Developer Documentation

Release 1.0

rehabstudio

July 18, 2016

Contents

CHAPTER 1

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>
```

CSS

- 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.

- Using **@import** creates global scope so avoid duplicating variable names, even between variables and maps.
- 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.
- 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

- Always use **var** to declare variables.
- Each variable on its own line.
- 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';
var _barBaz = 1234;
var 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.
- Frontend code comments 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 2

Useful tools/Tips and Tricks

2.1 Docker

2.1.1 What is Docker?

Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications. Consisting of Docker Engine, a portable, lightweight runtime and packaging tool, and Docker Hub, a cloud service for sharing applications and automating workflows, Docker enables apps to be quickly assembled from components and eliminates the friction between development, QA, and production environments. As a result, IT can ship faster and run the same app, unchanged, on laptops, data center VMs, and any cloud.

Solomon Hykes, Docker's Founder & CTO, gives an overview of Docker in this short video: https://www.youtube.com/watch?v=ZzQfxoMFH0U

2.1.2 How we use it

We're just beginning to use Docker heavily throughout rehabstudio. We mostly use docker locally to fulfil the same purpose that Vagrant did previously (building isolated, reproducible development environments), however, we're beginning to use Docker much more heavily in production and deployment settings also. Keep tuned!

2.1.3 Installation

NOTE: The minimum required version of docker at rehabstudio is **1.3**. Docker/boot2docker 1.3.0 added support for mounted volumes when using boot2docker on OSX.

Linux

Docker is best supported on Linux, you can probably find packages for your preferred distribution here: https://docs.docker.com/installation/.

OSX

Install Docker and boot2docker following the instructions on this page: https://docs.docker.com/installation/mac/.

Next, we need to forward the appropriate ports so that we can reach the running appengine development server directly from the host OS:

```
$ VBoxManage controlvm boot2docker-vm natpf1 "aesdk,tcp,127.0.0.1,8080,,8080"
$ VBoxManage controlvm boot2docker-vm natpf1 "aesdkadmin,tcp,127.0.0.1,8000,,8000"
```

Please also ensure that the project your working exists inside your `/Users` directory in order for the auto-mounting to work.

Note: If you see a message in the terminal that looks something like this:

```
To connect the Docker client to the Docker daemon, please set: export DOCKER_HOST=tcp://192.168.59.1(
```

you can safely set the environment variable as instructed. You should also probably add this `export` command to your `.bashrc` so that it persists across terminal windows/reboots.

Windows

Not supported yet (we just haven't tried, give it a go, it might work). Pull requests very welcome.

2.2 Goro

2.2.1 What is Goro?

Goro is a "content production system" used to build **all sites** that live directly on the Google top-level domain (e.g. https://www.google.es/activate).

At its core, Goro is a web application that runs on Google App Engine and provides a set of tools and processes to site owners in order to cut down the time it takes to produce a high-quality web site.

2.2.2 Learning Resources

If you cannot access any of these learning resources, Goro itself or even the Johnny Cage repository then you will need to contact your projects ATL and request access.

General

- What's The Deal With Goro?
- Agency Guide: Goro
- High-level Guidelines

Development

- Johnny Cage repository
- First Time Developer Guide
- Goro Help Centre
- Viewing & Editing Files
- Using AngularJS in Goro
- Template Context Variables
- Template Tags

- Using Custom Formbox Forms

Linting / Styling

- Brand Studio: Agency Code Review Checklist
- Template style guide
- HTML/CSS style guide
- JavaScript style guide
- AngularJS style guide

2.2.3 Coding Standards

Projects being built through Goro need to adhere to Google's linting rules. There are various different style guides and standards across different languages. Check out the linting section of the learning resources for more specifics.

To lint JavaScript files you can use the "Linter Tool" while editing a file on Goro. Note that this will not catch AngularJS formatting rules, only plain JavaScript rules. The tool is located next to the "Actions" and "Preview" buttons. There is also a command-line linter known as **gjslint** that can be run with corresponding rulesets. This local linting will speed up development overall as you'll not have to deploy files to lint them.

2.2.4 Workflow

Creating and authoring project files can be done entirely through the Goro web application, however this has many disadvantages:

- You need an active internet connection to be able to develop.
- To ensure your JS files lint you'd need to open each one manually and check.
- Goro doesn't have real file source control and has a limited history stack.
- The project will have access to only one "branch" and preview area. If there are multiple developers on the project you'll quickly tread on each other's toes.

To combat the majority of these errors we can use a local development server for Goro known as Johnny Cage. Utilising this tool allows us to develop locally for speed and use our own local build tools such as Grunt or Gulp for things such as file uploading and linting.

Developing locally will also allow us to use our own project git repository. Doing this lets us treat the Goro branch as a preview environment and a tool for exporting files.

2.2.5 Command-line Tool

Goro has a CLI that can be installed locally. There is documentation that will help you get it installed on Mac / Linux. The CLI lets you upload files to Goro in bulk rather than uploading folders through the Goro web application.

There is an additional "tip" in the documentation which shows you how to alias *goro*, however, their method listed will only work for your current terminal session if you're on anything other than a mac.

NOTE: If you are installing the CLI tool, it only works with versions of keyring lower than or equal to *8.4.0*. At this time you'll need to update the relevant line in the *requirements.txt* of the CLI vendor download.

If you try to run *goro* from any folder and it fails then you will need to create your own command. Below is an example *goro* command that can be placed into any of your *bin* folders (*~/bin*, */usr/local/bin*) e.g. */usr/local/bin/goro*:

```
#! /bin/bash
python /path/to/your/goro/folder/goro.py "$@"
```

You may need to restart your terminal for changes to take effect. To test things work as intended, simply try to run *goro* from any folder.

This section contains an overview of some of the tools we use day-to-day at rehabstudio. Sometimes just documenting that we use them, sometimes noting any gotchas we've found or describing how we at rehabstudio use a certain tool differently than is normal. Dig in!

CHAPTER 3

Image Formats

3.1 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.

3.2 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.

3.3 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 4

Prelaunch Checklist

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

4.1 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?

4.2 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)

4.3 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

4.4 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 5

Versioning

5.1 Make your life easier with Git

Git is a distributed revision control and source code management system with an emphasis on speed, data integrity and support for distributed, non-linear workflows.

As with most other distributed revision control systems, and unlike most client–server systems, every Git working directory is a full-fledged repository with complete history and full version-tracking capabilities, independent of network access or a central server.

Like the Linux kernel, Git is free software distributed under the terms of the GNU General Public License version 2.

5.2 Resources

- GitHub Help

5.3 Generating SSH Keys

- Generating SSH Keys (Linux, Windows, Mac OS X)

5.4 Create a new repository on the command line

Follow these instructions:

```
touch README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/user/projectname.git
git push -u origin master
```

5.5 Push an existing repository from the command line

```
git push <REMOTENAME> <BRANCHNAME>
```

- More info about pushing to a remote

5.6 Why?

- It's all about team work, code backup and version control

5.7 Methodology

All our code must use the GitFlow Methodology <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>.

CHAPTER 6

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

6.1 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]

6.2 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

6.3 Questions

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

CHAPTER 7

Performance Plan

7.1 Responsibilities

Granular detail of the responsibilities at each stage of build.

7.2 Strategy

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

7.3 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

7.4 Production

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

7.5 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

7.6 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

7.7 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

7.8 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]

7.9 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 8

Docker

8.1 Docker images

All reusable Docker images should abide by the following rules:

- Where less frequently used packages are desired, consider making a base image and use that for the generic packages.
- Use the tagging structure, `0.9` for images - this should be reflected in GitHub & Dockerhub releases

8.2 Distro

For build tools, where possible we should make use of Alpine Linux as this is a very lean image. Additional packages can be added with:

```
apk add --update {packages}
```

When mirroring server environments, the distro should always match the server. For Google App Engine we should use debian:wheezy.

8.3 Dockerfile

- All images should be ephemeral
- MAINTAINER set to devops@rehabstudio.com
- WORKDIR should point at the mounted root
- COPY instead of ADD, unless you want to unpack an archive
- RUN should be used efficiently. Multi-line when possible, e.g.

```
RUN apt-get install -y curl \
        wget \
        perl
```

- CMD should be in the format `CMD ["exec", "param1", "param2"]`, not `CMD exec param1 param2`
- ENTRYPOINT don't use this unless you have a specific need to

- VOLUME to be used where data should be added to a volume on mount. The container shouldn't rely on anything being mounted from host though
- USER shouldn't be specified unless you're doing some user-specific actions inside the container. The default user (e.g. root) is fine for most containers.
- ONBUILD shouldn't be used unless there's a really good reason to do so
- Use a .dockerignore when it makes sense - exclude what doesn't need added to the container
- Cleanup after yourself. Once you have installed whatever packages you desire, RUN a clean up, e.g. *rm -rf /var/lib/apt/lists/** (This can be appended at the end of a multi-line apt-get install, making the install and cleanup a single RUN)

8.4 Mounting Volumes

For those using VirtualBox (likely most of the OSX folks) remember that you can not specify local folders outside of */Users/...* to mount. This means that if you're trying to `docker run -v /var/host/www:/var/container/www ...` you will likely have some weird results. You can either specify directories that live inside your */Users/* directory, or you can open the VirtualBox UI and add whatever folders you require as a shared folder.

Basically, it's best if you just keep everything in */Users/...* for now.

8.5 Security

It is important to remember that a docker container can be inspected and as such shouldn't contain any sensitive data, such as secrets. Where necessary, use a *.dockerignore* file to exclude sensitive (or useless...) files from the container, such as *.env* files.

If you need to make use of keys in your container, it's a good idea to do so at run time and mount said keys as a volume.

8.6 Repo

- All public images should live in a GitHub repo
- The repo should be named *docker-{DISTRO}-{NAME}*

8.7 Docker Hub

The following steps should be taken for images on the rehabstudio docker hub:

- Use https://imagelayers.io/ and include an embed
- In the long description:
- Include examples as to how to build and run the container
- Run examples should also include instructions on how to mount volumes, if relevant
- Mention that issues / comments should be raised on the GitHub page, not the Dockerhub page

8.8 README

The README.md should follow these rules:

- Include a Dockerhub embed
- Have a Usage section
- Mention any variants, if applicable (e.g. -alpine)
- Specify which docker version has been tested

CHAPTER 9

Security

9.1 OWASP Top 10

All staff should be aware of the OWASP Top 10 - details of which can be found here: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

9.2 General Security Considerations

All projects should have its security properly scoped and signed off by internal and external security teams prior to build. Some common security considerations are as follows:

- **What are the threat scenarios specific to your project?**
  - How secure is data in transport and at rest?
  - What data is stored/shared?
  - What legalities apply to this project?
  - What priviledges are available in this app & can they be abused?
  - How is data destroyed?
- **Is access effectively managed?**
  - Onboarding
  - Offboarding
  - Inter-team permissions
  - Deployments restricted to core members
  - Apps locked down prior to launch
  - Logs and backups secured
- **Have all common attack vectors been considered? e.g.**
  - SQL injection
  - Cross site scripting
  - Email Form Header Injection
  - Malicious File Upload and Execution

  - User Authentication
  - Error suppression
- **Is the server secured?**
  - Access management
  - Secure logs
  - IDS
  - Firewall properly configured
  - Can handle expected load
  - Is DDOS protection required?
  - SSL
- **Have all frameworks, third party libraries and APIs been approved?**
  - Licensing
  - Maintenance & support
  - Ensure the latest stable versions are used
  - Where applicable, state specific versions when using package management to avoid any issues caused by automatic package updates

9.3 Security Scanning Tools

All projects should have a security suite ran against it. The following tools are freely available and commonly used by staff:

- Nikto https://cirt.net/Nikto2
- Skipfish https://code.google.com/archive/p/skipfish/

CHAPTER 10

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 are the media 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 11

Formats

You can view this documentation online at http://devdocs.rehabstudio.com

You can also view this documentation as ebook or PDF formats.

CHAPTER 12

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 `https://github.com/rehabstudio/devdocs`, these pages are coded in reStructured Text markup.

For local testing, makefiles are available. These docs will automatically update the live page once a pull request is merged on GitHub.