

Site Architecture

The architecture of a Web site is the organization of all its pages, and how the pages relate to one another. A good site architecture matters to you as a site's designer, as it helps you to easily locate and edit your pages, and link them to one another.

Site architecture is equally important to your site's visitors. A sensible URL structure and a site navigation that reveals the general contents of your site increase the likelihood that users will understand what's on your site, how to find it, and where they are relative to the rest of your site.

To build a manageable Web site involves developing a thoughtful, scalable architecture for its pages—and a Web-like environment to test it in. This chapter looks at some of the choices you will have to make in developing your site's architecture once you have a local Web server running to test it in.

There are three types of architectures that are commonly used on Web sites:

- **File-oriented architecture**, which places all of the XHTML pages of a site in the root Web folder
- **Folder-oriented architecture**, which places related pages into separate folders off of the root Web folder

One alternative method for controlling the clutter of individual files in your root Web folder while also helping users orient themselves within your site is to create folders for each of your major site areas.

For example, rather than having a portfolio overview located at <http://example.com/portfolio.htm> , your portfolio can be located at <http://example.com/portfolio/> , thanks to the magic of the index file (see “The Index File” sidebar). Then, all of your portfolio items, like a company newsletter that you designed, can be stored in the portfolio folder, and accessed at URLs like <http://example.com/portfolio/company-newsletter.htm> . The benefit of folders is that users can cut down the URLs to move up to higher levels in a site. In other words, visitors to your site can go to the address bar in their browsers and delete the file name off of most Apache Web servers are configured to serve `index.htm` when either the root of a site or a folder is requested (e.g., <http://example.com/> or <http://example.com/contact/>). If you save a file named `index.htm` in your root, but still see a listing of files, you may need to configure your Web server by adding a line that looks something like this to your `.htaccess` file: `DirectoryIndex index.htm index.html index.php`
If the server finds `index.htm` , it will display that; otherwise, it looks for

the .html or .php versions of index .

If you do not create your own index file, your Web server might list all of the files and folders in a given directory. To prevent your Web server from doing that in indexless folders (such as your media folder), add this line to your .htaccess as well: Options -Indexes

That should prevent people from snooping the files on your site, in case you forget to save an index file in a folder where you have files that aren't quite ready for the world. There are additional .htaccess directives available at this book's companion site, <http://sustainablewebdesign.com/>

- **Data-driven architecture,**

which typically relies on databases and the Web server to mimic file and folder references. Each of these types of architectures has its benefits and appropriate applications, depending on the size and type of site that uses them. If you decide to build a site using WordPress or another blogging or content management system (CMS), your site's architecture with that system can be entirely dynamic. For example, <http://example.com/resume/> on a WordPress site would not point to a resume folder; instead, WordPress uses the resume/ part of the URL to pull your resume out of its database. (See Chapter 22 for information about configuring WordPress and your Web server to use these so-called pretty URLs.) However, if you're not yet ready to make the leap to WordPress, opting for a consistent, folder-oriented structure may make it easier for you to transfer your site to WordPress or another CMS's control later. To Google and to your users, a URL is a URL, whether it points to actual files and folders, or later to an abstract reference in a database.

Simple: File-Oriented Architecture

The most basic site architecture is created by saving all of your XHTML files right into the root of your Web folder. This keeps your URLs short and simple, in a pattern like <http://example.com/mypage.htm> . Having all of your XHTML files located in the root of a site may not be a problem if there are only a dozen or so pages on your site. But if the site grows to several dozen or more, having a massive list of all of the .htm files may make it difficult to find the page you want to edit. Scale presents another problem for a designer who dumps all pages into the root of a site, regardless of the site's size. Pages that are related to one another, such as pages for individual portfolio items, will not necessarily be grouped together in file listings, which are ordered alphabetically or by modification date. With a file-oriented architecture, users may become needlessly disoriented

as well. That is, if all of your pages are kept in the root Web folder, but there are distinct areas of your site, a file-only URL does not reveal anything about the user's location within the larger structure of your site—or the context of a given page. (Navigation might help suggest context—but you shouldn't put everything in the navigation, either.) Consider the difference between <http://example.com/fruit.htm> and <http://example.com/paintings/fruit.htm> ; which page can you better guess the contents of? The latter provides more than a hint and is the product of a folder-oriented architecture the end of the URL and come upon, for example, an overview page at <http://example.com/portfolio/> . And that is what is meant by a shallow architecture and navigation: a long URL like <http://example.com/portfolio/design/> newsletters/ represents a deep architecture, presumably (or ideally) with overviews or landing pages at each level. Representing those in navigation or promotional links becomes a challenge—as does sharing URLs in email and elsewhere. Being selective of materials for a site and coming up with a shallow architecture help prevent a site from becoming needlessly complex. That being said, even for areas of your site that might have only one page, you can still use a shallow folder-style architecture. You might save your resume, for example, as <http://example.com/resume/> .

Note 1:

ARCHITECTURE, PATHS, AND NAVIGATION

Site architecture matters also when it comes time to start linking your pages together, whether through site navigation or contextual links in your site's content. To link to resources within your site requires an understanding of URL paths, which instruct the browser to load different resources from your site onto a page (images or other media, as well as CSS and JavaScript files), or to take visitors to different pages.

Note 2:

Absolute, Relative, and Root-Relative Paths

There are three types of paths that you can write for your links: absolute, relative, and root-relative. To keep Web sites portable and to make their development easier (especially using an XAMPP installation's <http://localhost/> URL), it's generally preferable to use relative or root-relative links.

Absolute Links

Absolute links (sometimes called absolute URLs or absolute paths) include your full domain name and the name of the page/resource. For example, the absolute link to your resume might be <http://example.com/resume.htm> . Absolute URLs are what people commonly share in email and what must be used to link one Web site to another.

However, aside from the absolute link to your Web site's home page in the header area of your pages , it's usually not a good idea to use absolute links to pages within your own site: not only are they longer, but if you should switch domain names or set up archives of your site at a sub domain, for example, [http://archive .example.com/resume.htm](http://archive.example.com/resume.htm) , any absolute <http://example.com/> URLs in your links will no longer refer to items within the same version of the site.

Relative Links

To make sites more portable, you can use relative links, which are links created relative to the current document's place in the site architecture.

In a site with a file-oriented architecture, where all files exist directly in the root Web folder, relative links are very easy to write: to link from any page in the site to, for example, your resume, you would just write

```
<a href="resume.htm">view my resume</a> .
```

But if your portfolio were in one folder and your resume in another (and saved as `index.htm`) , to link to your resume from a page in your portfolio folder, you would have to write `` or `` . The `../` tells the server to move up one folder (out of `portfolio/` and up to the Web root) and then down into the resume folder. To move up two folders would be `../../` , three would be `../../..` and so on. It gets confusing pretty quickly; so let relative links serve as another argument against a deep architecture, and perhaps against relative links themselves.

Root-Relative Links

I prefer to write root-relative links in most situations; root-relative links always begin with a slash (`/`), representing the root Web folder, and proceed to the full path relative to the root of the site. Root relative links will work from anywhere in a site, even if you have a very complex architecture: `` can be used anywhere; because it starts from the root, it can always be found— provided that `resume/` is in the root Web folder. (However, root relative links will only work

during the development and testing of your site if you use something like XAMPP to run a Web server on your local computer.)

There is one case where you cannot write root-relative links beginning with a slash, though. If you are using a Web account like you might get through your school or business and it has a URL structure like

`http://university.edu/~yourusername/` ,

you'd need to prefix all of your links with `/~yourusername/` to make them root relative otherwise, the root-relative links will point to (nonexistent) files and folders off of `university.edu/` . And, of course, once you have added

`/~yourusername/` to your links, they will no longer be portable if you decide to purchase your own domain name. So add the root-relative link issue to the list of reasons why we urged you to buy your own domain name, rather than relying on hosting from your school or employer.