datasette-ripgrep: deploy a regular expression search engine for your source code

28th November 2020

This week I built datasette-ripgrep—a web application for running regular expression searches against source code, built on top of the amazing ripgrep command-line tool.

datasette-ripgrep demo

I've deployed a demo version of the application here:

ripgrep.datasette.io/-/ripgrep?pattern=pytest

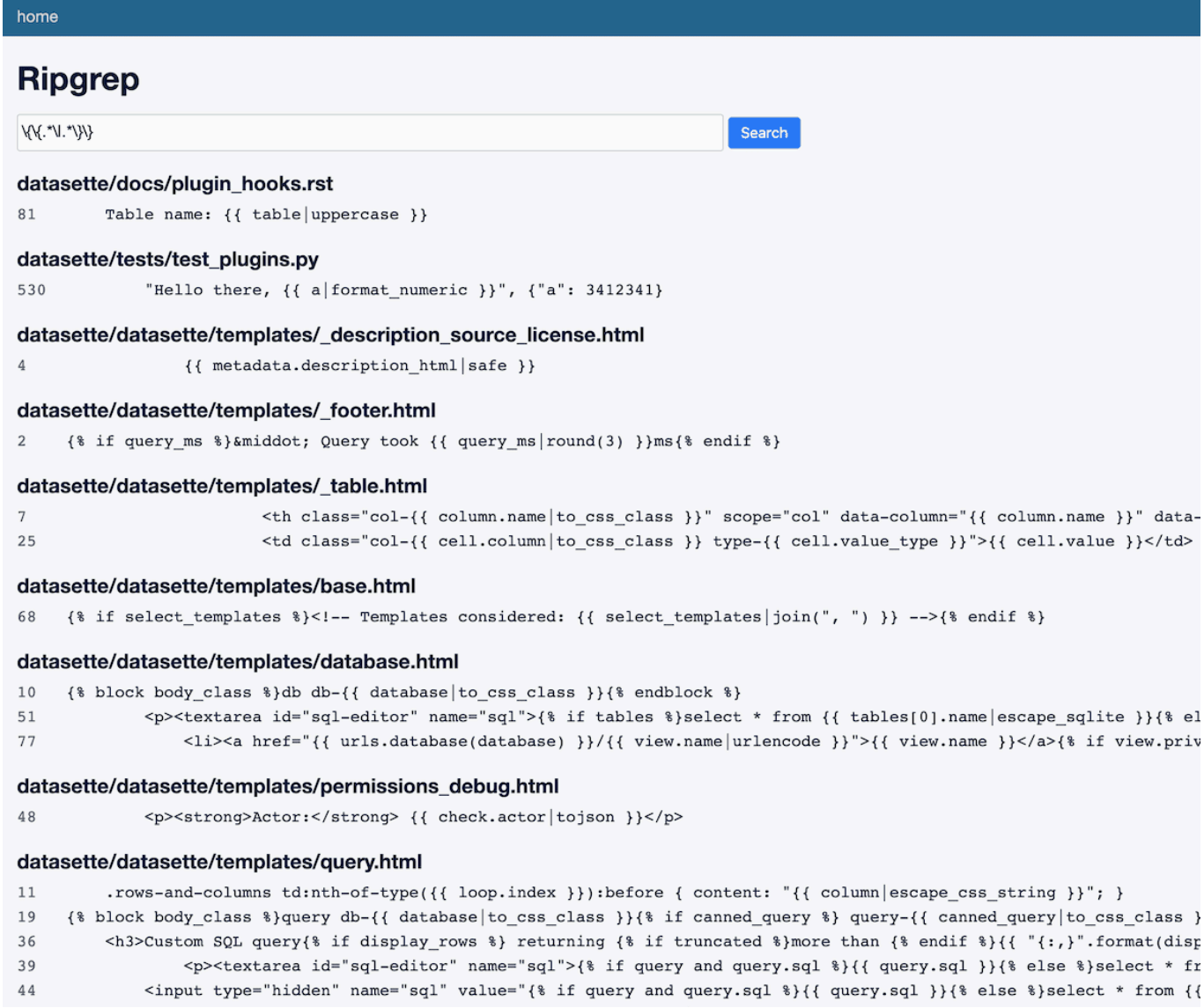
The demo runs searches against the source code of every one of my GitHub repositories that start with `datasette`—61 repos right now—so it should include all of my Datasette plugins plus the core Datasette repository itself.

Since it's running on top of `ripgrep`, it supports regular expressions. This is absurdly useful. Some examples:

- Every usage of the `.plugin_config(` method: plugin_config\(
- Everywhere I use `async with httpx.AsyncClient` (usually in tests): async with.*AsyncClient
- All places where I use a Jinja `|` filter inside a variable: \{\{.*\|.*\}\}

I usually run ripgrep as `rg` on the command-line, or use it within Visual Studio Code (fun fact: the reason VS Code's "Find in Files" is so good is it's running ripgrep under the hood).

So why have it as a web application? Because this means I can link to it, bookmark it and use it on my phone.



Why build this?

There are plenty of great existing code search tools out there already: I've heard great things about livegrep, and a quick Google search shows a bunch of other options.

Aside from being a fun project, `datasette-ripgrep` has one key advantage: it gets to benefit from Datasette's publishing mechanism, which means it's really easy to deploy.

That ripgrep.datasette.io demo is deployed by checking out the source code to be searched into a `all` directory and then using the following command:

```
datasette publish cloudrun \
    --metadata metadata.json \
    --static all:all \
    --install=datasette-ripgrep \
    --service datasette-ripgrep \
    --apt-get-install ripgrep
```

`all` is a folder containing the source code to be searched. `metadata.json` contains this:

```
{
    "plugins": {
        "datasette-ripgrep": {
            "path": "/app/all",
```

```
            "time_limit": 3.0
        }
    }
}
```

That's all there is to it! The result is a deployed code search engine, running on Google Cloud Run.

(If you want to try this yourself you'll need to be using the just-released Datasette 0.52.)

The GitHub Action workflow that deploys the demo also uses my github-to-sqlite tool to fetch my repos and then shallow-clones the ones that begin with `datasette`.

If you have your own Google Cloud Run credentials, you can run your own copy of that workflow against your own repositories.

A different kind of Datasette plugin

Datasette is a tool for publishing SQLite databases, so most Datasette plugins integrate with SQLite in some way.

`datasette-ripgrep` is different: it makes no use of SQLite at all, but instead takes advantage of Datasette's URL routing, `datasette publish` deployments and permissions system.

The plugin implementation is currently 134 lines of code, excluding tests and templates.

While the plugin doesn't use SQLite, it does share a common philosophy with Datasette: the plugin bundles the source code that it is going to search as part of the deployed application, in a similar way to how Datasette usually bundles one or more SQLite database files.

As such, it's extremely inexpensive to run and can be deployed to serverless hosting. If you need to scale it, you can run more copies.

This does mean that the application needs to be re-deployed to pick up changes to the searchable code. I'll probably set my demo to do this on a daily basis.

Controlling processes from asyncio

The trickiest part of the implementation was figuring out how to use Python's `asyncio.create_subprocess_exec()` method to safely run the `rg` process in response to incoming requests.

I don't want expensive searches to tie up the server, so I implemented two limits here. The first is a time limit: by default, searches have a second to run after which the `rg` process will be terminated and only results recieved so far will be returned. This is achieved using the asyncio.wait_for() function.

I also implemented a limit on the number of matching lines that can be returned, defaulting to 2,000. Any more than that and the process is terminated early.

Both of these limits can be customized using plugin settings (documented in the README). You can see how they are implemented in the async def run_ripgrep(pattern, path, time_limit=1.0, max_lines=2000) function.

Highlighted linkable line numbers

The other fun implementation detail is the way the source code listings are displayed. I'm using CSS to display the line numbers in a way that makes them visible without them breaking copy-and-paste (inspired by this article by Sylvain Durand).

```
code:before {
    content: attr(data-line);
    display: inline-block;
    width: 3.5ch;
    -webkit-user-select: none;
    color: #666;
}
```

The HTML looks like this:

```
<pre><code id="L1" data-line="1">from setuptools import setup</code>
<code id="L2" data-line="2">import os</code>
<code id="L3" data-line="3"> </code>
<code id="L4" data-line="4">VERSION = &#34;0.1&#34;</code>
...
```

I wanted to imitate GitHub's handling of line links, where adding `#L23` to the URL both jumps to that line and causes the line to be highlighted. Here's a demo of that—I use the following JavaScript to update the contents of a `<style id="highlightStyle"></style>` element in the document head any time the URL fragment changes:

```
<script>
var highlightStyle = document.getElementById('highlightStyle');
function highlightLineFromFragment() {
    if (/^#L\d+$/.exec(location.hash)) {
        highlightStyle.innerText = `${location.hash} { background-color: yellow; }`;
    }
}
highlightLineFromFragment();
window.addEventListener("hashchange", highlightLineFromFragment);
</script>
```

It's the simplest way I could think of to achieve this effect.

Update 28th November 2020: Louis Lévêque on Twitter suggested using the CSS :target selector instead, which is indeed MUCH simpler—I deleted the above JavaScript and replaced it with this CSS:

```
:target {
    background-color: #FFFF99;
}
```

Next steps for this project

I'm pleased to have got datasette-ripgrep to a workable state, and I'm looking forward to using it to answer questions about the growing Datasette ecosystem. I don't know how much more time I'll invest in this—if it proves useful then I may well expand it.

I do think there's something really interesting about being able to spin up this kind of code search engine on demand using `datasette publish`. It feels like a very useful trick to have access to.

Better URLs for my TILs

My other project this week was an upgrade to til.simonwillison.net: I finally spent the time to design nicer URLs for the site.

Before:

```
til.simonwillison.net/til/til/javascript_manipulating-query-params.md
```

After:

```
til.simonwillison.net/javascript/manipulating-query-params
```

The implementation for this takes advantage of a feature I sneaked into Datasette 0.49: Path parameters for custom page templates. I can create a template file called `pages/{topic}/{slug}.html` and Datasette use that template to handle 404 errors that match that pattern.

Here's the new pages/{topic}/{slug}.html template for my TIL site. It uses the `sql()` template function from the datasette-template-sql plugin to retrieve and render the matching TIL, or raises a 404 if no TIL can be found.

I also needed to setup redirects from the old pages to the new ones. I wrote a TIL on edirects for Datasette explaining how I did that.

TIL this week

- Redirects for Datasette

Releases this week

- datasette-ripgrep 0.2—2020-11-27
- datasette-ripgrep 0.1—2020-11-26
- datasette-atom 0.8.1—2020-11-25
- datasette-ripgrep 0.1a1—2020-11-25
- datasette-ripgrep 0.1a0—2020-11-25
- datasette-graphql 1.2.1—2020-11-24

Posted 28th November 2020 at 6:51 am · Follow me on Mastodon or Twitter or subscribe to my newsletter

More recent articles

- Three major LLM releases in 24 hours (plus weeknotes) - 10th April 2024
- Building files-to-prompt entirely using Claude 3 Opus - 8th April 2024
- Running OCR against PDFs and images directly in your browser - 30th March 2024

- llm cmd undo last git commit - a new plugin for LLM - 26th March 2024
- Building and testing C extensions for SQLite with ChatGPT Code Interpreter - 23rd March 2024
- Claude and ChatGPT for ad-hoc sidequests - 22nd March 2024
- Weeknotes: the aftermath of NICAR - 16th March 2024
- The GPT-4 barrier has finally been broken - 8th March 2024
- Prompt injection and jailbreaking are not the same thing - 5th March 2024
- Interesting ideas in Observable Framework - 3rd March 2024

async 39 | css 145 | projects 366 | python 917 | datasette 388 | weeknotes 178 | ripgrep 5 | bakeddata 9

Next: Weeknotes: github-to-sqlite workflows, datasette-ripgrep enhancements, Datasette 0.52

Previous: Weeknotes: datasette-indieauth, datasette-graphql, PyCon Argentina

> *datasette-ripgrep: deploy a regular expression search engine for your source code*
>
> *https://t.co/vBMvQy3SG0* — Simon Willison (@simonw) November 28, 2020