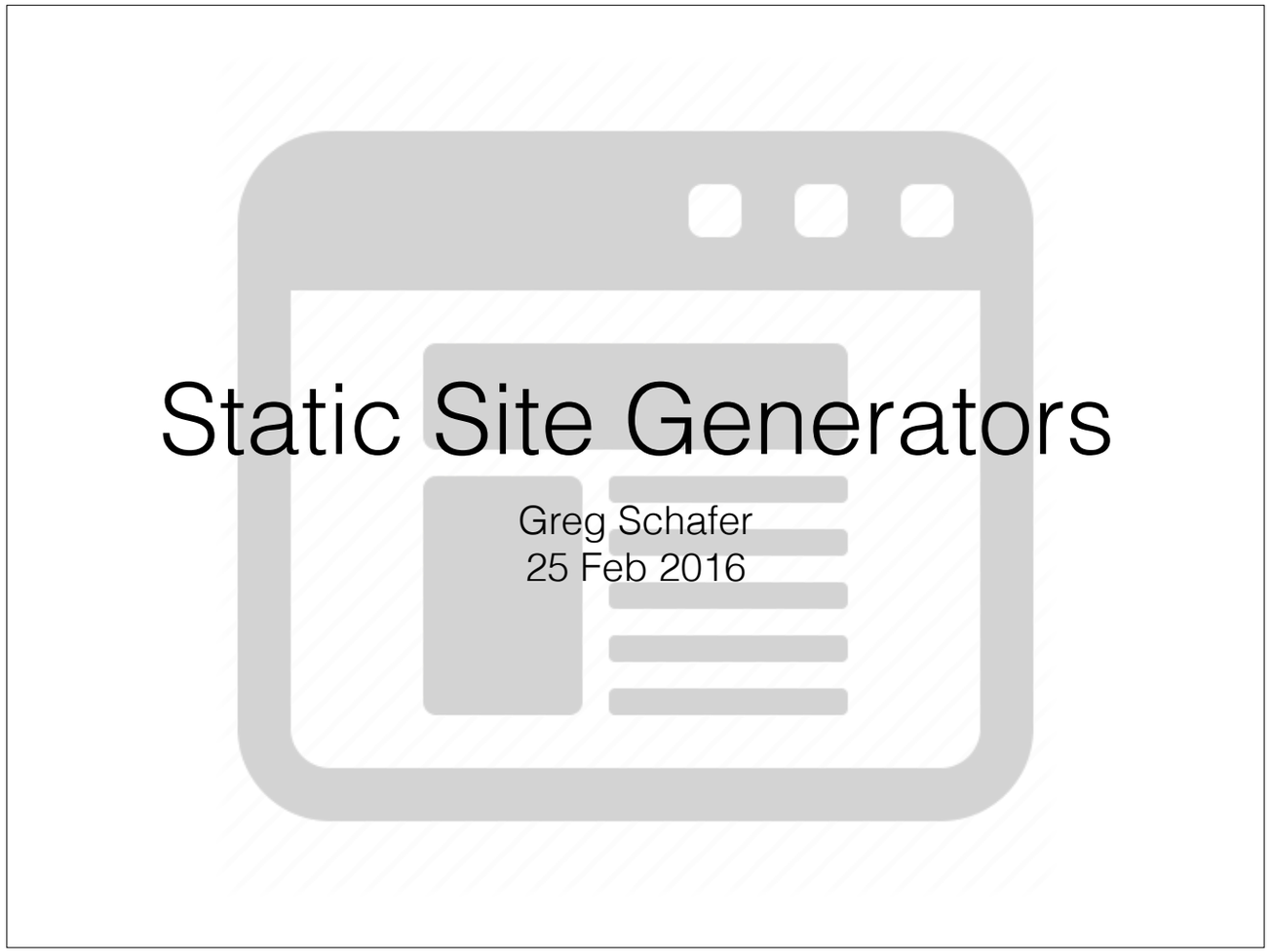


Static Site Generators

Greg Schafer
25 Feb 2016

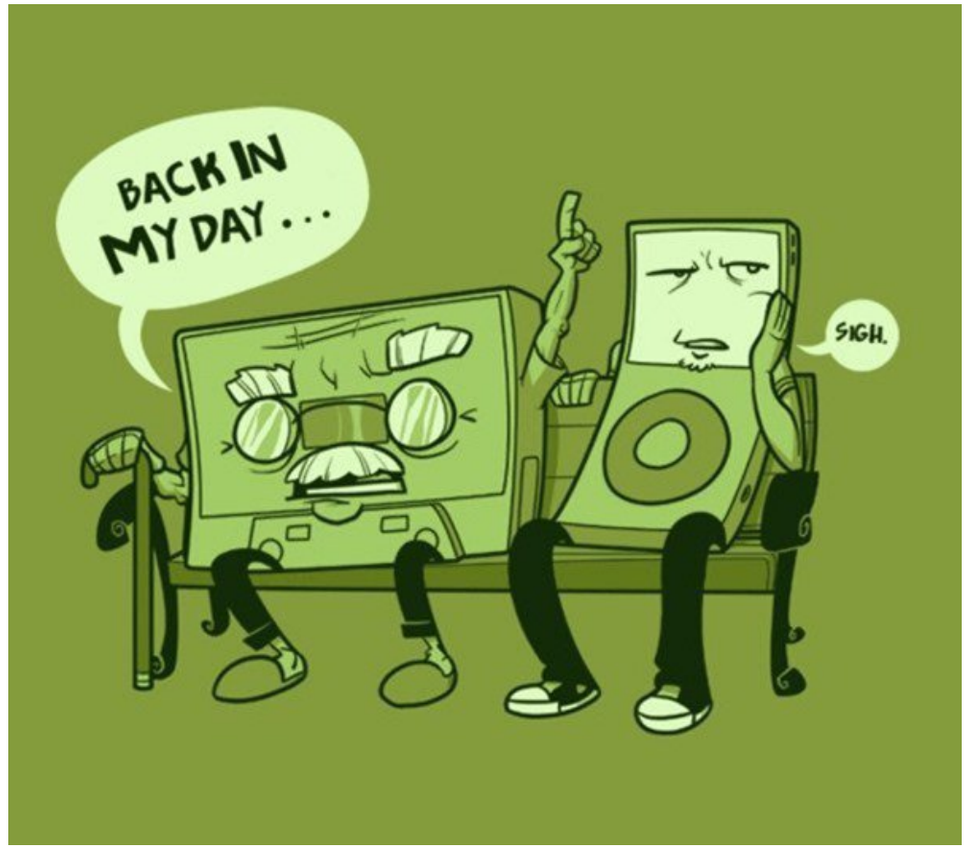
https://www.iconfinder.com/icons/374477/landing_page_layout_seo_web_design_web_page_webpage_website_icon

Outline

- History
- Purpose
- Pros/Cons
- Examples
- Let’s try out Lektor!

Outline

- History
- Purpose
- Pros/Cons
- Examples
- Let’s try out Lektor!



http://www.blamnews.com/90s-kids/

http://readwrite.com/2013/04/02/a-visual-history-of-the-web

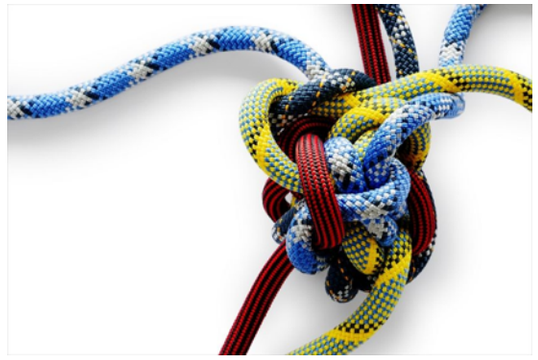
- all sites were static originally
- eventually complex nav and graphics
- then SQL databases (mysql 1995) rose and became the home of content
- PHP (1995) and soon followed by wordpress (2003), drupal (2001), joomla (2005)
    - over 25% of sites are wordpress
        - http://w3techs.com/technologies/details/cm-wordpress/all/all
    - over 80% of sites are php
        - http://w3techs.com/technologies/overview/programming_language/all

Challenges of dynamic

- Complexity
- Security (i.e. WordPress)
- Speed
  - Caching
- Traffic spikes

Challenges of dynamic

- Complexity
- Security (i.e. WordPress)
- Speed
  - Caching
- Traffic spikes



Challenges of dynamic

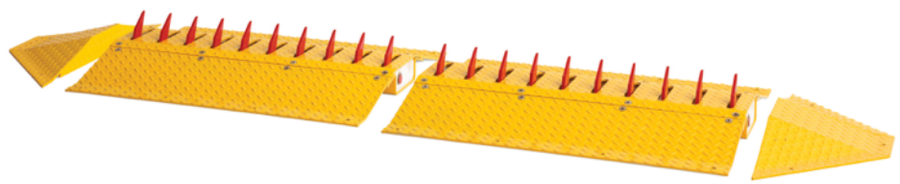
- Complexity
- Security (i.e. WordPress)
- Speed
  - Caching
- Traffic spikes

Challenges of dynamic

- Complexity
- Security (i.e. WordPress)
- Speed
  - Caching
- Traffic spikes

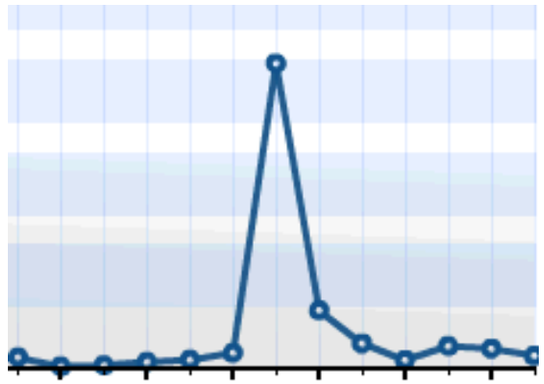
Challenges of dynamic

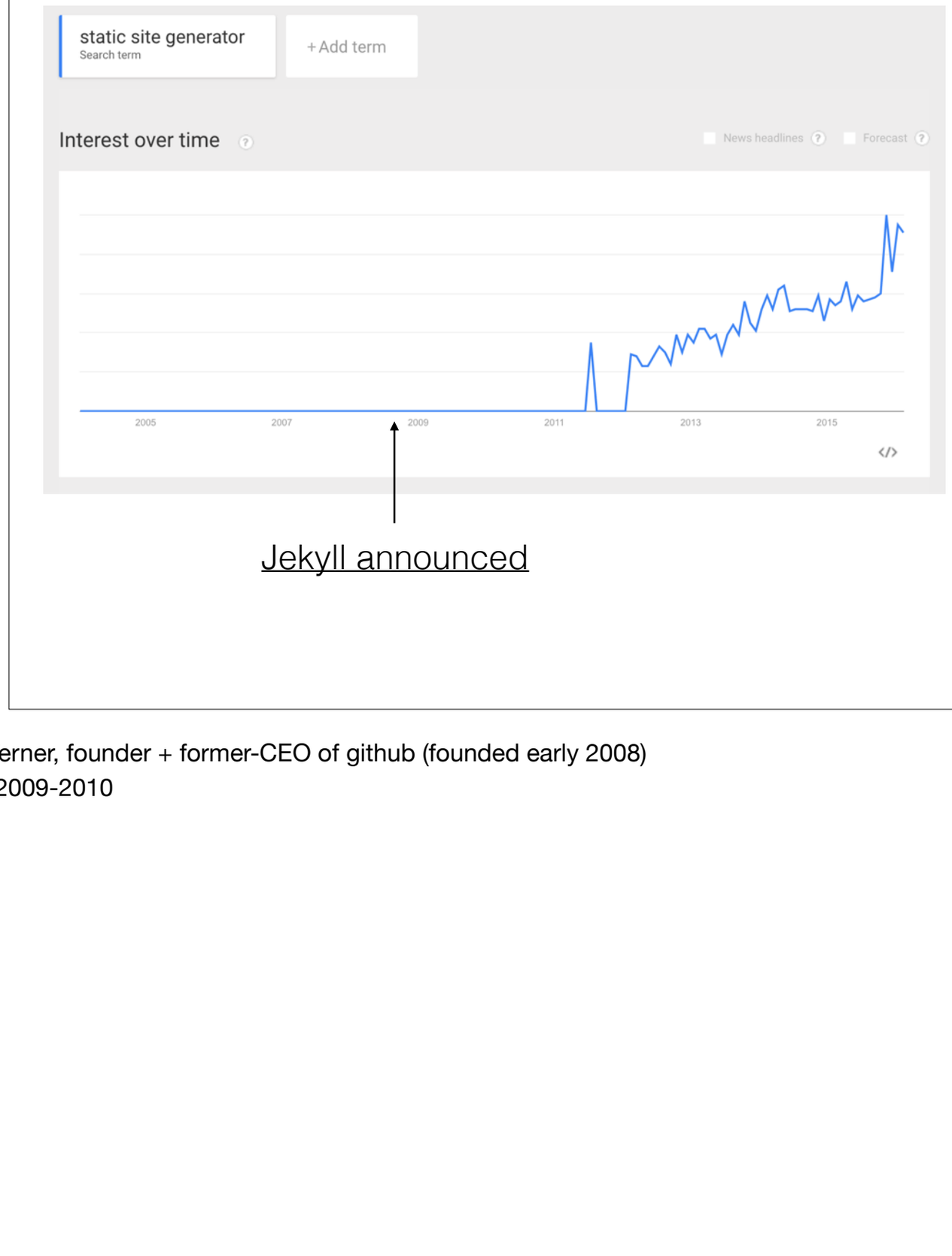
- Complexity
- Security (i.e. WordPress)
- Speed
  - Caching
- Traffic spikes



Challenges of dynamic

- Complexity
- Security (i.e. WordPress)
- Speed
  - Caching
- Traffic spikes





- jekyll announced by Tom Preston-Werner, founder + former-CEO of github (founded early 2008)
- dunno what happened during gap of 2009-2010

Static Site Generators

- Produce HTML pages
- Features
  - Templating
  - Markdown support
  - Metadata or content model
  - Asset pipeline

- metadata = categories, author, date, etc.
  - can be used for sorting, etc.
- asset pipeline = css compiling/transpiling, minification, bundling

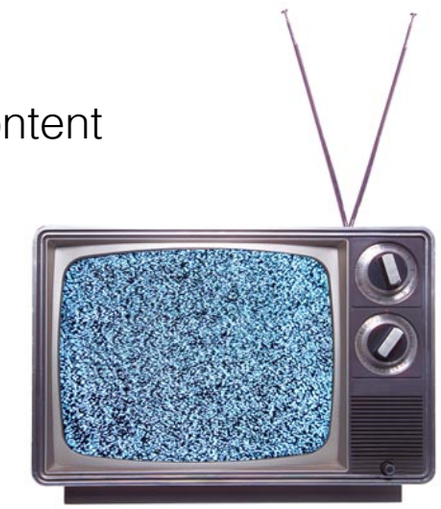
Advantages

- Speed
- Version control for content
- Security
- Simpler deployment, maintenance
- Scalable (traffic spikes)

- speed is a boon in mobile-focused era
- static lends itself well to CDNs
- mature build tools
- security concern reduced in scope to just web server (instead of app server, frameworks, databases, many servers, etc)
- sources:
    - https://www.smashingmagazine.com/2015/11/modern-static-website-generators-next-big-thing/
    - https://davidwalsh.name/introduction-static-site-generators

Disadvantages

- They’re uhh… static
  - No real-time or customized content
  - No user input
- Authoring tools lacking



- can add interactivity via javascript (post user input to API server or SaaS), embed disqus, etc.
  - risk/downside is SEO/searchability
- authoring tools lacking compared to CMS
  - not reasonable to expect authors to learn git and use a text editor to produce content
  - some generators rely on gists, google sheets, email, tacked-on CMS, etc for non-developer authoring

Generator	Released	Language	Template	Notable for	Used by
Jekyll	2008	Ruby	Liquid	Maturity, support	GitHub Pages, healthcare.gov, tons more
Middleman	2008	Ruby	Ruby ERB templates	Broader capabilities	Mailchimp, Nest, Simple
Roots	2012	Javascript	Jade	Extensible, mature asset pipeline	Carrot, Vice Media group
Hugo	2013	Go	Amber, Ace	Fast, but a bit barebones	UCSB, Gophercon
Pelican	2010	Python	Jinja2	Most popular python generator	Linux Kernel Archives
Lektor	2015	Python	Jinja2	Includes admin	Not much yet

https://staticsitegenerators.net/

Lektor

- Armin Ronacher
  - Flask, Jinja2, Werkzeug, other Pocoo
- Python
- Released in Dec 2015
- Motivation: “run a simple website in a secure manner without having to resort to all kinds of user-unfriendly hackery"

- armin ronaha
  - might know him from jinja2, flask, werkzeug, other pocoo stuff
- main complaint is existing site generators are too hacker-oriented, complex
- https://www.getlektor.com/blog/2015/12/hello-lektor/
- plays dota2: http://www.dotabuff.com/players/37429863

```
lektor-sites$ lektor quickstart
Lektor Quickstart
=================

This wizard will generate a new basic project with some sensible defaults for
getting started quickly.  We jsut need to go through a few questions so that
the project is set up correctly for you.

Step 1:
| A project needs a name.  The name is primarily used for the admin UI and
| some other places to refer to your project to not get confused if multiple
| projects exist.  You can change this at any later point.
> Project Name: pythonweb-demo

Step 2:
| This is the path where the project will be located.  You can move a
| project around later if you do not like the path.  If you provide a
| relative path it will be relative to the working directory.
> Project Path [/Users/occipital/repos/lektor-sites/pythonweb-demo]:

Step 3:
| Do you want to generate a basic blog module?  If you enable this the
| models for a very basic blog will be generated.
> Add Basic Blog [Y/n]: Y

Step 4:
| Your name.  This is used in a few places in the default template to refer
| to in the default copyright messages.
> Author Name [greg]: Greg Schafer

That's all. Create project? [Y/n] Y
```

.lektorproject

- name
- locales
- deployment settings
- external packages

```
pythonweb-demo$ tree
.
├── assets
|   └── static
|       └── style.css
├── content
|   ├── about
|   |   └── contents.lr
|   ├── blog
|   |   ├── contents.lr
|   |   └── first-post
|   |       └── contents.lr
|   ├── contents.lr
|   └── projects
|       └── contents.lr
├── models
|   ├── blog-post.ini
|   ├── blog.ini
|   └── page.ini
├── pythonweb-demo.lektorproject
└── templates
    ├── blog-post.html
    ├── blog.html
    ├── layout.html
    ├── macros
    |   ├── blog.html
    |   └── pagination.html
    └── page.html

10 directories, 16 files
```

Content

- 1 folder —> 1 page —> 1 URL
  - “about” folder —> /about
- contents.lr contains the content

```
pythonweb-demo$ tree
.
├── assets
|   └── static
|       └── style.css
├── content
|   ├── about
|   |   └── contents.lr
|   ├── blog
|   |   ├── contents.lr
|   |   └── first-post
|   |       └── contents.lr
|   ├── contents.lr
|   └── projects
|       └── contents.lr
├── models
|   ├── blog-post.ini
|   ├── blog.ini
|   └── page.ini
├── pythonweb-demo.lektorproject
└── templates
    ├── blog-post.html
    ├── blog.html
    ├── layout.html
    ├── macros
    |   ├── blog.html
    |   └── pagination.html
    └── page.html

10 directories, 16 files
```

```
pythonweb-demo$ cat content/about/contents.lr
title: About this Website
---
body:

This is a website that was made with the Lektor quickstart.

And it does not contain a lot of information.
```


About this Website

This is a website that was made with the Lektor quickstart.

And it does not contain a lot of information.

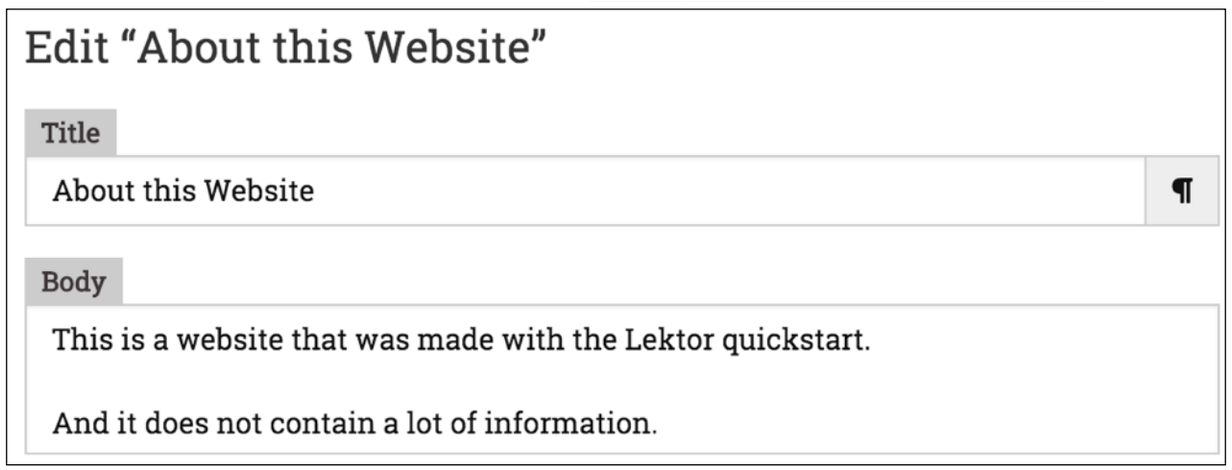
Models

- Define which fields exist and what goes in them
- Default model is “page”

```
pythonweb-demo$ tree
.
├── assets
|   └── static
|       └── style.css
├── content
|   ├── about
|   |   └── contents.lr
|   ├── blog
|   |   ├── contents.lr
|   |   └── first-post
|   |       └── contents.lr
|   ├── contents.lr
|   └── projects
|       └── contents.lr
├── models
|   ├── blog-post.ini
|   ├── blog.ini
|   └── page.ini
├── pythonweb-demo.lektorproject
└── templates
    ├── blog-post.html
    ├── blog.html
    ├── layout.html
    ├── macros
    |   ├── blog.html
    |   └── pagination.html
    └── page.html

10 directories, 16 files
```

models/page.ini

```
[model]
name = Page
label = {{ this.title }}

[fields.title]
label = Title
type = string

[fields.body]
label = Body
type = markdown
```



Templates

- Defines how to render the corresponding model as html
  - “page” model rendered by “page.html” template
- Jinja2

```
pythonweb-demo$ tree
.
├── assets
|   └── static
|       └── style.css
├── content
|   ├── about
|   |   └── contents.lr
|   ├── blog
|   |   ├── contents.lr
|   |   └── first-post
|   |       └── contents.lr
|   ├── contents.lr
|   └── projects
|       └── contents.lr
├── models
|   ├── blog-post.ini
|   ├── blog.ini
|   └── page.ini
├── pythonweb-demo.lektorproject
└── templates
    ├── blog-post.html
    ├── blog.html
    ├── layout.html
    ├── macros
    |   ├── blog.html
    |   └── pagination.html
    └── page.html

10 directories, 16 files
```

templates/page.html

```
1 {% extends "layout.html" %}
2 {% block title %}{{ this.title }}{% endblock %}
3 {% block body %}
4   <h2>{{ this.title }}</h2>
5   {{ this.body }}
6 {% endblock %}
```

templates/layout.html

```
<!doctype html>
<meta charset="utf-8">
<link rel="stylesheet" href="{{ '/static/style.css'|url }}">
<title>{% block title %}Welcome{% endblock %} - pythonweb-demo</title>
<body>
  <header>
    <h1>pythonweb-demo</h1>
    <nav>
      <ul class="nav navbar-nav">
        <li{% if this._path == '/' %} class="active"{% endif %}>
          <a href="{{ '/'|url }}">Welcome</a>
        </li>
        {% for href, title in [
          ['/blog', 'Blog'],
          ['/projects', 'Projects'],
          ['/about', 'About']
        ] %}
          <li{% if this.is_child_of(href) %} class="active"{% endif %}>
            <a href="{{ href|url }}">{{ title }}</a>
          </li>
        {% endfor %}
      </ul>
    </nav>
  </header>
  <div class="page">
    {% block body %}{% endblock %}
  </div>
  <footer>
    &copy; Copyright 2016 by Greg Schafer.
  </footer>
</body>
```

- come back to redoing the navigation

Assets

- Copied as-is to output
- css, javascript, favicon.ico, .htaccess, etc.
- Can use plugins (e.g. webpack) to render less/sass to assets folder

```
pythonweb-demo$ tree
.
├── assets
|   └── static
|       └── style.css
├── content
|   ├── about
|   |   └── contents.lr
|   ├── blog
|   |   ├── contents.lr
|   |   └── first-post
|   |       └── contents.lr
|   ├── contents.lr
|   └── projects
|       └── contents.lr
├── models
|   ├── blog-post.ini
|   ├── blog.ini
|   └── page.ini
├── pythonweb-demo.lektorproject
└── templates
    ├── blog-post.html
    ├── blog.html
    ├── layout.html
    ├── macros
    |   ├── blog.html
    |   └── pagination.html
    └── page.html

10 directories, 16 files
```

Building on the Basics

- New blog post
- Add image to blog post
- Categories/tags
- Automatic nav menu
- Databags
- Flow

People make a living doing this!

(aka adding a blog post)

- open admin
- click Blog
- add new page
- fill in title, id
- next page allows filling in blog-post fields: author, pub_date, body

Writing is hard

(aka adding pictures instead)

- image: http://oldhatcreative.com/sites/default/files/styles/blog-full-preview/public/blog-featured-images/blog.jpg?itok=gxnVBXed
- image: https://xkcd.com/1444/
- go to page
- add attachment
- edit page
- include via markdown: ![alt](path)
- show directory (cloud.png in the folder for that blogpost)
- image link broken in blog top-level because it’s relative
- options
    - render summary instead of post body (exclude image; this is what lektor-website does)
    - build image into blogpost template/model (image url still renders incorrectly)
        - seems like you’re not supposed to render attachments of one page on another page
    - some url shenanigans?
        - add a derived field to the model or construct url manually using “_slug”?
        - WORKS: absolute url_style and site.get(‘/blog').url_to(post)
    - REAL SOLUTION: reference attachment object
        - can query for it by id or first: https://www.getlektor.com/docs/api/db/query/

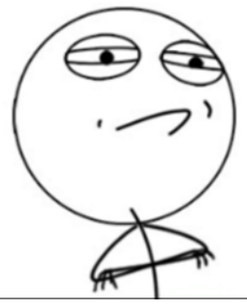
Tag all the things

(aka adding categories)

- image: https://www.drupal.org/files/x-all-the-things-template.png
- see c6c9afb
- mashed up https://www.getlektor.com/docs/guides/categories/ with existing blogstuff
- basics:
    - add blog-categories and blog-category models
        - blog-category has children.replaced_with magic
    - add blog-categories.html and blog-category.html
    - add helpers to macros/blog.html
    - add content/blog-categories/contents.lr to make category-creation available in admin
    - checkbox categories in blogpost admin
- you could do something similar with year/month archives
    - site.query('/blog').filter(F.pub_date.startswith(this)) where this is /2016 or /2016/02
    - in fact, lektor-website already does this:
        - https://www.getlektor.com/blog/2015/

Laziness is a virtue

(aka automating navbar content)



- concept: querying the database of your filesystem (content folder hierarchy)
- show lektor-website documentation recursive nav
    - generate sitemap.xml or rss
    - https://www.getlektor.com/sitemap.xml
    - https://www.getlektor.com/blog/feed.xml
        - atom feed plugin
        - https://github.com/ajdavis/lektor-atom

Laziness is a virtue

(aka automating navbar content)

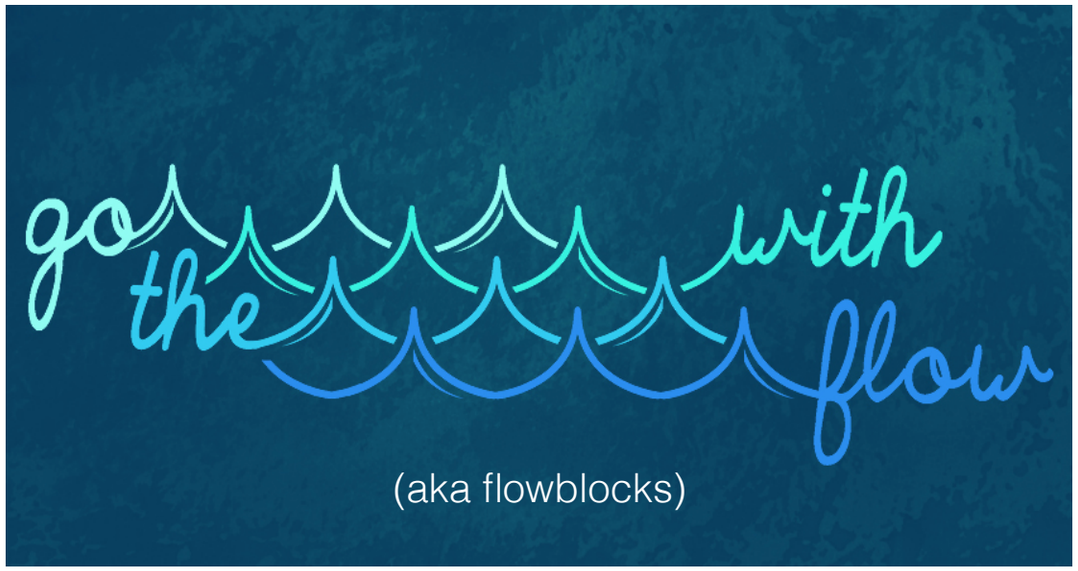
- concept: querying the database of your filesystem (content folder hierarchy)
- show lektor-website documentation recursive nav
    - generate sitemap.xml or rss
    - https://www.getlektor.com/sitemap.xml
    - https://www.getlektor.com/blog/feed.xml
        - atom feed plugin
        - https://github.com/ajdavis/lektor-atom

Databags

```
 1 [download]
 2 path = /downloads
 3 label = Download
 4
 5 [docs]
 6 path = /docs
 7 label = Documentation
 8
 9 [blog]
10 path = /blog
11 label = Blog
~
databags/menu.ini
```

```
22       <div id="navbar" class="collapse navbar-collapse">
23         <ul class="nav navbar-nav">
24           {% for id, item in bag('menu').iteritems() %}
25             <li{% if this.is_child_of(item.path) %} class="active"{% endif
26               %}><a href="{{ item.path|url }}">{{ item.label }}</a></li>
27           {% endfor %}
28         </ul>
29       </div>
30     </div>
templates/layout.html                                          19,9
```

- demo locally hosted lektor-website
  - live lektor site has changed (lektor 2.0)



- image: http://www.careerhubblog.com/.a/6a00d834516a5769e201a73dd63073970d-pi
- banner, text flowblocks on blog posts
    - http://localhost:8080/admin/root:blog:hello-lektor/edit
- slideshow, banner, text flowblocks on root page
    - http://localhost:8080/admin/root/edit
- docs:
    - https://www.getlektor.com/docs/content/flow/
    - https://www.getlektor.com/docs/models/flow/

lektor-website is a good reference

but rewind to eaeaec7 or so to avoid lektor 2.0 errors

Thanks!

Questions?

Thanks!

Questions?