# Software Engineering Misc

Cover image by AltumCode

## Security

- The OWASP API Top 10 security measures may be a good place to start when trying to decide what security to implement on your web project

# Gitea and DroneCI

I use Gitea to self-host my code + projects here and I use DroneCI, a lightweight CI pipeline that integrates into gitea to do automation stuff.

## Configuration of Drone + Gitea

TODO: write about setup here - docker etc

## Drone CI Config

Drone works like many other CI systems - use yaml files in the repository to control builds. You can specify the type of CI run at the top level of the yaml doc and also give it a name:

```
kind: pipeline
type: docker
name: test and build
```

You can define multiple build steps - each one can have its own docker image - this is useful for example if you have a React frontend and a Golang backend and you need to build both

```
steps:
  - name: test_backend
    image: python:3.7
    commands:
     - pip install poetry
     - poetry install
     - poetry run pytest

  - name: test_frontend
    image: node
    commands:
     - npm install
     - npm test
```

## Shared State Between Steps

Steps can share files via temporary volumes if needed but are generally stateless and independent.

## Conditional Execution of Steps

You can opt to only run steps under certain conditions. For example you might only want to publish code when stuff gets pushed to master. Use conditions to do this:

```
  - name: publish
    when:
      branch:
       - master
      event:
       exclude:
       - pull_request
    image: python3.7
    commands:
     - twine upload...
```

The drone documentation has a full set of conditions that you can use - you can also whitelist and blacklist certain events and certain branches

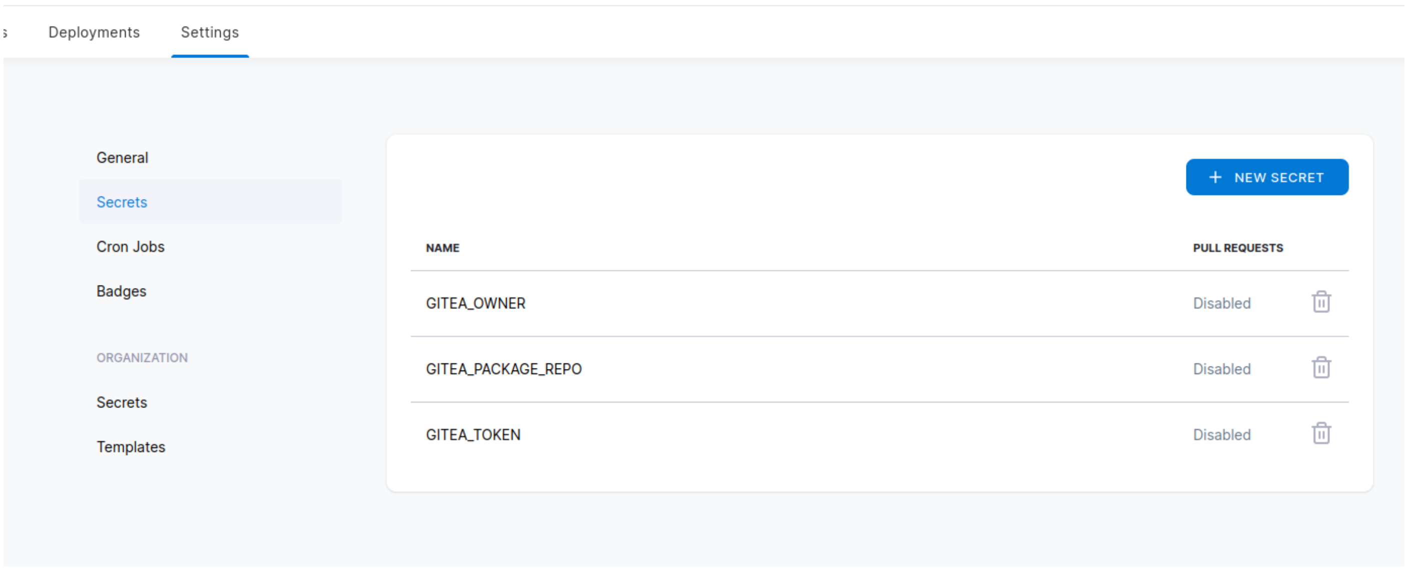
## Secrets

Secrets can be used to pass things like auth tokens into CI pipelines. This is useful if you want to do things like publish packages or upload files. You essentially declare an environment variable and then define which "secret" it came from in your CI yaml file:

```
- name: publish
    when:
      branch:
       - master
      event:
       exclude:
       - pull_request
    image: python3.7
    environment:
```

```
GITEA_PACKAGE_REPO:
  from_secret: gitea_package_repo
GITEA_OWNER:
  from_secret: gitea_owner
GITEA_TOKEN:
  from_secret: gitea_token
```

Then in the drone frontend you can add the value and it will be stored securely and passed to the CI at run time:



| NAME | PULL REQUESTS |
|---|---|
| GITEA_OWNER | Disabled |
| GITEA_PACKAGE_REPO | Disabled |
| GITEA_TOKEN | Disabled |

## Backup Mechanism

# Caddy Handler

You can set up caddy to do different things depending on the prefix using handle directive

```
testsite.com {
  handle_path /media* {
    root * /path/to/media
    file_server
  }

  handle {
    root * /path/to/normal/content
    file_server
  }
}
```

# FOSS Funding

Underfunding of FOSS projects can be disasterous as this list shows.

# Golang Web Services and Gin

I've been using GoLang to build IndieScrobble

## Live Reload

I use this package to live-reload my application as I make changes to it.

# CRON No MTA installed discarding output

Answer from here

> Linux uses mail for sending notifications to the user. Most Linux distributions have a mail service including an MTA (Mail Transfer Agent) installed. Ubuntu doesn't though.
>
> You can install a mail service, postfix for example, to solve this problem.
>
> ```
> sudo apt-get install postfix
> ```
>
> Or you can ignore it. I don't think the inability of cron to send messages has anything to do with the CPU spike (that's linked to the underlying job that cron is running). It might be safest to install an MTA and then read through the messages (`mutt` is a good system mail reader).

The [best option](https://askubuntu.com/a/804289) seems to be redirect all output to a log file:
> (use `sudo` if the issue is with root’s crontab) and add `>> */some/log/file* 2>&1` after every command, like this:

```
0 3 * * * <em>cmd</em>  >> <em>/some/log/file</em> 2>&1
```

# Low and No Code Frontends

Quite often it is useful to have ugly-but-functional frontends for accessing things like databases and carrying out user management. Recently commercial tools like Retool have made it really easy to build this kind of thing by providing drag-and-drop UI builders that are reminiscent of the Visual Studio tooling that we had in the late 90s and early 00s.

## Appsmith

Appsmith is a FOSS low-code app builder. IT can connect with a variety of data sources and apps can be exported to git

# RSync

RSync is a FOSS file copying/syncing tool that has a number of uses and can be used to sync via SSH.

## Preserving User Permission in Rync

https://brainsteam.co.uk/2024/01/03/migrating-users-across-servers-with-rsync/

## Syncing with Non-Standard SSH Ports

It's generally good practice to run SSH services on non-standard ports so that they can't be (as) easily port-scanned and attacked. If you need to use RSync with a non standard port you can tell it any extra ssh arguments it needs to know about via the `-e` argument as explained in this article ( mirror):

```
rsync -arvz \
  -e 'ssh -p <port-number>' \
  --progress --delete \
  user@remote-server:/path/to/remote/folder /path/to/local/folder
```

# Story Mapping

## Resources

https://www.easyagile.com/blog/the-ultimate-guide-to-user-story-maps/#what-is-user-story-mapping

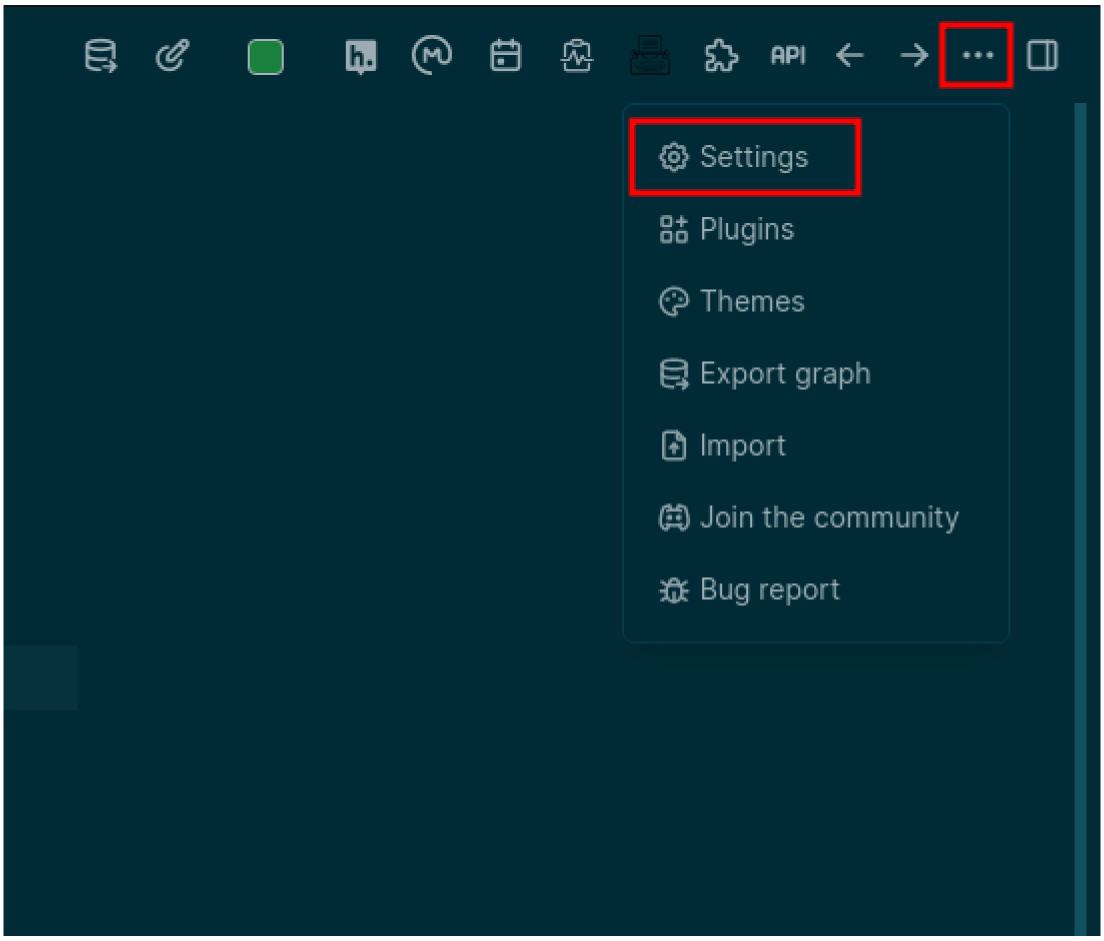
# Logseq HTTP API

LogSeq provides a HTTP API for developing plugins. The documentation is not particularly intuitive to get used to.

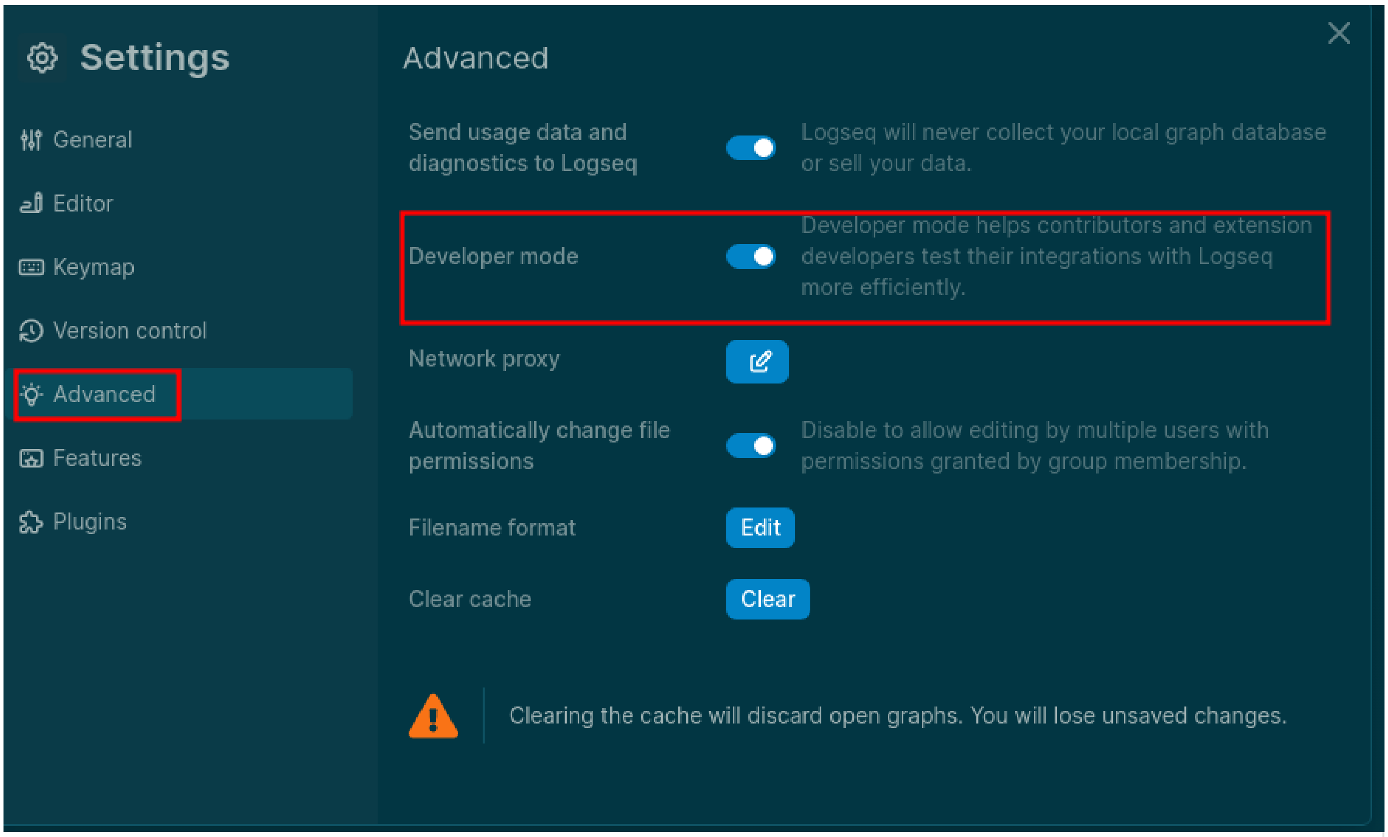
## Enabling the API

**1. Turn on Dev Mode + API**

You need to turn on developer mode within Logseq via the settings menu:



Go into the advance settings and enable developer mode



Settings

General
Editor
Keymap
Version control
Advanced
Features
Plugins

Advanced

Send usage data and diagnostics to Logseq

Logseq will never collect your local graph database or sell your data.

Developer mode

Developer mode helps contributors and extension developers test their integrations with Logseq more efficiently.

Network proxy

Automatically change file permissions

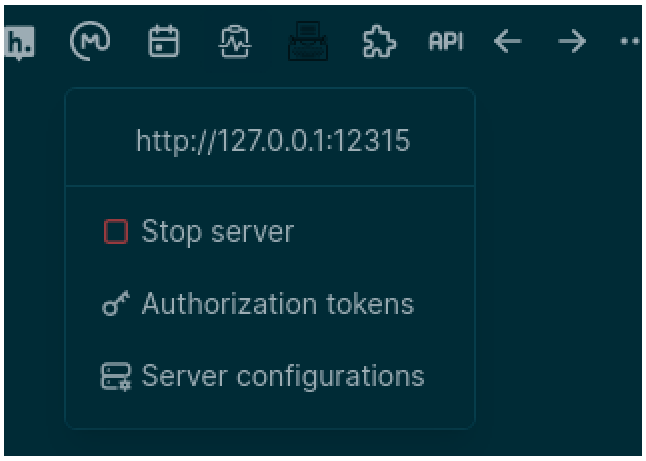
Disable to allow editing by multiple users with permissions granted by group membership.

Filename format
Edit

Clear cache
Clear

Clearing the cache will discard open graphs. You will lose unsaved changes.

Then when the app restarts you should be able to enable the API



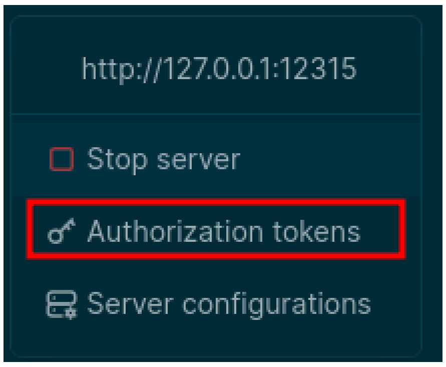
API

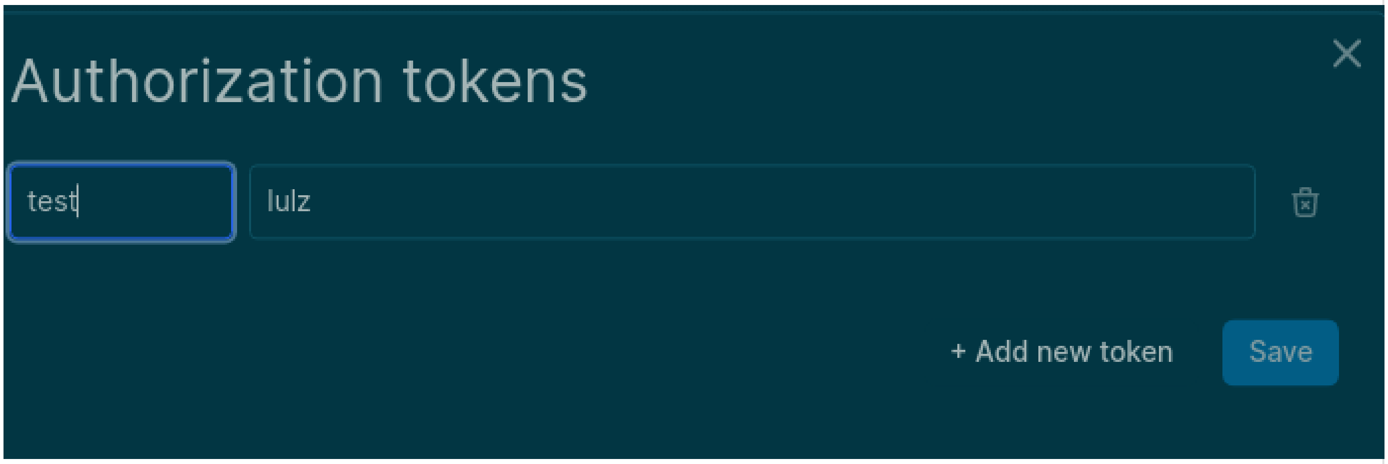
http://127.0.0.1:12315

Stop server
Authorization tokens
Server configurations

## 2. Add a Token

by default no token is provided so you won't be able to call the api. Open the manage tokens dialog and create a new token:





You will now be able to make HTTP requests to the given URL and PORT using `Authorization: Bearer lulz` or whatever value you chose.

## Using the API

If you open your browser and head to http://127.0.0.1:12315/ you will be advised that you can POST to http://127.0.0.1:12315/api with a JSON payload and you can specify which method to call and what arguments to pass. You can use the Logseq Plugin Docs to find a list of methods that can be used.

For example if I have a page called `Logseq` I could use the following payload along with an `Authorization: Bearer <token>` header to get the page's markdown block content:

```
{
  "method":"logseq.Editor.getPageBlocksTree",
  "args":[
    "Logseq"
  ]
}
```

You can pretty much use any of the methods listed on the plugin doc page with this method - you'll need to check the required arguments in the documentation and make sure that you pass the correct args.

# Data Lakehouse

A data lake house combines together the best bits of data warehouses and data lakes.

Data Lakehouses could be seen as the natural convergence of the two architectures (see https://cloud.google.com/blog/products/data-analytics/data-lake-and-data-warehouse-convergence)

## Data Lake

Data Lake is the name we give to a collection of tools that are often used together to process large amounts of data. Typically it includes a storage system like S3 or HDFS and a processing system like Apache Spark or Hadoop.

- Store lots of data - often in its raw "unprocessed" form in pseudo-real-time
- Process a subset of data in real-time or in batch modes
- Provide language-agnostic language runtimes for data analysis.

## Data Warehouse

A data warehouse is usually where data that has been processed and is now structured is stored. It is often used directly by business analysts in downstream applications. Data warehouses don't scale easily and typically have a lot more validation and processing associated with them.

## Data Lakehouse

A data lakehouse attempts to combine elements of both Data Lake and Data Warehouse - again it is typically the name given to a group of systems architected together to provide this set of functionality. It normally supports Extract, Load and Transform paradigm.

## References

- https://cloud.google.com/learn/what-is-a-data-lake
- https://www.snowflake.com/guides/what-data-lakehouse

- https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-data-lake
-

# Design Frameworks

Design frameworks provide out of the box styling and components for use in websites. Many frameworks sit on top of Javascript and Typescript libraries and some lightweight frameworks simply provide CSS styles on top of static HTML.

## React Frameworks

## Lightweight CSS Frameworks

- SimpleCSS - written by Kev Quirk and provides a very simple and lightweight framework on top of standard HTML5 elements and components.
- Foundation - another lightweight CSS framework that works without loads of javascript libraries.