.

They promised to make you more productive. More organized. More efficient. But let's be honest: They've turned you into a highly paid bureaucrat who occasionally writes code.

Don't believe me?

Look at your browser tabs. Chances are, work-related tools, like project trackers such as Jira, are taking up quite a few. You probably have shared documents open that need attention, and maybe some other productivity app that feels like a chore.

Don't just take my word for it. Research by Zenhub in 2022 found that in a typical 40-hour week, most developers log under 20 hours of actual coding time. The rest? Lost to the bureaucratic black hole.

As one developer put it during their research:

> *Interruptions during the workday are a big problem for coding productivity. I often find myself coding in the evening because then all the meetings are over.*

That's not productivity — that's what happens when tools control the developer, not the other way around.

We've been sold this lie that "good developers" carefully document everything, track every minute, and slice user stories into perfectly sized chunks… all while maintaining their coding flow.

I got sick of this. So I built Djin, my personal AI developer assistant. It lives in the terminal — you know, where you actually work — and handles all that management overhead without making you play tab-switching pinball.

Need to update a ticket?

Just tell Djin and keep coding. Breaking down stories? Tracking time? Writing acceptance criteria? One command and you're done. No context switching. No browser tabs.

No more pretending that interrupting your flow state is "best practice."

In this article, I'll show you how I built Djin, and why the future of developer productivity isn't about cramming more tools into your browser.

It's about bringing everything into your natural workflow.

I'll walk you through Djin step-by-step. We'll start with a quick demo so you can see *what* it does.

Then, we'll explore the *why* — the Vertical Slice Architecture that makes integrating AI features so much easier.

Finally, we'll get into the *how*, looking at the key implementation details under the hood.

Ready to code again? Let's burn down the bureaucracy.

Watch Djin in action: Coding without the admin headache

I could go on and on about how Djin saves me from the woes of productivity tools, but you have to see it for yourself to believe it.

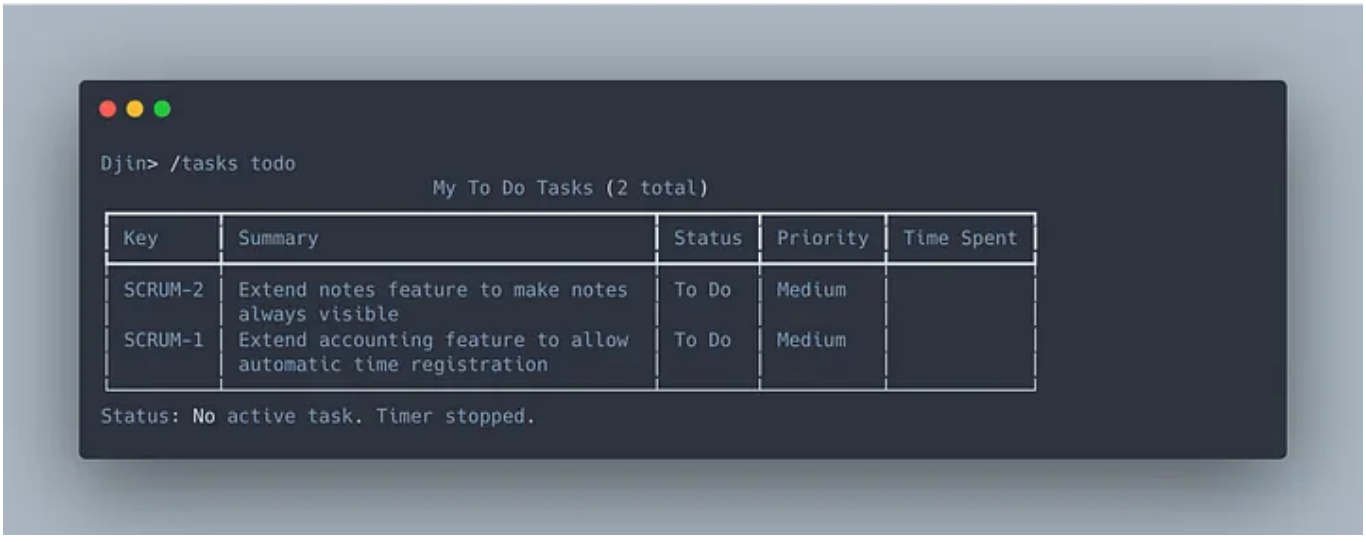
Check out this quick demo below.

Djin Showcase: Streamlining tasks and time logging via CLI, video by author.

Did you catch what happened there? Let me break it down.

Finding work without leaving your code

No more alt-tabbing to Jira or whatever task tracker you're using. I simply typed: `/tasks todo`

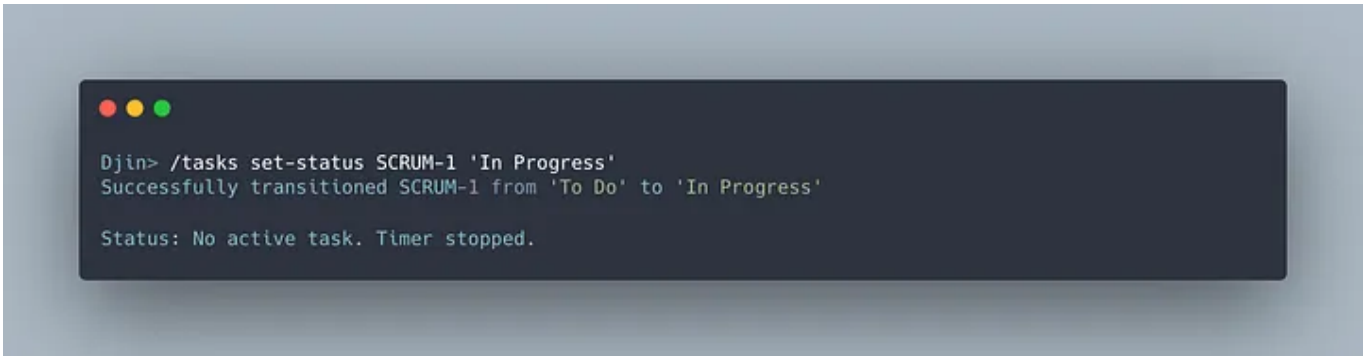


Your Jira tasks at your fingertips — view your to-do list without ever leaving the terminal, image by author

There is no context switch. You don't have to wait for browser tabs to load. It's just me and my code.

Updating status without the usual friction

Notice how I changed a task status with a single command? No clicking through three different menus or waiting for page refreshes.

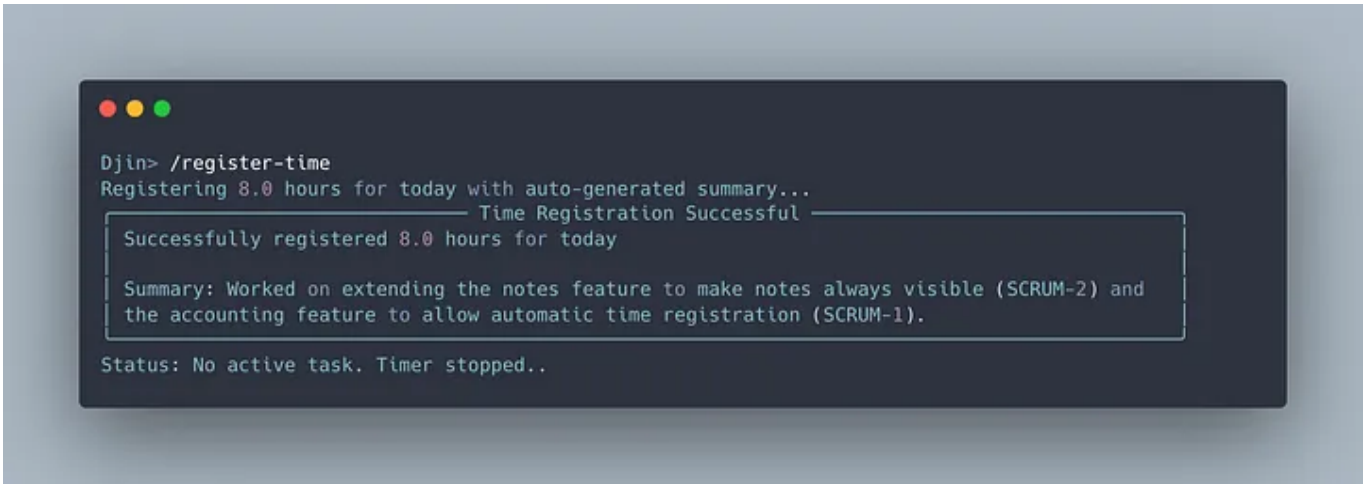


Djin CLI: Seamlessly updating ticket status from your terminal — no more context switching, image by author

That's it. Task updated, and I never stopped coding.

The magic of automated time tracking

The best part? Watch how I logged my hours at the end. That used to be a Friday afternoon nightmare — trying to remember what I worked on and for how long. Now Djin handles it automatically.



Automate time tracking with smart summaries — eliminating timesheet entries, image by author

It pushed everything to my accounting system without me filling out a single form. The hours I spent actually writing code got tracked in the background while I worked.

The real problem Djin solves

This isn't just about saving a few clicks. It's about protecting your mental bandwidth.

Every time you switch contexts — from code to browser to Jira to Slack and back — you're burning mental energy that should go toward solving actual problems. Research shows it takes 23 minutes to fully recover from an interruption. Twenty-three minutes!

Djin isn't just a cool terminal tool. It's a shield against the constant interruptions that fragment our workday into useless little chunks.

Seeing Djin streamline those common tasks is satisfying. But making it work reliably and ensuring it could grow required a solid foundation. That led me to a key architectural decision.

Djin's architecture: Why vertical slice architecture was the perfect choice

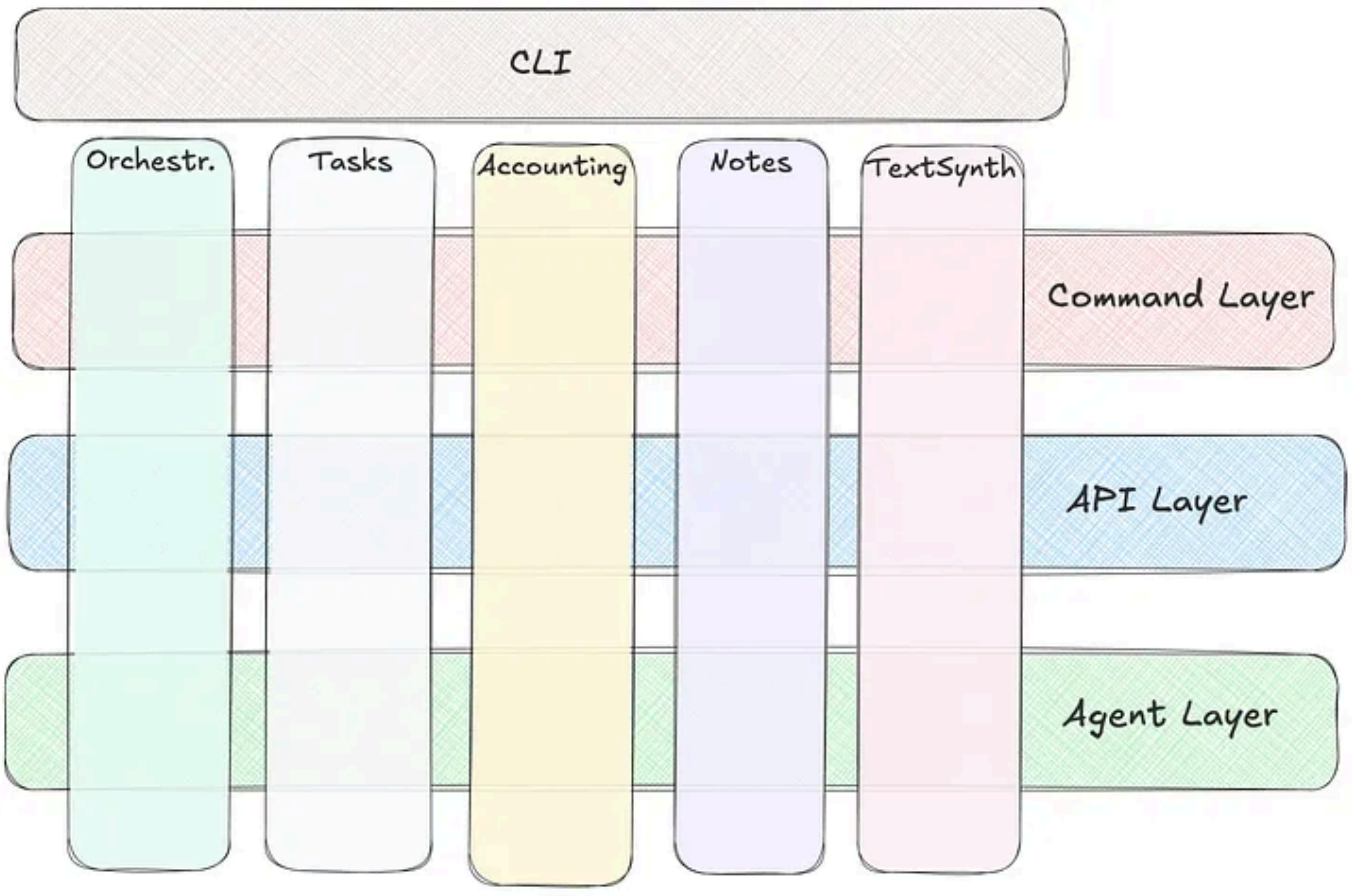
Ever felt overwhelmed by traditional layered architecture, especially with larger projects? You know, the kind where you split your app into neat horizontal layers, but it still somehow becomes a tangled mess?

I've been there. That's why I want to share a decision that transformed how I built Djin, my AI-powered developer assistant: embracing Vertical Slice Architecture (VSA).

What makes VSA different?

Picture your typical layered architecture like a cake — each layer can only talk to the ones below it. Now, imagine slicing that cake vertically. That's VSA.

Instead of spreading features across layers, each feature gets its own complete slice from top to bottom.



Djin's vertical slices architecture (VSA), image by author.

And trust me, this simple change transforms how you build software.

Here's why VSA is such a game-changer:

1. **Feature Focused:** When you make changes, they stay in their lane. No more surprise bugs popping up in seemingly unrelated parts of your app!
2. **Scalability:** Growing team? No problem. Different developers can own different features without constantly bumping into each other's code.
3. **Maintainability:** Finding your way around the code becomes a breeze since everything you need for a feature lives in one place.

Plus, it fits perfectly with how modern development actually works — those agile sprints where you're building one feature at a time? They were made for each other.

But here's where it gets really interesting — VSA turns out to be perfect for working with AI tools. More on that game-changing aspect in a bit.

Breaking down Djin's slices

Let me walk you through how this works in practice. Each slice in Djin has three main parts:

1. The Command Layer — This is where the magic starts. Take the Tasks feature, for example. When you type `/tasks todo`, this layer handles that command and gets things moving.
2. The API Layer — Think of this as the engine room. It is what the Commands use to perform their function. It is also the layer where overarching features may interact with other features.
3. The Agent Layer — This is where the AI brains live. Using the LangGraph framework, each feature gets its own AI agent that knows exactly what it

needs to do.

Why this architecture is perfect for AI developer tools

Remember my deep dive into finding the sweet spot in the AI developer tools triangle?

Well, here's where that insight pays off big time.

You know that frustrating moment when your AI coding assistant seems lost in your codebase, suggesting solutions that just don't fit? That's usually because it's missing crucial context. But here's the beauty of VSA — it solves this headache in one elegant sweep.

Think about it: having all your features code in one place is like giving your AI assistant the perfect cheat sheet.

Take my Tasks feature, for example. When I'm working on it, my AI helper can see everything at once — from the command structure right down to how the AI agent thinks.

It's like having a dev partner who actually gets what you're trying to build.

No more explaining the same thing twice. No more piecing together scattered code. Just smooth, contextual collaboration.

The real-world benefits

Building Djin with VSA has been a game-changer because:

- Adding new features is a breeze — each one gets its own space and doesn't mess with existing code

- The codebase stays clean and makes sense — no more hunting through layers to find what you need
- AI tools work more effectively — they get all the context they need in one place
- The command-line interface fits naturally into the structure — perfect for VS Code integration

Looking ahead

As Djin grows, VSA keeps proving its worth. Want to experiment with a new language model? Add another AI capability? Expand the command-line features?

Each addition has its natural home in the architecture.

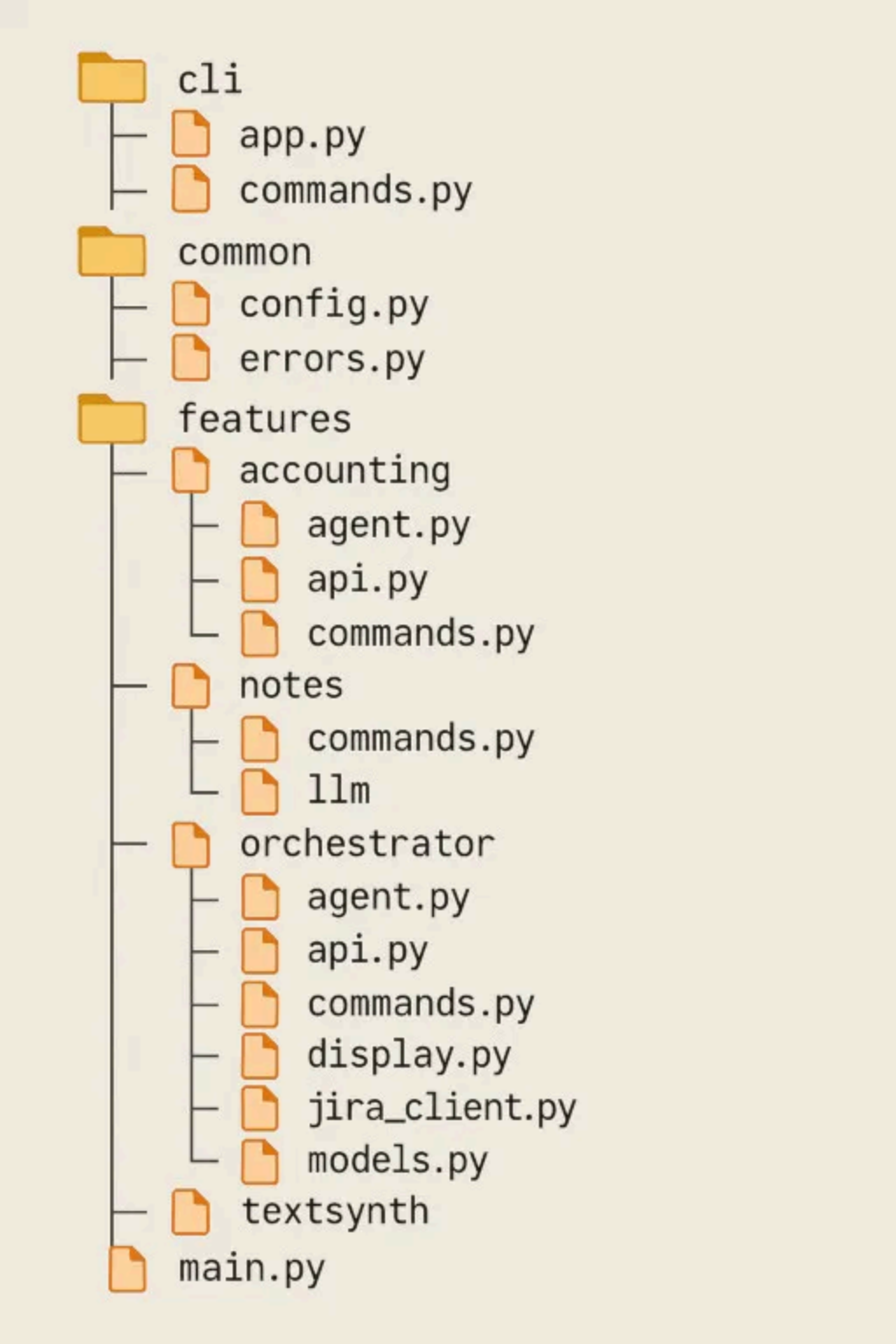
If you're using AI-powered tools, I can't recommend VSA enough. It's not just about keeping your code organized — it's about creating an architecture that works in harmony with modern AI development tools.

And in today's rapidly evolving tech landscape, that's more valuable than ever.

Deep dive: The secret sauce behind Djin's AI magic

Remember that Vertical Slice Architecture I mentioned earlier? Here's where it gets real. Instead of splitting things by technical layers, each feature in Djin gets its own complete slice from top to bottom.

Let me show you what I mean:



Djin's File Structure: Clean organization of features in vertical slices, image by author.

See those folders? Each one represents a unique part of Djin's brain. Instead of cramming everything into one giant file (we've all been there!), Djin splits things up in a way that mirrors how you actually work.

Ever wondered how Djin actually works its magic? Let me take you behind the curtains and show you the most interesting parts that make it tick.

I'll focus on three game-changing aspects that make Djin different from traditional developer tools.

The AI agent architecture: Your personal dev team

At its core, Djin isn't just one AI — it's a team of specialized AI agents working together. Think of it like having a group of highly focused assistants, each with their own expertise:

```
class OrchestratorAgent:
  def __init__(self):
    self._task_api = get_tasks_api()
    self._textsynth_api = TextSynthAPI()

  def generate_work_summary(self, date_str: Optional[str] = None) -> str:
    # Get tasks worked on for the day
    worked_on_data = self._task_api.get_worked_on_tasks(date_str)
    tasks = worked_on_data.get("tasks", [])

    # Let the AI summarize your work
    summary = self._textsynth_api.summarize_tasks(tasks)
    return summary
```

This isn't just about splitting up code — it's about having different AI personalities working together.

The `TaskAgent` knows all about managing your Jira tasks, while the `TextSynthAgent` is your writing buddy that turns your day's work into clear summaries.

The `OrchestratorAgent` is like your personal assistant, coordinating everything.

Context-aware time tracking: No more Friday panic

Remember those Friday afternoons spent trying to remember what you worked on? Here's how Djin tackles this:

```
def register_time_with_summary(self, date_str: Optional[str] = None, hours: floa
  # Generate a smart summary of your work
  summary = self.generate_work_summary(date_str)

  # Register hours automatically in MoneyMonk
  accounting_api = get_accounting_api()
  registration_result = accounting_api.register_hours(
      date=date_str or datetime.now().strftime("%Y-%m-%d"),
      description=summary,
      hours=str(hours)
  )
```

The LangGraph framework: Making AI interactions natural

The real breakthrough in Djin is how it understands what you mean, not just what you say.

Here's a peek at how it processes your commands:

```
def create_task_fetching_graph():
  """Create a graph for fetching and processing tasks"""
```

```
    workflow = StateGraph(TaskState)

    # Add nodes that form the conversation flow
    workflow.add_node("fetch_tasks", fetch_tasks_node)
    workflow.add_node("process_tasks", process_tasks_node)
    workflow.add_node("format_output", format_output_node)

    # Define how the conversation should flow
    workflow.set_entry_point("fetch_tasks")
    workflow.add_edge("fetch_tasks", "process_tasks")
    workflow.add_edge("process_tasks", "format_output")

    return workflow.compile()
```

This is where LangGraph comes in — it's not just about matching commands to functions. It's about understanding the flow of a natural conversation.

Why this matters for you

All this technical talk is cool, but here's why it matters: Djin isn't just another tool that saves a few clicks. It's a fundamental rethink of how developers should interact with their tools.

Every time you switch context (like when you alt-tab to Jira), you're not just losing seconds — you're dropping out of that precious flow state where your best code happens.

By keeping everything in the terminal and using AI to handle the cognitive overhead, Djin lets you stay in that zone.

Think about it: When was the last time you had a truly uninterrupted coding session? That's what Djin is fighting to protect.

Want to dive even deeper? Check out the full source code where you can see each component in detail.

So, how did Djin come about? It wasn't a grand plan initially. It was born from frustration.

Reclaiming the joy of development

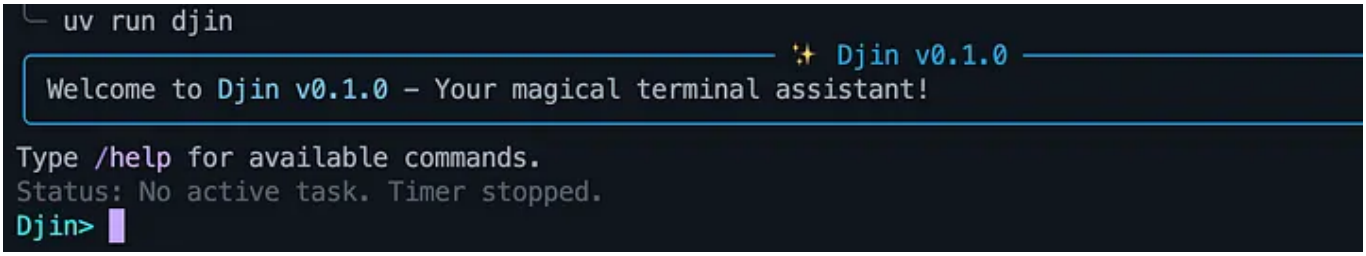
Let me tell you the authentic story behind Djin. Like many developers, I was drowning in Jira tasks and context switching. My first attempt at fixing this? A Telegram bot. I figured it would be easier to manage everything through chat.

Big mistake.

Instead of jumping between browser tabs, I was now bouncing between my terminal and Telegram. Same problem, different app. I needed something better.

That's when I discovered Aider, a terminal-based AI coding tool. It opened my eyes to a simple truth: if we spend most of our time in the terminal anyway, why leave it at all?

So I scrapped the Telegram bot and rebuilt everything. Djin was born as a true terminal companion, handling my Jira tasks without making me switch contexts.



Meet Djin v0.1.0: Your friendly AI terminal assistant that makes work feel like magic, image by author.

What started as a way to save my sanity has grown into something bigger: a

setup. It's my digital Swiss Army knife, built for my specific needs. But that's exactly why I made it open source.

- Connecting to your Jira instance: This is a simple configuration change — just point Djin to your Jira URL and credentials via `uv run djin config`
- Integrating a different task management system: VSA makes this straightforward. The Jira integration code lives entirely within the 'Tasks' feature slice (specifically `jira_client.py`). Supporting tools like Asana or Trello just means replacing this file with a new client for their specific API.

What's next for Djin

The current version is just the beginning. Here's what I'm cooking up next:

- Always-on Jira monitoring that instantly flags new tickets assigned to you
- Background task tracking that catches approaching deadlines before they sneak up on you
- Automatic Jira updates based on your commit messages (no more manual status updates!)
- Implementation of the Model Context Protocol for seamless integration between LLM and tools
- Better integrations with other dev tools beyond Jira

Each feature has one goal: keeping you in the flow state where you do your best work. No context switching, no surprise interruptions — just smooth, focused coding.

The future of developer productivity

We're at a turning point in software development. The old way — juggling dozens of tools and constantly context-switching — is unsustainable.

The future isn't about adding more tools to your stack. It's about having intelligent assistants that bring your tools to you, right where you're working.

Djin is my contribution to that future. While I built it for my own needs with my company's Jira setup, the core idea is universal. Your development environment should work for you, not against you.

As always, the code is open source, check it out, adapt it to your workflow, or just share your thoughts on how we can make development more focused and enjoyable.

Check out Djin on GitHub and start imagining a world with fewer context switches and more actual coding.

Python | Artificial Intelligence | Developer Productivity | Software Development | Terminal

Written by Patrick Kalkman

3.5K Followers · 402 Following

Following

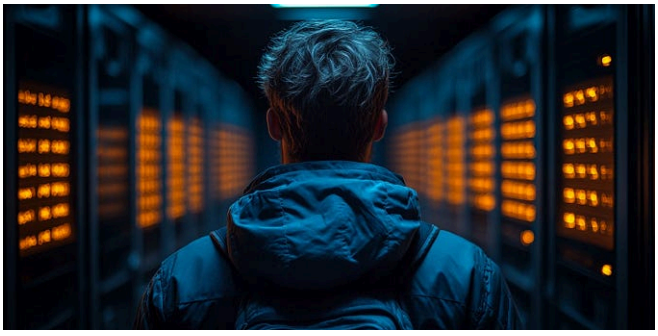
IT Architect, Developer & Writer focused on open-source, AI, LLMs & AI agents. Embracing agile methodologies, lifelong learning, and knowledge sharing.

No responses yet

Alex Mylnikov

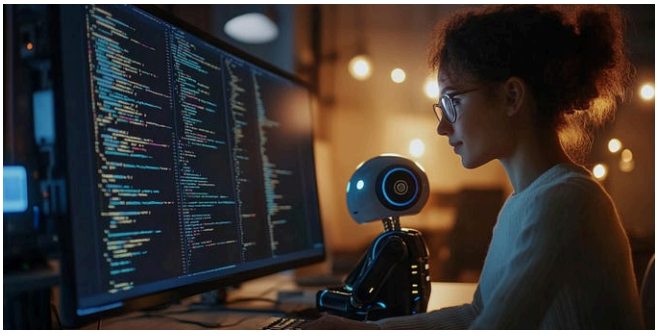
What are your thoughts?

More from Patrick Kalkman



In Level Up Coding by Patrick Kalkman

The Friendly Hacker Who Saved Our Company



In Generative AI by Patrick Kalkman

Good News, Devs: AI Makes You More Valuable

5 Essential AI skills to boost your career today

One careless checkbox, 1,000 exposed customer devices, and the security breach...

Mar 4 223 6

Mar 19 101

In Generative AI by Patrick Kalkman

Watch 5 AI Frameworks Evolve: A Stunning Visual History

See how open-source AI gets built, commit by commit

Mar 25 65

In Better Programming by Patrick Kalkman

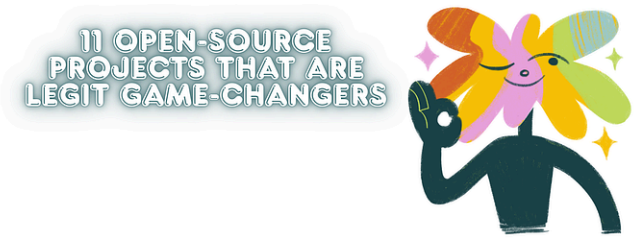
Python Persistence Perfected

Optimizing data storage with the repository and unit of work patterns

Apr 10, 2023 305 3

See all from Patrick Kalkman

Recommended from Medium

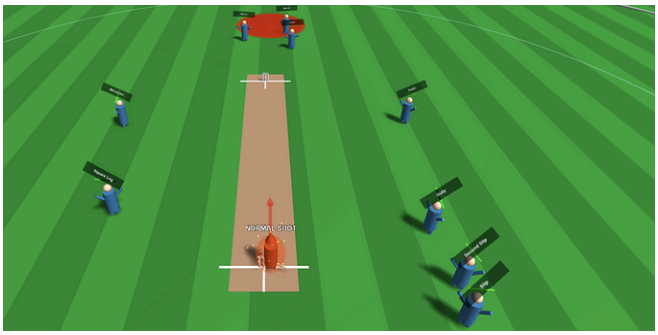


In Let's Code Future by Let's Code Future

11 Open-Source Projects That are Legit Game-Changers

Really unique and valuable

Mar 24 949 14

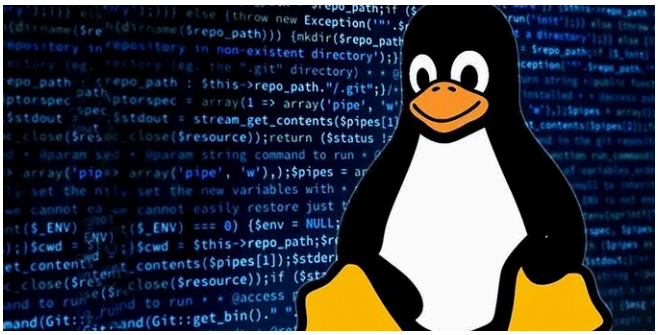


Chris Dunlop

Cursor Rules with Examples, the secret trick to building bigger and...

If you have been using Cursor, then I recommend you learn about Cursor Rules.

Mar 12 34 1



In InfoSec Write-ups by Frost

Powerful Linux Tricks That Will Change Your Life

If you've ever worked in a Linux environment, you know how powerful and versatile it can...

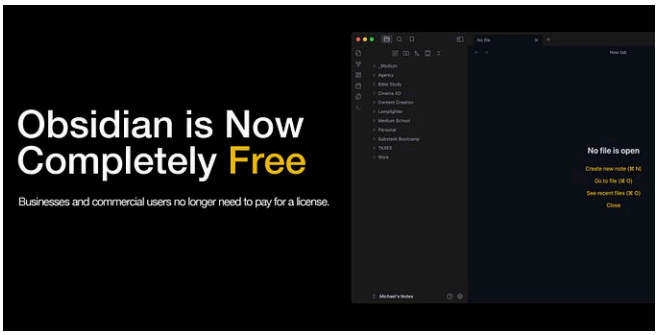
Mar 12 389 7

Yong kang Chia

Maximizing Your Cursor Use: Advanced Prompting, Cursor...

Productivity Tips with cursor

Feb 20 37 1



Eduard Ruzga

Claude with MCPs Replaced Cursor & Windsurf—How Did That...

You can see in the screenshot that I was using Windsurf in December. But by January and...

Feb 26 1.6K 36

Michael Swengel

Obsidian is Now Completely FREE to Use—Even for Businesses

But there are other ways to support Obsidian if you want

Feb 24 239 6

See more recommendations