SCAM Manual

jul

$Version : 202503$

Contents

WTFPL 2.0 Do any thing you want with this book except claim you wrote it.

Synopsis

To look smart you either say it in latin or write in LaTeX and add the Naviers Stocke equation

$$\frac{\partial \rho}{\partial t} + \vec{\nabla} \cdot \vec{j} = 0$$

to look smarter

But most of us ain't smart enough to use LaTeX, at most we can use markdown an easy to learn text renderer.

So here is my solution to be a professional scamer : this is a front end to a pandoc toolchain based on mind mapping for structuring the thoughts with a real time rendering of the markdown and quite a few tricks.
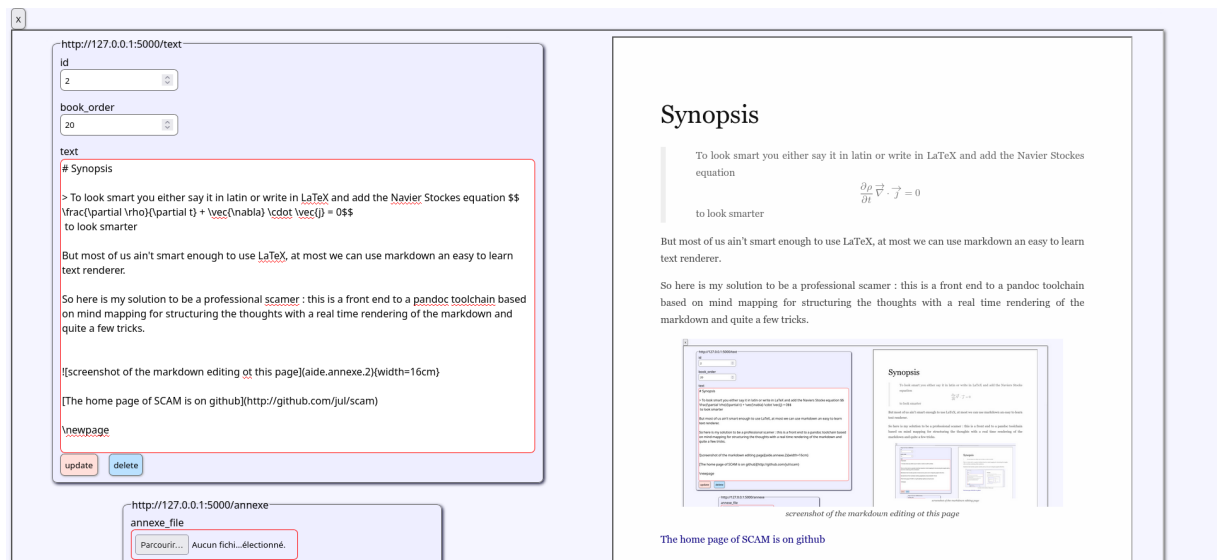

Figure 1: screenshot of the markdown editing ot this page

The home page of SCAM is on github

Installation

I am too lazy to write an install script, make a debian package, or a full pip requirements solution because I need binary installs.

As a result I resorted to the solution of the lazy man which source tells you all you need : making it a docker file.

```dockerfile
FROM debian
ARG DB
ENV DB $DB
ENV LANG C.UTF-8
RUN mkdir -p /scam/assets
RUN mkdir -p /usr/share/man/man1 && mkdir -p /usr/share/man/man7
RUN apt-get update && apt-get -y dist-upgrade \
    && rm -rf /var/lib/apt/lists/*
RUN apt-get update && apt-get -y --no-install-recommends install \
    python3 python3-pip python3-venv python3-setuptools \
    python3-sqlalchemy texlive pandoc graphviz virtualenv \
    python3-magic sqlite3 texlive-xetex texlive-latex-extra \
    texlive-fonts-recommended texlive-lang-french graphviz lmodern

RUN sed -i 's/^Components: main$/& contrib/' \
        /etc/apt/sources.list.d/debian.sources
RUN apt-get update
RUN apt-get install -y ttf-mscorefonts-installer fontconfig
RUN fc-cache -f -v

RUN useradd scam -d /scam --uid 1000 -m -s /bin/bash
RUN mkdir /venv
RUN virtualenv --system-site-packages /venv
COPY --chown=scam . /scam
WORKDIR /scam
RUN . /venv/bin/activate
ENV PYTHONPATH=/venv/bin
RUN /venv/bin/python -m pip install --no-cache-dir \
    --disable-pip-version-check -r /scam/requirements.full.txt
USER scam
EXPOSE 5000
CMD . /venv/bin/activate && cd /scam \
    && DB=${DB:-scam} /venv/bin/python /scam/scam.py
```

with the following requirements :

```
archery
pandoc-include
dateutils
multipart
filelock
Mako
pandocfilters
panflute
passlib
python-dateutil
SQLAlchemy>=2
SQLAlchemy-Utils
time-uuid
```

to use it I recommend the side car technique wich can be used this way so that
you can access the assets dir which contain the book :

```
docker build -t scam .
docker run -i -t -e db=bookname \
    --mount type=bind,src=.,dst=/scam \
    -p5000:5000  scam
firefox http://127.0.0.1:5000
```
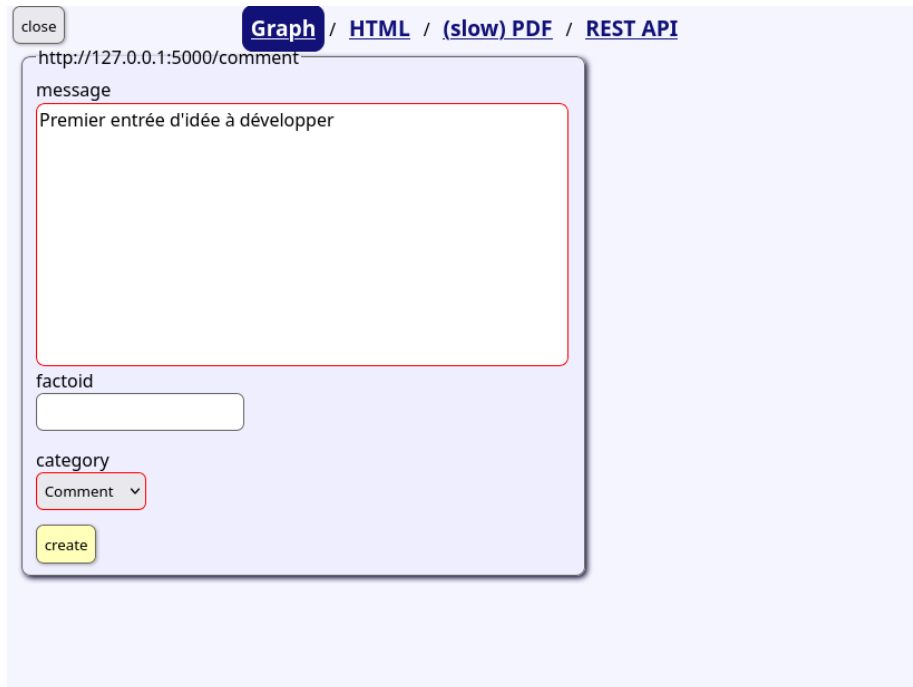
Walkthrough: writing the aide

Creating your first post

The landing page give you one option : POST :D

Figure 2: Graph interface : time to create your first post

On click you should see this and be able to fill the value:

Figure 3: First post

and then click *create*

Attaching a content to an entry.

I chose the policy that one micro item is related to one and only one attachment of the embedable kind you want.

The **annexe** widget below is used to load and delete annexes (attached files) that will be stored in the database.

These items will be available as pictures in the markdown editor by the given name on the top.

Figure 4: by clicking on the « attach » button you can attach a content to the post it entry

Accessing the text

To access the post entry you click on the node/post and it will open a modal window in which the **text** button is accessible for editing your book text
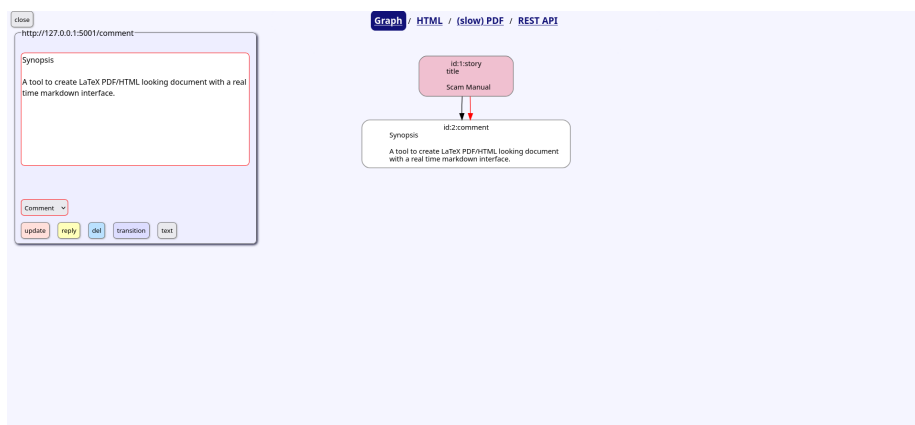

Figure 5: Once you click on a node in the graph the post editing interface appear

Developping your first comment and setting your book title

First comment is specific in the sense it is also used for the title. With pandoc you can add metada used for LaTeX.

The *markdown* extension used here is the pandoc one

Here is a typical Pandoc flavored Markdown entry to setup the LaTeX settings here with the french settings :

```
% TITLE
% AUTHOR
% DATE \
  \
  \ ![](aide.annexe.1){width=15cm}


---
mainfont: Georgia
header-includes:
 - \usepackage[french]{babel}
 - \usepackage{hyperref}
 - \definecolor{myblue}{rgb}{0.28, 0.24, 0.48}
 - \hypersetup{colorlinks=true, allcolors=myblue}
 - \let\tmp\oddsidemargin
 - \let\oddsidemargin\evensidemargin
 - \let\evensidemargin\tmp
  - \reversemarginpar
 ---
```

Your real time markdown input is definitely confused but that's fine. It is a quirk :D

Scam Manual

Jul

© 2025
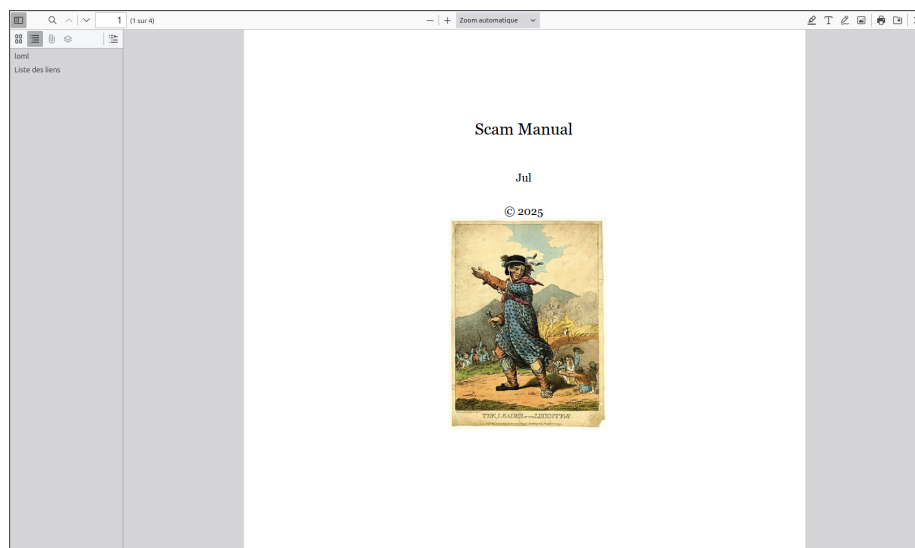
Figure 6: screenshot of the rendered PDF with the title

Visualizing your document

Now that you have rinced and repeated a few time the post entry/text process you may want to check the output.

There are 2 main output : HTML and PDF.

HTML output

To see the result

it's now time to visit the HTML rendering URL. You should have a side by side view of the generated standalone HTML and the generated PDF.
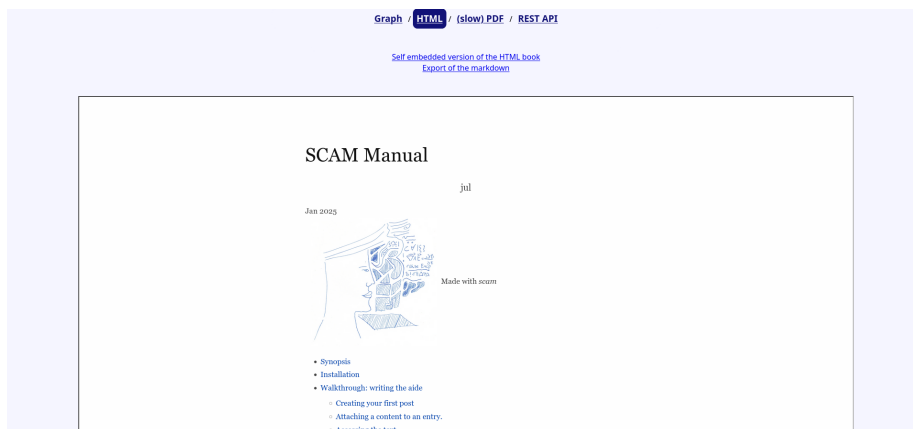
Figure 7: book rendering

You notice that there is a nice self embedded HTML link. This includes CSS and pictures inside the document for serving the document as a single file.

PDF

The PDF renderer accessible from the menu will give you the PDF rendering.
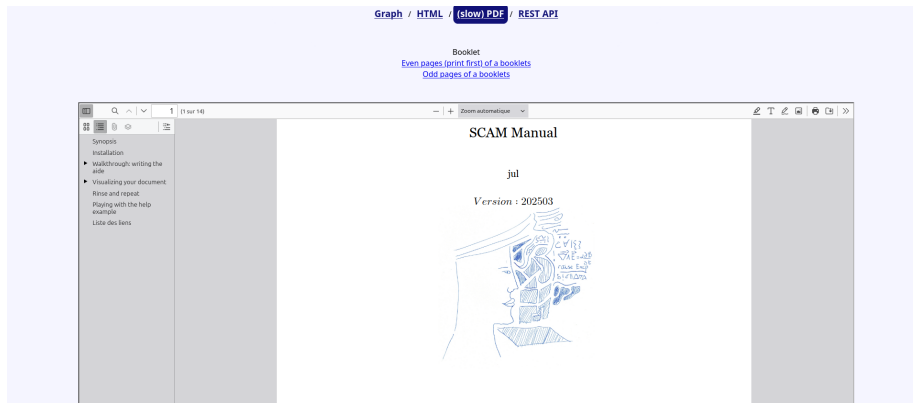
Figure 8: PDF view of the document

Rinse and repeat

Ather a few more entries that are boring because very repitive if you consult the graph URL you should have now more entries.
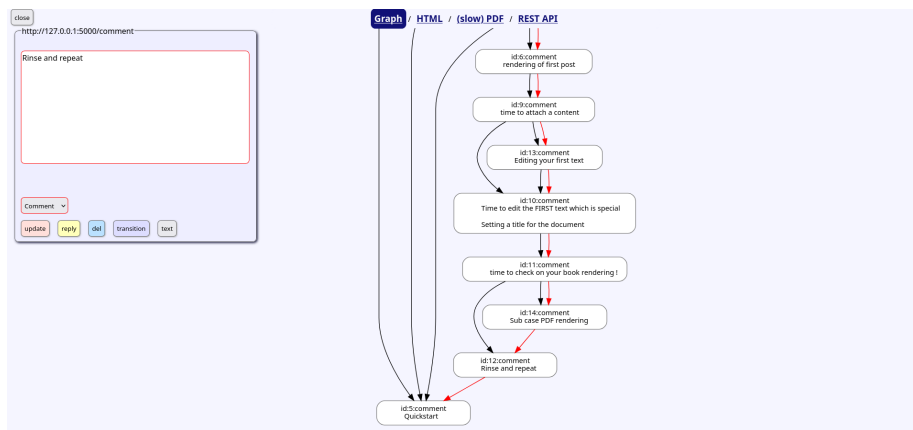
Figure 9: Your graph should now expand itself as the book

The « book order » is the red lines, they follow the ascending id order but can be overriden with the book_order rank available in the **text** view.

Playing with the help example

This book is available in the repository as a sqlite database[1].

To try it :

```
docker run -i -t -e DB=aide --mount type=bind,src=.,dst=/scam \
    -p5000:5000 --user  1000:1000  scam
firefox http://127.0.0.1:5000
```

---

[1]I am beginning to find this feature useless. Less is more, hence I'm thinking of removing it.

Dev corner

« Design »

Well, there was no design.

All I know is that web services came first and that all User Interface is based on calling them in ajax.

Web services are presented as a serie of form, one for each table, and the actions required to be filled in the request are the *input type=submit* button.

You can check how it is done in the lite suite I use to test part of the design.

Figure 10: Exemple of a REST access to the table comment

Synchronous

Since all my *events* meaningful (saving) are plugged on *onclick* events in javascript, it is a synchronous application ensuring that when my wife and kid interrupt me, the document is always in a sane state. So far, I love it so much because it actually acted as a safeguard in the here fore mentioned situations in real life during the writing of this manual, that I am reluctant to put an « autosave » feature.

But, nice to have would be a « CTRL+S » binding that does the save with a nice modal message. [2]

CAVEAT : hit update or CTRL+S often or you shall lose work.

Model

A careful examination of the entity relationship diagram will point at **dead** data such as the user table, or the factoid column that is not used anymore.
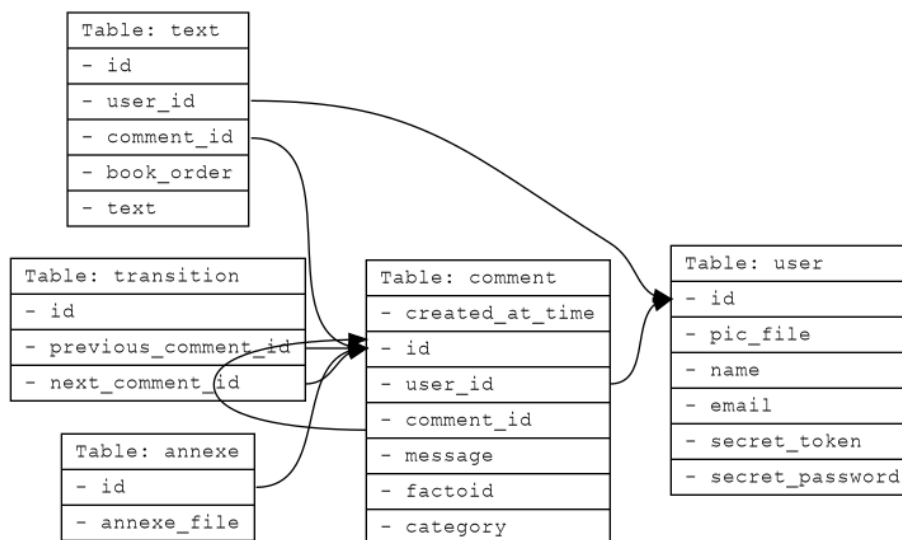

Figure 11: Entity Relationship Diagram

As you can notice the **comment** table is the center of the model.

---

[2]The save by hitting CTRL +S is actually in the base code but I don't know how to show it to the user. Lol.

HTML As A Model (HaaM)

Well, I made sure the web services serving the REST API would be in sync, by .... creating the model from the HTML forms served there.

The rule being one form, one table. One input, one column. And by adding some attributes to enhance the HTML I have an HTML dialect called **HaaML** as *HTML as a model Markup Language* to add sql related attributes and map them into the right SGBDR type.

Making me thus the *"piped piper of HaaML-in"* from which your mother guarded you from.

The only documentation of this language is of course the source code of scam.py itself.

Hence, to change the model, you have to change the HTML served to describe the model.

I still am not sure whether it's genius or stupid. But facts are : the web service always reflect with a 100% certainty the database model served by the application. Which as far as I am concerned is the right way to do it.

Creativity boosters (limitations)

Pretty much these limitations are due to some of my lack of skills.

You have the following important limitations :

- there is a one to one association between *annexe* (joint picture) and a *comment*[3] ;

- there is a one to one association between *annexe* (joint picture) and a (markdown) *text* and the joint is on .... *comment_id* or *id*[4] ;

- the table **transition** adds an edge between comments on the graph[5]

- the *factoid* column of the **comment** table is not used anymore [6];
- the **user** table is not used anymore [7];

---

[3]because I did not wanted to code a file explorer of the potential attachment (I hate front end development as much as back end one);

[4]I did messed up thing if I remember well, and I'm pretty sure that instead of a joint on *comment_id* I do it in one tiny **vicious** place of the interface on *id.*

[5]I am beginning to find this feature useless. Less is more, hence I'm thinking of removing it.

[6]I changed the interface by removing features, but I had the lazyness to write the migration script and change the custom SQL I wrote. Planning for migration scripts and version handling is quite a lot of work you know, that I will not priorize because, I have the skills, but not the will.

[7]same thing as above

Serendipity

Also called « un planned features ».

Vanilla Markdown export with assets (gruik inside)

You noticed there on the page for the html export that there is a link for the full markdown export.

Figure 12: yep on this page

Actually, it is a modified markdown with with some custom pandoc processing with pandoc lua filters applied and a tad of pandoc magic.

So you will need this file to make your own builder, and also the images that are automatically generated upon calling the HTML view[8]. The picture needed to complete the markdown export will be located in the assets directory with the name `assets/$DB.annexe.*` where DB is the name of the DB you use (default is **scam**).

---

[8]I think I forgot to check if images were present before generating the PDF assuming people would check the HTML first. Stupid me.

Self embedded HTML

I am testing single HTML self containing page with the pictures embedded inside.

Test it for me, and if it works it will become the default and only HTML view.

Psycodelic experience

Most the making of this software *-due to an intense real life pain-* have been made under drugs such as ketoprofen, opium and more.

Pain and drugs totally change the way you code, if you want to go on foreward, you have to sharpen your mind through the pain to stay focused, ensuring a maximal brutality to hack your way.

As the maintainer of the source code I have been pleasantly surprised by how easy it was for me to understand what I did, even though it totally goes against the currently admitted best practices of full stack development (such as using fucking frameworks for every layers of the stack).

It is like discovering your true self in coding because you had to code so diminished intellectually that it prevented your past self to be too smart for your present self with more brain.

Since I don't like pain, it is an experience I don't recommend.

Liste des liens

http://127.0.0.1:5000/ The landing page

https://pandoc.org/MANUAL.html#pandocs-markdown markdown extension used here is the pandoc one

http://127.0.0.1:5000/book it's now time to visit the HTML rendering URL

http://127.0.0.1:500/pdf PDF renderer

http://127.0.0.1:5000/svg you consult the graph URL

http://127.0.0.1:5000/model Web services

https://github.com/jul/scam/blob/main/test/load.py the lite suite I use to test part of the design

https://github.com/jul/scam/commit/1957c08d958d8a8b8e67324d19e6602fa7a1612c The save by hitting CTRL +S is actually in the base code

http://127.0.0.1:5000/model web services serving the REST API

https://github.com/jul/scam/blob/main/scam.py#L63 scam.py

http://127.0.0.1:5000/book page for the html export

https://github.com/jul/scam/blob/main/mkdoc.sh custom pandoc processing

http://github.com/jul/scam The home page of SCAM is on github