workflow-reference Documentation

Release 0.1

Visualspace

November 20, 2013

Contents

A collection of best practises focused at (front-end) web development: HTML, CSS, JS and tooling. This is the workflow we use internally at Visualspace.

You can find a rendered version of these best practises on Read the Docs.

For editing this documentation, please refer to the Sphinx manual and the reStructuredText Primer in particular.

Note: This documentation is currently in a very early stage of development and should not be considered ready for production use.

Contents:

CHAPTER 1

Workflows

Contents:

1.1 Single page static apps

1.2 Multi-page static sites

1.3 Generated static sites

1.4 PhoneGap applications

1.4.1 Viewport scaling

Note: Using `target-densitydpi` is deprecated. See: https://petelepage.com/blog/2013/02/viewport-target-densitydpi-support-is-being-deprecated/

CHAPTER 2

Reference

Contents:

2.1 HTML

2.1.1 HTML5

See Also:

HTML5 Rocks http://www.html5rocks.com/

HTML5 Doctor http://html5doctor.com/

W3Schools http://www.w3schools.com/

Favicon

Todo

Document favicon best practises.

Doctype

When creating pages, make sure to use a Doctype declaration. For HTML5 this means *all* HTML files should start with:

```
<!DOCTYPE html>
```

Warning: Before the doctype declaration, no spaces, characters or other content is allowed.

See Also:

Doctype at HTML5 Doctor http://html5doctor.com/element-index/#doctype

App Cache

This explicitly allows browsers to download web application for offline availability. This uses a so-called cache manifest which looks like this:

```
# <VERSION IDENIFIER>
CACHE MANIFEST
FALLBACK:
# This will cause any uncached URL to be substituted with offline.html
/ /offline.html
NETWORK:
# These resources will only be available online.
/checking.cgi
CACHE:
# These resources will be downloaded in the background and cached
/offline.html
/test.css
/test.js
/test.png
```

Note: The cached files are only updated when the contents of the manifest file have changed. Hence, it is essential that a some kind of version identifier or last modified date be added in a comment in the file.

Note: Using a cache manifest causes the cached files to be loaded instead of the online version of files, while uploads are downloaded in the background. Updated files will only be available after a reload of the page, which can be automated using JavaScript.

Note: For HTML5 offline app cache to function it is absolutely essential that the MIME type be set to *text/cache-manifest*.

See Also:

Cache manifest in HTML5 https://en.wikipedia.org/wiki/Cache_manifest_in_HTML5

A Beginner's Guide to Using the Application Cache http://www.html5rocks.com/en/tutorials/appcache/beginner/

Video

See Also:

HTML5 Video at W3Schools http://www.w3schools.com/html/html5_video.asp

Video.js http://www.videojs.com/

Responsive images

Within HTML5, several tools are available to support multi-resolution images, each one complementing the other. For example:

```
<picture>
    <source srcset="med.jpg 1x, med-hd.jpg 2x" media="(min-width: 40em)" />
    <source srcset="sm.jpg 1x, sm-hd.jpg 2x" />
    <img src="fallback.jpg" alt="" />
</picture>
```

This snippet uses the `<picture>` element together with the `srcset` attribute.

Todo

This section suggests several alternate approaches for responsive images. Add a recommendation for a particular approach or a proper argumentation allowing sensible decisions.

See Also:

W3C: Use Cases and Requirements for Standardizing Responsive Images http://usecases.responsiveimages.org/

Srcset

The srcset attribute allowed developers to specify a list of sources for an image attribute, to be delivered based on the pixel density of the user's display:

```
<img src="low-res.jpg" srcset="high-res.jpg 2x">
```

See Also:

WebKit Has Implemented srcset, And It's A Good Thing http://mobile.smashingmagazine.com/2013/08/21/webkit-implements-srcset-and-why-its-a-good-thing/

srcset-polyfill

Device pixel ratio

Picture

The picture element is an image container whose source content is determined by one or more CSS media queries:

```
<picture>
    <source src="med.jpg" media="(min-width: 40em)" />
    <source src="sm.jpg" />
    <img src="fallback.jpg" alt="" />
</picture>
```

See Also:

HTML5 adaptive images: end of round one http://html5doctor.com/html5-adaptive-images-end-of-round-one/

HTML5 <PICTURE> ELEMENT http://html5hub.com/html5-picture-element/

Picturefill

W3C http://www.w3.org/TR/2013/WD-html-picture-element-20130226/

2.2 Style sheets

See Also:

Pears are common patterns of markup & style

2.2.1 Modular architecture

Every project needs some organization. Throwing every new style you create onto the end of a single file would make finding things more difficult and would be very confusing for anybody else working on the project.

See Also:

Scalable and Modular Architecture for CSS (SMACSS) http://smacss.com/book/

The Sass Way http://thesassway.com/

File structure

There are several ways of organizing CSS into files. Whereas traditionally it was easier to put all the styles for a single site either into a single file or to simply concatenate and compress a bunch of files, modern style languages like Sass and Less allow for much smarter and potentially faster ways to set things up.

One way to setup a (S)CSS file structure is a combination of an ‘onion’ and a modular pattern. The modular pattern assures maximal reusability of design patterns and common solutions to common problems (*DRY*). The onion model helps us steer clear of *precedence* issues.

While being a work in progress, the import order in a hypothetical `main.scss` would look as follows:

```
// Modular mixins. These should generate no CSS of themselves but merely
// make mixins, variables and functions available and can be reused
// from site to site.
@import "buttons";
@import "shades";
...

// Project-specific modules (again: not producing any actual CSS output)
@import "variables";
@import "colours";
@import "fonts";

// Common site-wide components
@import "reset"; // Browser reset
@import "tags"; // Tag selectors
@import "grid"; // Grid system
@import "classes"; // Common classes (object-based / SMACSS)
@import "ids"; // Common ID-referenced styles (keep these to a minimum)

// App-specific overrides of common ids and classes
// (Try to minimize tag selectors here)
@import "admin";
@import "shop";
@import "blog";
...

// Media-specific overrides of tags, classes, apps and grid.
@import "media";
```

Warning: This is a very early sketch of a mere candidate of a CSS structure which is untested and not yet ready for actual implementation. Unless you’re brave.

See Also:

How to structure a Sass project http://thesassway.com/beginner/how-to-structure-a-sass-project

Naming conventions

See Also:

Modular CSS naming conventions http://thesassway.com/advanced/modular-css-naming-conventions

2.2.2 Style Precedence

CSS Specificity is one of the most difficult concepts to grasp in Cascading Stylesheets. The different weight of selectors is usually the reason why your CSS-rules don’t apply to some elements, although you think they should have.

Every selector has its place in the specificity hierarchy. There are four distinct categories which define the specificity level of a given selector:

1. Inline styles (Presence of style in document). An inline style lives within your XHTML document. It is attached directly to the element to be styled. E.g. `<h1 style="color: #fff;">`
2. IDs (# of ID selectors) ID is an identifier for your page elements, such as `#div`.
3. Classes, attributes and pseudo-classes (# of class selectors). This group includes `.classes`, `[attributes]` and pseudo-classes such as `:hover`, `:focus` etc.
4. Elements and pseudo-elements (# of Element (type) selectors). Including for instance `:before` and `:after`.

See Also:

CSS Specificity: Things You Should Know http://coding.smashingmagazine.com/2007/07/27/css-specificity-things-you-should-know/

Understanding Style Precedence in CSS: Specificity, Inheritance, and the Cascade http://www.vanseodesign.com/css/css-specificity-inheritance-cascaade/

2.2.3 Browser reset

A CSS Reset (or "Reset CSS") is a set of CSS rules that resets the styling of all HTML elements to a consistent baseline across browsers.

Todo

Include one of the following alternatives as bad practise and the others as explicitly deprecated.

- *normalise.css*?
- *Compass*' CSS reset
- Eric Meyer's original

See Also:

What Is A CSS Reset? http://www.cssreset.com/what-is-a-css-reset/

Eric Meyer's original Reset CSS http://meyerweb.com/eric/tools/css/reset/

normalise.css

normalise.css

A modern, HTML5-ready alternative to CSS resets.

Normalize.css makes browsers render all elements more consistently and in line with modern standards. It precisely targets only the styles that need normalizing.

See Also:

http://necolas.github.io/normalize.css/

2.2.4 Sass

Sass is an extension of CSS that adds power and elegance to the basic language. It allows you to use variables, nested rules, mixins, inline imports, and more, all with a fully CSS-compatible syntax. Sass helps keep large stylesheets well-organized, and get small stylesheets up and running quickly, particularly with the help of Compass.

See Also:

Sass reference

Media queries

As of 3.2 (the current release), Sass has smart support for CSS3 media queries. This allows for patterns like:

```
$information-phone: "only screen and (max-width : 320px)";

@media #{$information-phone} {
  background: red;
}
```

This compiles to:

```
@media screen and (max-device-width: 320px) {
  background: red;
}
```

See Also:

http://thesassway.com/intermediate/responsive-web-design-in-sass-using-media-queries-in-sass-32

2.2.5 Compass

Compass is a CSS authoring framework based on Sass providing:

- Cross browser CSS3 mixins that take advantage of available pre-spec vendor prefixes
- Mixins for common typography patterns.
- Mixins for other common styling patterns.
- An optional *Browser reset* component.
- Page layout modules for: grid backgrounds, sticky footers, stretching.

See Also:

Compass Reference http://compass-style.org/reference/compass/

2.2.6 Grid systems

Several grid systems exist to make the life of web designers easier. We currently recommend the usage of the *Susy* responsive grid system, unless *Twitter's Bootstrap* is used, which works better with the already included grid system.

Susy

Susy is a responsive grid system for Compass.

See Also:

Using Susy with Yeoman http://susy.oddbird.net/guides/getting-started/#start-yeoman

Susy documentation http://susy.oddbird.net/

2.2.7 CSS Workflow

See: https://vimeo.com/15982903

2.2.8 Viewport

For modern web development, we have to account for several types of viewports:

1. The visual viewport; the part of the page that's currently on-screen.
2. The layout viewport; the viewport referenced to in CSS.
3. The ideal viewport, where the layout viewport is equal to the visual viewport.

In responsive designs we generally want an ideal viewport and adjust the elements to the available pixels instead of zooming the whole site. For this we use the viewport meta tag in the HTML header to disable zoom and set the width of the layout viewport equal to that of the device:

```
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalabl
```

For full-screen applications which are not meant to scroll, also set the height of the viewport to the device height:

```
<meta name="viewport" content="width=device-width, height=device-height, initial-scale=1, maximum-
```

Note: The device pixels (`device-width` and `device-height`) are not necessarily equal to actual screen pixels due to the device pixel ratio. See *Device Independent Pixels*.

See Also:

CSS Device Adaptation With @viewport http://blog.teamtreehouse.com/thinking-ahead-css-device-adaptation-with-viewport

Stop using the viewport meta tag (until you know how to use it) http://blog.javierusobiaga.com/stop-using-the-viewport-tag-until-you-know-ho

Mozilla: Using the viewport meta tag to control layout on mobile browsers https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag

Mobiles and Tablets – Viewport Sizes http://i-skool.co.uk/mobile-development/web-design-for-mobiles-and-tablets-viewport-sizes/

Browser compatibility — viewports http://www.quirksmode.org/mobile/tableViewport.html

Device Independent Pixels

2.2.9 Device Independent Pixels

Todo

There are several references here, with varied quality and usability. Please remove what's not usable and summarise the useful bits.

See Also:

A Pixel is not a Pixel by Peter-Paul Koch http://fronteers.nl/congres/2012/sessions/a-pixel-is-not-a-pixel-peter-paul-koch

Towards A Retina Web http://coding.smashingmagazine.com/2012/08/20/towards-retina-web/

A Pixel Identity Crisis http://alistapart.com/article/a-pixel-identity-crisis/

Device pixel ratio

Viewport

2.2.10 Device pixel ratio

`devicePixelRatio` is the ratio between physical pixels and device-independent pixels (dips) on the device:

```
window.devicePixelRatio = physical pixels / dips
```

See Also:

devicePixelRatio http://www.quirksmode.org/blog/archives/2012/06/devicepixelrati.html

More about devicePixelRatio http://www.quirksmode.org/blog/archives/2012/07/more_about_devi.html

Device Independent Pixels

Viewport

2.3 JavaScript

2.3.1 Asynchronous Module Definition

Asynchronous module definition (AMD) is an API for modular JavaScript development such that defined modules can be loaded asynchronously while automatically taking care of loading dependencies.

It is useful in improving the performance of websites by bypassing synchronous loading of modules along with the rest of the site content. In addition to loading multiple JavaScript files at runtime, AMD can be used during development to keep JavaScript files encapsulated in many different files.

Defining a module definition in AMD looks as follows:

```
define(function (require) {
    var dependency1 = require('dependency1'),
        dependency2 = require('dependency2');

    return function () {};
});
```

For AMD dependencies, the extension `.js` is automatically added to modules, so `dependency1` will be loaded from `dependency1.js`.

Note: A common alternate API is to define requirements when calling the `define()` function. While sometimes shorter, this quickly becomes dreadful to read and is therefore considered an anti-pattern.

For example (note the lack of readability):

```
define([ "require", "jquery", "blade/object", "blade/fn", "rdapi",
         "oauth", "blade/jig", "blade/url", "dispatch", "accounts",
         "storage", "services", "widgets/AccountPanel", "widgets/TabButton",
         "widgets/AddAccount", "less", "osTheme", "jquery-ui-1.8.7.min",
         "jquery.textOverflow"],
function (require,   $,        object,         fn,         rdapi,
          oauth,   jig,         url,         dispatch,   accounts,
          storage,   services,   AccountPanel,           TabButton,
          AddAccount,           less,   osTheme) {

});
```

See Also:

Asynchronous module definition https://en.wikipedia.org/wiki/Asynchronous_module_definition

Require.js

RequireJS is perhaps the most used AMD loader. It is optimized for in-browser use but can also be used with *Node.js* to build and optimize JS before distributing.

See Also:

RequireJS API http://requirejs.org/docs/api.html

Using RequireJS with jQuery http://requirejs.org/docs/jquery.html

RequireJS Optimizer http://requirejs.org/docs/optimization.html

2.3.2 Template libraries

Several client-side template languages exist, the most elementary one being the one included in Underscore.

Handlebars

Handlebars is an extendable but compatible variant of the Moustache minimal logic-less template library.

See Also:

Handlebars http://handlebarsjs.com/

Moustache http://mustache.github.io/

2.3.3 MVC/MVP libraries

Backbone

Backbone.js gives structure to web applications by providing **models** with key-value binding and custom events, **collections** with a rich API of enumerable functions, **views** with declarative event handling, and connects it all to your existing API over a RESTful JSON interface.

Backbone requires Underscore and is commonly used with a *templating library* and a *DOM library*.

See Also:

Backbone.js http://backbonejs.org/

2.3.4 Underscore

Underscore is a util library required by Backbone, including a minimalist template engine.

It provides 80-odd functions that support both the usual functional suspects: map, select, invoke — as well as more specialized helpers: function binding, javascript templating, deep equality testing, and so on. It delegates to built-in functions, if present, so modern browsers will use the native implementations of forEach, map, reduce, filter, every, some and indexOf.

Note: Several performance-optimized compatible drop-in replacements for Underscore exist which are *much* faster and are recommended over the original Underscore library: Lazy.js, Lo-Dash.

See Also:

Underscore.js http://underscorejs.org/

Lo-Dash http://lodash.com/

Lazy.js http://danieltao.com/lazy.js/

2.3.5 DOM Libraries

The DOM (Document Object Model) is an in-memory representation of the HTML structure in a web page, which can be accessed using so-called DOM libraries, the best example of which is jQuery.

DOM libraries provide uniform access for iterating over, reading, manipulating and responding to events on live elements in the browser.

See Also:

jQuery http://jquery.com/

DOM Introduction http://www.quirksmode.org/dom/intro.html

DOM on Wikipedia https://en.wikipedia.org/wiki/Document_Object_Model

Zepto

Zepto is a minimalist JavaScript library for modern browsers with a largely jQuery-compatible API. Because Zepto lacks support for Internet Explorer, it is much smaller and faster than jQuery while providing largely equivalent functionality.

As such, it can often be used as a drop-in replacement using the following snippet for jQuery fallback on IE:

```
<script>
document.write('<script src=' +
('__proto__' in {} ? 'zepto' : 'jquery') +
'.js><\/script>')
</script>
```

See Also:

Zepto.js http://zeptojs.com/

2.3.6 Node.js

Node.js is a platform for easily building fast, scalable network applications using JavaScript.

See Also:

The Node Beginner Book http://www.nodebeginner.org/

A guided introduction to Node.js https://www.youtube.com/watch?v=jo_B4LTHi3I

Node.js API docs http://nodejs.org/api/

NPM

Node Package Manager. Installs, publishes and manages node programs.

By default, NPM installs packages and dependencies in the current directory, yielding the equivalent of Python's VirtualEnv. This is a particular convenience when installing project dependencies, for example:

```
git clone git@github.com:alexyoung/nodepad.git nodepad
cd nodepad
npm install

node app.js
```

This installs Nodepad, a Node notepad part of a tutorial series on DailyJS.

Alternately, to install packages globally use the `-g` option. For example:

```
npm install -g yeomen
```

This makes sure the `yo` command of *Yeoman*, *Grunt* and other commands are available regardless of the *Current Working Directory*.

Web application frameworks

There exist several frameworks to aid in the development of Node web applications. Some of these are:

- express
- partial.js

2.4 Tools

Tools used for *HTML*, *Style sheets* and *JavaScript* in web development.

2.4.1 Yeoman

Yeoman bundles Grunt and Bower with the scaffolding tool Yo used to setup a new web application as follows:

```
yo webapp
```

See Also:

Yeoman http://yeoman.io/

Yo https://github.com/yeoman/yo

2.4.2 Grunt

Grunt is a build tool for compiling static HTML, CSS, JS and the likes from source files such as SCSS. It can be used to automatically recompile, show and/or reload files in the browser by running:

```
grunt watch
```

See Also:

Grunt http://gruntjs.com/

2.4.3 Assemble

Assemble is a static site generator for use with *Grunt*. Starting an assemble project is easy with *Yeoman*:

```
npm install -g generator-assemble
mkdir project && cd project
yo assemble
```

See Also:

Using assemble with Yeoman (adding Yeoman to an existing project) http://www.fettblog.eu/blog/2013/09/02/using-assemble-io-with-yeoman-ios-webapp-gruntfile/

assemble http://assemble.io/

Yeoman assemble generator https://github.com/assemble/generator-assemble

2.4.4 Bower

Bower is used like a package manager for client-side JS, CSS and other packages. It automatically installs, updates and manages libraries such as *Twitter's Bootstrap*. For example, installing *Backbone* is easy:

```
bower install backbone
```

This will also include Backbone dependencies such as *Underscore*.

See Also:

Bower http://bower.io/

2.4.5 Twitter's Bootstrap

Bootstrap is a comprehensive front-end framework consisting of:

- A basic HTML templates and good examples.
- CSS with a grid systemm, sensible defaults for tags and styling for UI elements.
- Reusable components built to provide iconography, dropdowns, navigation, alerts, popovers, and much more.
- jQuery plugins for common interaction patterns.

The original version of bootstrap is built using Less CSS but a port using *Compass* is available as Sass Bootstrap.

See Also:

Bootstrap http://getbootstrap.com/

Sass Bootstrap http://alademann.github.io/sass-bootstrap/

2.4.6 Cross-browser testing

It is essential to test the design and functioning of a site across a range of different browsers and devices. To make this simpler, several services exist to create screenshots of webapps in different browser environments and/or to have live access to apps on different browsers and devices.

See Also:

BrowserStack http://www.browserstack.com/

SauceLabs https://saucelabs.com/

2.4.7 Polyfills

Tools allowing new HTML and CSS features to be used in browser that do not (yet) support them.

Picturefill

An adaptive ('retina') images approach that you can use today that mimics the proposed *Picture*.

See Also:

https://github.com/scottjehl/picturefill

srcset-polyfill

See Also:

https://github.com/borismus/srcset-polyfill

2.5 Unix

Elementary survival guide for the UNIX terminal. This guide assumes the bash shell is used.

2.5.1 Concepts

A good reference for computer terms can be found at Computer Hope.

Shell

A shell or command-line interpreter is a simple textual interface allowing users to execute commands on a UNIX system. Typically, a shell displays the Command Prompt and allows users to type in commands which will be execute by the press of the return key.

A commonly used shell is bash.

See Also:

Shell on Computer Hoper http://www.computerhope.com/jargon/s/shell.htm

Shell https://en.wikipedia.org/wiki/Shell_(computing)#Text_.28CLI.29_shells

Current Working Directory

The current directory or current working directory is the directory which is currently open in the user's terminal. The value of the working directory can usually be read from the command prompt:

```
drbob@swordfish ~/Development/workflow-reference/unix $
```

In this example ~`/Development/workflow-reference/unix` is the working directory where ~ is a common abbreviation for the user's home directory.

Note: The specific command prompt might look different depending on the configuration of your particular computer.

The value of the working directory can be found at any time using the `pwd` command:

```
drbob@swordfish ~/Development/workflow-reference/unix $ pwd
/Users/drbob/Development/workflow-reference/unix
```

See Also:

Current Directory on Computer Hope http://www.computerhope.com/jargon/c/currentd.htm

Current Working Directory Definition http://www.linfo.org/current_directory.html

Working directory on Wikipedia https://en.wikipedia.org/wiki/Working_directory

Home Directory

This is the directory where the user stores all of his or her personal information and files as well as log in scripts and user information. The user's home directory is commonly abbreviated as ~.

Returning to the user's home directory from any other directory can be accomplished with the cd command:

```
drbob@swordfish ~/Development/workflow-reference $ cd
drbob@swordfish ~ $
```

The *Current Working Directory* is now equal to the user's home directory so that the full path name to the home directory can be found through pwd:

```
drbob@swordfish ~ $ pwd
/Users/drbob
```

See Also:

Home Directory on Computer Hope http://www.computerhope.com/jargon/h/homedir.htm

Command Prompt

See Also:

Command Prompt on Computer Hope http://www.computerhope.com/jargon/c/commprom.htm

Path Name

See Also:

Path Name on Computer Hope http://www.computerhope.com/jargon/p/path.htm

2.5.2 Commands

Some common UNIX commands.

Change Directory (cd)

Changes into a particular (sub)directory or returns to the user's home directory when no (sub)directory is specified.

See Also:

https://en.wikipedia.org/wiki/Cd_(command)

pwd

Returns the name of the *Current Working Directory*.

2.6 Design patterns

A design pattern is a general reusable solution to a commonly occurring problem within a given context.

See Also:

Design patterns https://en.wikipedia.org/wiki/Software_design_pattern

Django Design Philosophies https://docs.djangoproject.com/en/dev/misc/design-philosophies/

Seven Principles Of Software Development http://c2.com/cgi/wiki?SevenPrinciplesOfSoftwareDevelopment

2.6.1 Don't Repeat Yourself

Every distinct concept and/or piece of data should live in one, and only one, place. Redundancy is bad. Normalization is good.

The framework, within reason, should deduce as much as possible from as little as possible.

See Also:

DRY on the Portland Pattern Repository http://c2.com/cgi/wiki?DontRepeatYourself

DRY https://en.wikipedia.org/wiki/Don't_repeat_yourself

2.6.2 Keep It Simple, Stupid!

Design is not a haphazard process. There are many factors to consider in any design effort. All design should be as simple as possible, but no simpler. This facilitates having a more easily understood, and easily maintained system. This is not to say that features, even internal features, should be discarded in the name of simplicity. Indeed, the more elegant designs are usually the more simple ones. Simple also does not mean "quick and dirty." In fact, it often takes a lot of thought and work over multiple iterations to simplify. The payoff is software that is more maintainable and less error-prone.

See Also:

KISS https://en.wikipedia.org/wiki/KISS_principle

2.7 Design

2.7.1 Perceived size calculation

See Also:

Web Type, Meet Size Calculator http://alistapart.com/blog/post/web-type-meet-size-calculator

CHAPTER 3

To do

Todo

Include one of the following alternatives as bad practise and the others as explicitly deprecated.

- *normalise.css*?
- *Compass*‘ CSS reset
- Eric Meyer’s original

(The *original entry* is located in /var/build/user_builds/workflow-reference/checkouts/latest/reference/css/index.rst, line 129.)

Todo

There are several references here, with varied quality and usability. Please remove what’s not usable and summarise the useful bits.

(The *original entry* is located in /var/build/user_builds/workflow-reference/checkouts/latest/reference/css/index.rst, line 289.)

Todo

Document favicon best practises.

(The *original entry* is located in /var/build/user_builds/workflow-reference/checkouts/latest/reference/html/index.rst, line 30.)

Todo

This section suggests several alternate approaches for responsive images. Add a recommendation for a particular approach or a proper argumentation allowing sensible decisions.

(The *original entry* is located in /var/build/user_builds/workflow-reference/checkouts/latest/reference/html/index.rst, line 118.)

Indices and tables

- *genindex*
- *search*