Developer Documentation

Release 1.0

rehabstudio

August 20, 2014

CONTENTS

CHAPTER

ONE

CODING STANDARDS

1.1 Frontend Standards

1.1.1 Low-hanging fruit

The long-term objective is that all the code we produce has a single voice. This means easier code maintenance in the future, and increases portability.

General

- Use four spaces for tab indentation.
- Be a good citizen. Consider your colleagues now and in the future.

HTML

- Double-quote all attributes which require values
- Don't use values for Boolean attributes
- Don't use closing slashes on empty elements

```
<img src="foo.png" hidden>
```

- For styling, a class is always preferred to an id; reserve the id for truly unique features such as an attached JS event.
- Class names should be lowercase and hyphenated.
- If adding a class just as a JS hook, use the js- prefix on the name.

```
<div id="unique-1" class="component js-trigger"></div>
```

- Always use appropriate elements for the task at hand; for example, always use a button to submit a form, never another element that looks like a button but has behaviour added with script.

```
//Never this
<a onclick="submit()" class="button">Submit</a>

//Only this
<button type="submit">Submit</button>
```

- Order rules by property group, and break each rule onto a separate line.
- Leave a single space between the colon and the first value.

- Single-quote all string values, including inside the URL function.
- Except where specifically required (e.g. a time value for transitions), do not use a unit with a zero value; 0px is the same as 0em, so simply use 0.

```
div {
/* Position */
    z-index: 100px;
/* Box Model */
    margin: 0;
/* Appearance */
    background-image: url('foo.png');
/* Behaviour */
    animation: foo 1s;
}
```

- Don't use **id** selectors to apply rules.
- Use a new line for each selector.

```
/* Not this */
#foo, .bar { }

/* Only this */
.foo,
.bar { }
```

- When using **class** names, don't specify the element in the selector unless there is a specific reason for it (e.g. specificity)

```
/* Not this */
div.foo { }

/* Only this */
.foo { }
```

- Avoid **!important** wherever humanly possible.
- Don't use long selector chains; if you're going past two selectors, consider using a new class instead.

```
/* Not this */
div ul li a { }

/* Only this */
.list-link { }
```

- When listing vendor prefixes, always have the unprefixed property name last.

```
-webkit-transform: none;
transform: none;
```

Sass

- Keep all variables in a single variables file and use generic names; like classes, variables can be used in multiple places, and their function can change. An exception to this would be if you're using loops and require a variable specifically for the current scope.

```
/* Not this */
$textWhite: #fff;

```

```
/* Only this */
$keyColorMain: #fff;
```

- Do not nest more than three levels deep. While nesting is powerful, it can have a negative impact on readability and, therefore, maintainability.
- When available (in release 3.3), use source maps for easier debugging (http://devtoolsecrets.com/secret/editing-use-sass-source-maps.html).
- If not using Compass or an autoprefixer, make a Mixin for any CSS property which requires vendor prefixes.

```
@mixin transform($args...) {
    -webkit-transform: $args;
    transform: $args;
}
```

- Consider placeholder selectors for repetitive code instead of extending other typed classes. Placeholder selectors will not be written to the stylesheet.

```
%gutter {
    margin:0;
    padding:0;
}
.btn {
    @extend %gutter;
    background: #c9c9c9;
}
```

JavaScript

- Comma-separate blocks of variables rather than repeat var.
- Outer-encase all strings in single-quotes.
- Defined names should be camelCased.
- Use an underscore prefix to name private variables.
- Variables with a Boolean value should be prefixed with is.

```
var _foo = 'Hello World',
    _barBaz = 1234,
    isBoolean = true;
```

- Use line breaks to show the contents of a function or conditional statement.

```
// Not this
if(this){that;}

// Only this
if (this) {
    that;
}
```

Do not perform calculations or access the DOM when defining loops.

```
// Not this
for (var i=0; i < (foo * 5); i++) {alert(i);}

// Only this
```

```
var fooTotal = foo * 5;
for (var i=0; i < fooTotal; i++) { alert(i); }

// Not this
for (var i=0; i < $('.foo').length); i++) {alert(i);}

// Only this
for (var i=0,fooLen = $('.foo').length; i < fooLen; i++) { alert(i); }
```

- Lint your code automatically if your text editor allows, or manually if not. Use JSHint rules (http://jshint.com/).

Comments

- Comment everything, all the time; all code should be minified before going into production, so trying to save space at this point is a false economy.
- Code should follow phpDocumentor (http://bit.ly/3FPH7g) standards.
- For CSS, comment uncommon practices or decisions.
- Comment class methods and loose functionality, along with any other complex logic that may benefit from them.
- Document the parameters and return types of your methods and write an accurate description of the purpose of the method.
- If the method is complex and has multiple use syntaxes, document them as examples in the comment block.

```
// Ensures value can't go below zero or beyond the maximum possible value.
var newX = Math.min(maxDrag, Math.max(0, newX));
/**
 * Filters the data source to create a subset matching the chosen date.
 *
 * @param string requestDate - Following YYYY-MM-DD syntax.
 * @return array.
 */
filterByDate: function(requestDate, implementOffset) {
}
/**
 * Returns a User record along with nested Goal records, recent
 * activity and any notifications to be shown.
 *
 * Example Usage:
 * APIWrapper.getUserDetails({ facebookToken: '123456' });
 *
 * @param object requestData - Contain either Facebook or Instagram tokens.
 * @return object - jQuery promise (resolved) with User record.
 */
getUserDetails: function(requestData) {
}
```

Further reading

- https://github.com/necolas/idiomatic-css
- https://github.com/stubbornella/oocss-code-standards
- https://github.com/rwaldron/idiomatic.js
- https://github.com/anthonyshort/idiomatic-sass

1.2 Backend Standards

1.2.1 PHP

Existing Standards and Code Sniffer

Thankfully, there are already agreed PHP community coding standards, and rehabstudio adopts these.

When coding in CakePHP, please use the CakePHP coding standard, which is available for CodeSniffer, this makes it super simple to integrate into your Grunt/Gulp setup.

- http://book.cakephp.org/2.0/en/contributing/cakephp-coding-conventions.html
- https://github.com/cakephp/cakephp-codesniffer

Should you be working with any other framework, it is acceptable to use PSR2 or PSR1, but you must use one of these 3 standards!

Supplimentary

Furthermore, please observe the following:

- File encoding is UTF-8
- The default permissions for folders are octal 0755, for files octal 0644. Only if the file must be executable (i.e. from console) use octal 0744 for files.
- Never do a SELECT * unless you really need every field return, SELECT * queries have a massive overhead. (This also applies to "find all" commands within frameworks).

1.2.2 Python

Where applicable (in most cases), please adhere to the PEP 8 style guide. The PEP 8 guide itself advises when it is acceptable to disregard this.

Coding standards are extremely important for ensuring a consistent quality of coding output from the studio. This ensures that your coding skills are up to standard, that you're working in the most efficient way, and that anyone who must pick up your work, will be able to hit the ground running.

These standards are already agreed and must be obeyed until changes are debated and published here.

CHAPTER

TWO

IMAGE FORMATS

2.1 JPEG

JPEG images are best for photographs - unless transparency is required, in which case PNG is more suitable. However, with JPEG we always need to balance quality with filesize.

To accommodate high DPI screens, save the image at double the dimensions required (e.g. if 100x100 on the web, save as 200x200). Use Save for Web and drop the quality as low as possible before any obvious visible artefacting appears in the 50% preview pane. This will have to be done by eye. On some images we can get as low as ~25% without any noticeable artefacting.

See this example: http://www.broken-links.com/tests/highdpi.html. Images on the right are double dimension, low quality, but on high DPI screens generally appear as good or better than the higher quality on the left, and file size is roughly comparable.

If the image needs to be downloaded by the user (e.g. wallpapers) the approach above won't work, so save at regular dimensions but again, keep quality as low as possible without the appearance of artefacting.

2.2 PNG

PNG are most suited for non-photorealistic images, and photo images in which transparency is required. Transparency can be expensive (file size and performance) so try to keep it contained to smaller images.

If a simple image with no more than 256 colours, with no alpha transparency, save as an 8-bit PNG. Otherwise, use 24-bit. As before, if required to suit high DPI screens, save at double the dimensions.

2.3 SVG

SVG is generally the better option for icons, charts and logos as it's scalable so suits high-DPI screens.

Avoid using filters or gradients where possible as they're expensive to performance.

CHAPTER

THREE

PRELAUNCH CHECKLIST

Things that should be checked off (where appropriate) before we consider a site ready to launch.

3.1 Front End

- Meta data included and appropriate
- Facebook OpenGraph tags properly set up
- Page titles are descriptive and SEO friendly
- Images have appropriate alt text
- Images have been optimised
- CSS/JS minified
- Favicon created and displayed
- App icons created and displayed
- Analytics installed and reporting
- 404 page exists and is informative
- Javascript console messages suppressed/removed
- Unsupported browser/platform messages in place
- Does the site have RTL locales? Have you used direction: rtl; to mirror things?

3.2 Back End / Platform

- Default CMS user account created
- Test data removed from DB
- Debug modes turned off
- Setup Sentry logging/reporting
- Lockdown/htpasswd removed
- Ensure GZIP is serving assets.
- Third party
- Facebook Sandbox turned off

- Ensure all third party paid services have billing set up (no trials)

3.3 Sysadmin

- Ensure infrastructure backup is in place
- Ensure DB backup is in place
- New server environment is ready for live
- Logging system updated for live

3.4 Post Launch

If using any Facebook services, such as sharing, remember to also enter your URL into the Facebook Debugger (https://developers.facebook.com/tools/debug/) after your site goes live, to ensure the Facebook Cache is cleared. This will prevent 403 Authentication Required errors when sharing your site now that .htpassword has been removed.

CHAPTER

FOUR

PEER REVIEWS

This document exists as a proposal for future direction. This process is not yet doctrine within rehabstudio.

"Feedback is important for engineers to grow in their jobs. By having a culture of 'everyone's code gets reviewed' you promote a culture of positive, constructive feedback. In teams without review processes, or where reviews are infrequent, code review tends to be a tool for criticism, rather than learning and growth." - Alex Gaynor

4.1 Resources

- Effective Code Review
- How to do Effective Peer Code Reviews
- Peer Code Review An Agile Process
- 11 Best Practices of Peer Code Review [pdf]
- Best Kept Secrets of Peer Code Review [pdf]

4.2 Proposed Process

- Everybody gets code reviewed, but not every day - rotating/random checks
- Work in feature branches not to be merged into develop without a review
- Keep to 30 minute sessions (no more than 400 lines of code)
- Use a checklist to compare with internal standards

4.3 Questions

- Do pre-commit reviews fit with our Git flow?
- How do we review one-man projects?

CHAPTER

FIVE

PERFORMANCE PLAN

5.1 Responsibilities

Granular detail of the responsibilities at each stage of build.

5.2 Strategy

- *Source data on target market - mobile, OS, average broadband*

5.3 Design

- Design appropriately for the target market
- Design appropriately for the target devices, e.g. performance restrictions on phone/tablet
- Export images in the appropriate format, optimised for balance of quality and file size

5.4 Production

- Set a performance budget with devs and enforce the budget on change request

5.5 Dev Ops

- Use Page Speed module for compression, caching, expires headers
- Enable appropriate number of CDNs/distribution points
- Site should scale appropriately where needs be and have sufficient server specifications to support the core platforms tasks
- Server should be located in the optimum locale for the target market
- Server should be running the minimum services required to run the application
- Implement Sitespeed.io (or similar) into deploy script

5.6 Back-end

- Render HTML templates
- Cache assets, db queries and opcode where appropriate
- Optimise code, ensuring the likes of loops and file i/o are as efficient as possible
- Compress images upon upload where possible, according to the nature of the project

5.7 Front-end

- Use progressive enhancement
- Optimise code, ensuring the likes of loops are as efficient as possible
- Optimise (minify, concatenate, compress) assets (css, js, img, svg) in workflow
- Ensure that assets are only loaded when required; use lazy or conditional loading
- Log timestamps of key moments for measurement
- Enforce the performance budget

5.8 QA

- Use Page Speed Insights (or similar) and developer tools to flag slow page load problems[1]
- Visually inspection for-over optimisation of SVG
- Run selenium grid across main 4 desktop browsers[2]
- Automated testing across supported devices, including bandwidth throttling[3]
- Enforce performance budget
- Ensure no double redirects (particularly where specific mobile content is present)
- Measure performance of external asset servers
- Check that relevant assets are benefiting from server side compression[4]

5.9 Notes

1. Analyse what specific elements are causing issue and ensure that optimisations suggested above have been applied.
2. Use our own tool and check for any spinning slow page loads (general check but focus on anything identified in page speed insights analysis). Acceptable limits for animated sites are 10 seconds for primary non-cached first page load and no more than 4 seconds for cached page load. Any exceeding of these numbers needs accounted for. Load times for sites with no animation should be no more than 5 and 2 sec respectively. If no CMS present then lower again. Any exceptions need explained. Page load time is that returned by browser developer tools. Performance test runs should be performed against staging site which should be comparable cpu to the live site. A subsequent run is required if CDNs are to be enabled for go live. (up to 1 day required for propagation)
3. If it is agreed with customer that users are accessing apps over slower connections but this is an exception to standard QA testing

4. Developer responsibility but needs checked by QA.

CHAPTER

SIX

RESOURCES

6.1 rehabstudio Resources

- rehabstudio SOW Template

  Obviously every project is different, but this SOW Template serves as a baseline document which we can review and edit as time goes on. Edits made should be generic enough to be applicable to all projects. Please make suggested edits and inform Peter Gasston of what you've edited and why.

- Red Bull :: Build and Deployment Process Review

  Red Bull projects are typically huge and have very detailed architectured, from both a front-end and back-end perspective. This document aims to provide a developer guide to RedBull projects.

- GitHub account for rehabstudio

6.2 Educational Resources

6.2.1 Git

- All rehabstudio projects must use GIT-Flow, it's the law. http://nvie.com/posts/a-successful-git-branching-model/
- http://pcottle.github.io/learnGitBranching/index.html?demo
- http://gitref.org/
- http://progit.org/book/
- http://book.git-scm.com/index.html
- http://www.ndpsoftware.com/git-cheatsheet.html

6.2.2 JavaScript

- http://superherojs.com/

CHAPTER

SEVEN

ABOUT

This service provides a central platform to host all our development guidelines and agreed Coding Standards. This helps to ensure we have a known place-to-go to check the agreed 'rehab way' of doing things.

All of this is open to debate and change, though this is not the forum for such debate. Slack, Discourse, Email and human conversation is the medium for debate. Agreed policies are then posted here, and must be adhered to until such time as we agree to change, based on the outcome of debates.

CHAPTER

EIGHT

FORMATS

You can view this documentation online at http://devdocs.rehabstudio.com, as an eBook here or as a PDF here.

CHAPTER

NINE

YOUR CONTRIBUTIONS

We welcome your contributions and suggestions to this suite of documentation. Feel free to submit a pull-request or simply talk to other developers and let us know your ideas.

See GIT repo `/projects/RS/repos/devdocs/browse`, these pages are coded in reStructured Text markup.

Please build with: `make html && make pdf && make epub`