Blogging platform – What Would TEX Do?

Description	PHP, LATEXand Git based blogging engine.
Period	Summer 2013
Languages & Libs	PHP, LATEX, Mustache
Tags	Blog, Git

I wrote this blog engine to enable the creation of new articles in LATEX format and effortlessly publish them in Web. As a by-product it enables trivial PDF generation of each article, or even a combined PDF with all the articles concatenated with a interactive table of contents, automatically numerated figures with references from the text, and many other features that LATEX users take for granted.

First I looked at standard blogging solutions, but I quickly realized that my special requirements could difficult to implement as plugins or otherwise. At least it would take the same amount of time, and even then be more troublesome to maintain, deploy and backup. I really enjoy using LATEX for writing documents, so I figured that why not to use it also for blogging.

This platform uses Git for SCM and CMS, LATEX for writing, PHP to generate navigation links etc. and mustache templating to generate the final HTML. Most of the server side functionalities can be avoided by utilizing 3rd party services such as Google Analytics and Disqus. To my surprise it took only about 1000 lines of PHP to get the basic stuff working such as tag-based browsing of articles and references to automatically numbered figures. The current architecture can be seen in Figure 1. All three subsystems (LATEX, PHP and Mustache) are stored in a Git repository, and the logic to bring everything together is implemented in PHP. Actually this resembles a lot of the MVC pattern. The article (model) is stored in .tex files, for which two different views (PDF and HTML) can be produced. In this case the user cannot alter any of the views, so there is no need to deploy the actual model or controller components. Instead only static HTML, PDF etc. files need to be copied to the web server (or Dropbox's public folder).

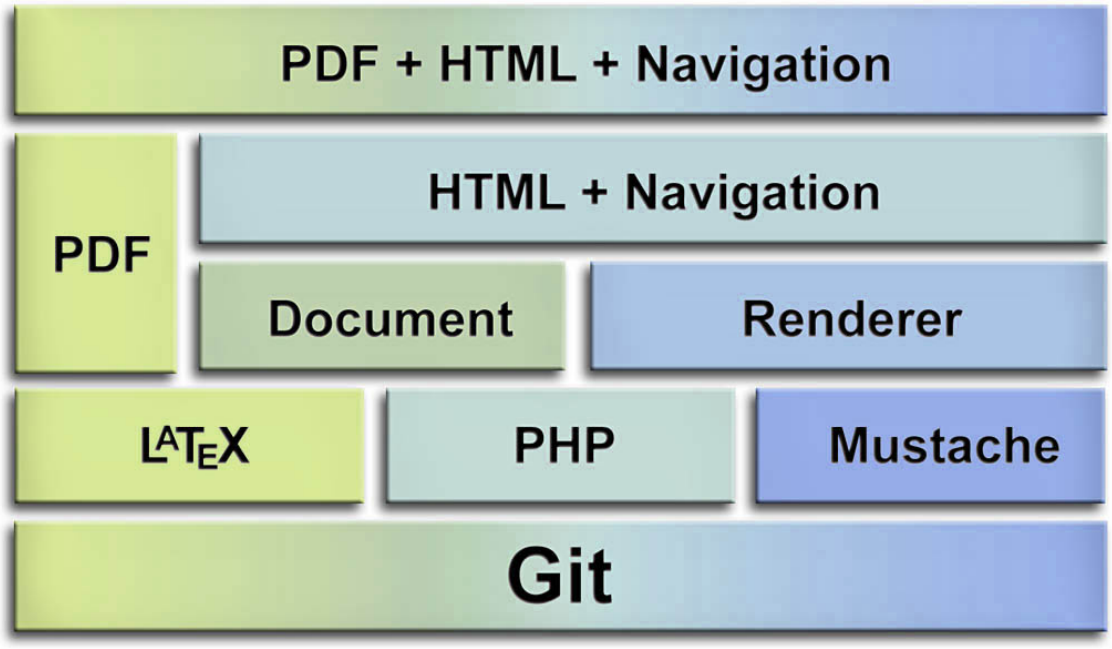

Figure 1: Current blog project's architecture. Everything is based on Git, which stores and supports the three sub-systems. This resembles the MVC model, where model has none of the logic.

PHP reads the .tex file to produce the blog posts' body and meta data, which is passed to a Mustache template which produces HTML and CSS. PHP also has the logic to generate navigation links, find out most popular tags and languages, and to execute other useful tasks.

Currently the platform supports following features (in PDF and/or HTML):

- Automatically numbered figures with named referencing from the main text
- Writing articles using any text editor, but IDEs such as TeXworks have their benefits
- Article browsing by project language or tag (in HTML)
- Links to external sites (with an indicating icon in HTML)
- Sophisticated symbol annotation, such as $\alpha = 360°$ and LaTeX
- Sorting the articles by last modification date (using Git blame to ignore minor changes)
- Identifying the modification date of each line in the article (using Git blame)
- PDF download of any article, or all articles in a chronological order
- Easy deployment (just a single folder to be copied, which contains all static generated content)
- BibTeX style article reference management
- Image magnification on click (easy with jQuery)
- Articles' language and tag co-occurrence matrix
- Automatic *hyphenation* in PHP to avoid sparse line when using justified text in HTML

At the file system & Git level, each article is stored under its own folder. This also ensures that each article has an unique identifier, which is used as the file name of the HTML files, and also as a prefix for all photos in articles. This avoids any name collisions, and lets us put all static content under the same folder for publication. The output generation script is also responsible for calling `pdfLaTex` for each article, and to copy PDFs along with article's images to the output folder.

A snapshot of my Git commit messages and timestamps can be seen in Figure 2. Basic features such as tag based browsing and mustache based templating were quite fast to implement. My normal development hours seem to be 2pm to 2am.

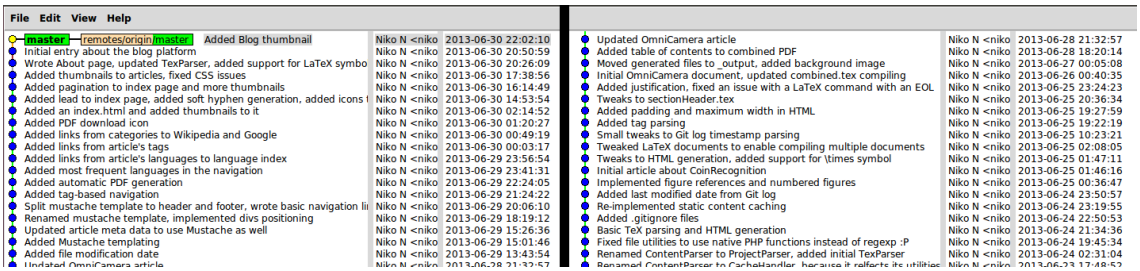

Figure 2: A snapshot of the development Git log.

These are some of the next features to be implemented (stroked out are done):

- Improve LaTeX tokenization, current regular expressions based implementation is limited
- Have a more scalable method of handling images, currently they are part of the Git repository
- Support for equations (possibly in MathML)
- ~~Better graphical design with better support for IE9~~
- ~~Support for embedded source code (at least LaTeX, Matlab and PHP) with syntax highlight~~
- ~~Support for data in a table format~~
- ~~Free text search engine (it is possible to use PHP to generate a static JavaScript file for this)~~
- ~~Ability to comment on posts (using Disqus or something similar)~~

Writing a new article is a fairly trivial process. There are four .tex files which are shared among all articles:

- commands.tex: Defines commands for describing article's metadata, and for inserting the header, footer, figures, references etc.
- header.tex: Has all the `\usepackage` commands, specifies page margins and starts the document.
- sectionHeader.tex: After the article's metadata has been specified, this inserts the project's title, thumbnail and displays the metadata.
- footer.tex: This only has `\end{document}`

The beginning of this blog article looks like this:

```
\input{../_latexCommon/commands.tex}
\myInput{../_latexCommon/header.tex}

\renewcommand{\basepath}{BlogPlatform}
\renewcommand{\projectTitle}{Blogging platform \--- What Would \TeX{} Do?}
\renewcommand{\projectStart}{Summer 2013}
\renewcommand{\projectEnd}{Summer 2013}
\renewcommand{\projectLanguages}{PHP, \LaTeX, Mustache}
\renewcommand{\projectDescription}{PHP, \LaTeX and Git based blogging engine.}
\renewcommand{\projectTags}{\tagBlog, \tagGit}
\renewcommand{\projectChallenges}{To develop and maintain a fully functional blog.}

\input{../_latexCommon/sectionHeader.tex}
```

In the blogging platform all "non-articles" are in folders which are prefixed with an underscore, so that PHP can easily detect and ignore them during HTML generation. The rest of the content can be normally written, and the file ends with an `\myFooter`.

Figures are inserted by using a custom command: `\insertFigure{git_log.png}{A snapshot of the development Git log.}{git_log}`. Parameters are filename, caption and figure name. This has two benefits: standardized image settings across all articles, and easier TEX parsing in PHP. Figures can be normally referenced from text by typing `\ref{fig:git_log}`. There is a similar custom command for listing references.

To enable multi-article PDF generation, the header isn't included by using the standard `\include`, but instead use `\myInclude`. In commands.tex this is defined to use the normal `\include`, but when the multi-article PDF is generated this commands is re-defined not to do anything. Similarly the footer would not actually end the article, but instead just insert a `\clearpage` command to make the next article start from a clean page.

With all these commands and practices in place, generating the multi-article PDF is very easy. The combined.tex has identical first two lines as normal articles, which inputs the commands.tex and header.tex files. Then there are two tweaks, which disables the `\myInput` and renews `\myFooter` to insert a `\clearpage`. Then the cover page and table of contents are generated (I'm sure the line positioning could be done in a prettier way).

After the table of contents, an additional _combined.tex is inserted. This is auto-generated by PHP, which just lists all existing articles. Before inputting each article the `\projectModified` is defined, which is the file's modification date which is read from Git's commit log. Articles are automatically sorted from newest to oldest. When a single article is generated, this date is not available. It would be possible for PHP to write this timestamp into a file inside each article's folder.

```
\input{../_latexCommon/commands.tex}

\renewcommand{\setFigureNumbering}{
        \usepackage{chngcntr}
        \counterwithin{figure}{section}
        \counterwithin{table}{section}
}

\renewcommand{\Section}[1]{
    \section{#1}
}

\input{../_latexCommon/header.tex}

\renewcommand{\myInput}[1]{}
\renewcommand{\myFooter}{\clearpage}

\ \vspace{2cm}

\begin{center}
        \line(1,0){200} \\ \ \\
        \LARGE Niko Nyrhilä\\
        \Large Combined project portfolio
        \ \\ \ \\
        Generated \today \\
        \line(1,0){200} \\
\end{center}

\ \vspace{1cm}
\tableofcontents
\clearpage

\input{_combined.tex}

\input{../_latexCommon/footer.tex}
```

Currently the five first items of _combined.tex looks like this:

```
\renewcommand{\projectModified}{\textbf{Modified} & 13th June 2021 \\}
\input{../Agadmator/agadmator.tex}

\renewcommand{\projectModified}{\textbf{Modified} & 10th May 2021 \\}
\input{../Satellites/satellites.tex}

\renewcommand{\projectModified}{\textbf{Modified} & 2nd April 2018 \\}
\input{../JGitBlame/jgitblame.tex}

\renewcommand{\projectModified}{\textbf{Modified} & 7th May 2017 \\}
\input{../BenchmarkTaxiridesEsSql/benchmarktaxiridesessql.tex}

\renewcommand{\projectModified}{\textbf{Modified} & 19th March 2017 \\}
\input{../CljTaxirides/cljtaxirides.tex}
```