Dokumentation VLP-Statisch

Angelique Gößl

Einleitung

- Ziel des Projekts

- Es gibt wissenschaftliche Webprojekte die verwaisen, da die Projektteilnehmer das Institut verlassen haben etc.
- Die Projekte stehen aber weiterhin im Netz und müssen am Leben erhalten werden.
- Die Software der Projekte veraltet und die Sicherheit der Server ist dadurch gefährdet.
- Frage: Wie können wir solche Projekte zukunftsfähig machen und erhalten.
- Ansatz: Statische Seiten und Template Engines

VLP

- Das VLP wurde als Beispiel für die Statifizierung ausgewählt, weil
  - das Projekt verwaist ist
  - es aber immer noch viel augerufen wird
  - die darunterliegende Serversoftware (Zope) veraltet ist und nicht mehr gewartet wird
- Das Projekt soll einen *proof of concept* liefern, ob und wie solch eine Website statifiziert werden kann.

Schritte

1. Auswahl eines zukuntfsfähigen Layouts (Responsives Design/Rechner,Tablet,Handy)
2. Auswahl einer hinreichend schnellen Template-Engine (Hugo, Perl Template Toolkit (TT2), RubyFrontier)
3. Daten sollen in reinen YAML, JSON oder TOML Dateien vorliegen (Trennung von Daten und Layout)
4. Implentierung der Template-Engine und Verknüpfung mit den Daten
5. Testlauf
6. Dokumentation
    1. Verwendete Werkzeuge
    2. Editor TextMate 2
    3. Pandoc (Markdown)
    4. LaTex (für die Dokumentation), Skim (als PDF Viewer)
    5. Hugo
    6. Perl Template Toolkit (TT2)
    7. RubyFrontier
    8. MAMP installiert
    9. Techniken
    10. HTML und CSS
    11. TOML
    12. Versionskontrolle mit Github und Git

Installation Perl Template Toolkit

Das Perl Template Toolkit (TT2) ist eines der ältesten *Template Engines* und da es seine interne Sprache nach Perl kompiliert relativ schnell. Aufgrund des Alters und seiner langjährigen Nutzung ist es zum einen *feature complete* und zum anderen nahezu fehlerfrei.

Wie fast alles zu Perl liegt auch TT2 auf CPAN zu finden und kann daher

```
Sudo cpan Template
```

installiert werden. Nach der erfolgten Installation mussten die einzelnen Dateien und Verzeichnisse angelegt werden. Dafür wurde folgende Ordnerstruktur benötigt:

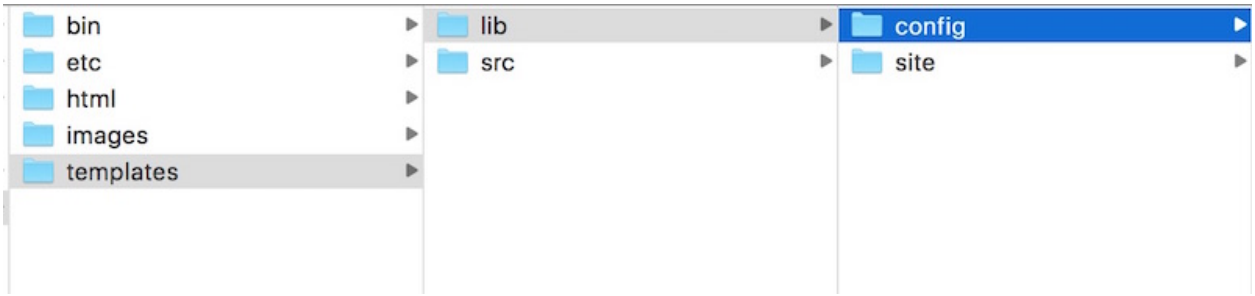

Abbildung 1: Ordnerstruktur TT2

Das Projektverzeichnis ist `vlptt2` und darin wurden die Verzeichnisse `bin`, `etc`, `html`, `images`, `src` und `templates` angelegt. Im Ordner `templates` wurden noch die Unterverzeichnisse `src` und `lib` benötigt. Die Namen der Verzeichnisse sind Konvention.

Im Ordner `etc` liegen Konfigurationsdateien. Für das Projekt wird nur eine Konfigurationsdatei benötigt, die ich `ttree.cfg` genannt habe. Diese Datei enthält folgende Daten:

```
# directories
src = ~/Documents/git/vlp/vlpstatisch2/vlptt2/templates/src
lib = ~/Documents/git/vlp/vlpstatisch2/vlptt2/templates/lib
dest = ~/Documents/git/vlp/vlpstatisch2/vlptt2/html

# copy images and other binary files
copy = \.(png|gif|jpg|jpeg|pdf|svg|mp4)$

# misc options
verbose
recurse
recursion

# TT2 options
pre_process = config/main
wrapper = site/wrapper

# define some location variables
define rootdir = ~/Documents/git/vlp/vlpstatisch2/vlptt2
define rooturl = /vlptt2
define debug = 0
```

Als erstes wird in diese Konfigurationsdatei TT2 mitgeteilt wo die benötigten Verzeichnisse zu finden sind. In `templates/src` liegen die Quelldaten und in `templates/lib` wiederverwertbare Teilmodule. `html` ist das Zielverzeichnis, wo TT2 die herausgeschriebenen HTML-Dateien hineinschreiben soll.

Der nächste Absatz ist ein regulärer Ausdruck (Regex) der TT2 mitteilt welche Dateiformate er nur in das Zielverzeichnis kopieren soll, ohne die Datei zu bearbeiten. Die Optionen `verbose` teilt TT2 mit, dass es alle Fehlermeldungen und Warnungen ausgeben soll. Die Option `recurse` teilt TT2 mit, dass es auch alle Unterverzeichnisse durchssuchen und abarbeiten soll. Die Option `recursion` ist wegen der Namensgleichheit mit der vorherigen Version etwas verwirrend, aber sie sagt dem Toolkit nur, dass auch alle Templates rekursiv aufgerufen werden können. Denn auch Templates können Untertemplates enthalten.

Um die Installation zu testen, habe ich einige Testdateien in `templates/src` angelegt. Als erstes eine `index.html` mit diesem Inhalt angelegt:

```
[% META title = "Template Toolkit Test" %]

<h1>[% template.title %]</h1>

<p>
Das ist die erste Testseite: Test! Test!</p>
<p>
Die Nachricht ist: »[% message %]«</p>
```

Und in der `templates/lib/config` habe ich eine Datei `main` angelegt, die den Inhalt der Testseite enthält. Sie ist erst einmal nur sehr kurz.

```
[% message = "Hallo TT2 Welt!" -%]
```

In das Verzeichnis `template/lib/site` wurde die Datei `wrapper` erstellt, die ein einfaches HTML5-Grundgerüst für den Test enthält.

```
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="utf8" />
    <meta name="generator" content="TT2" />
    <title>[% template.title %]</title>

</head>

<body>
    [% content %]

</body>
</html>
```

An diesen drei Dateien kann man die Arbeitsweise des Perl Template Toolkits erkennen. Ersetzungen, im TT2-Jargon *Direktiven* genannt, werden mit `[% ... %]` geklammert. Diese Direktiven ersetzt TT2 durch die entsprechenden Werte der Quell- und Konfig-Dateien.

Projektmanagent