Guidelines for External Web Development

Version 2.1

Drupal Development Guidelines

Unless agreed upon in writing, all development must take place on a locally hosted development environment (LF provided Vagrant environment) that is properly protected by full disk encryption.

Unless otherwise specified, all externally development modules must be compatible with the following:

- Nginx 2.0 (FastCGI)
- PHP 5.5 (FPM) with Zend OPcache
- MySQL 5.1
- jQuery version 1.8
- jQuery UI 1.8.7
- OpenLDAP (including SSL CA for LF servers)

All development must be done using a Vagrant project provided by the Linux Foundation. Local MAMP/LAMP/WAMP stacks as well as Docker or any other non-LF provided Vagrant setup are not acceptable development environments.

Please merge LF development branches into your own code prior to pushing code back to the LF.

All code will be subject to review by the Linux Foundation web team to ensure that they conform to the following principles:

- All code (JavaScript or PHP) must be thoroughly commented in conformity with Doxygen formatting conventions.
- Code must not duplicate existing solutions already in use within the Linux Foundation codebase.
- All code should be deployed and tested against the Linux Foundation's code base and configuration before delivery.
- Code that interacts with content must ensure that content is run through appropriate filters. Custom text formats must ensure that runnable PHP and JavaScript code can not be added into content. CSS properties that might allow JavaScript injection or other malicious activities should also be filtered out.
- DB queries must use slave DB, when applicable, and should conform to SQL standards. Queries must be optimized and sanitized to prevent injection. If new indexes are required please get LF approval.
- All UI or theme layer elements must be responsive and display properly in all supported web browsers.
- All deliverables should be committed directly to a provided Linux Foundation git repository.

Unless otherwise specified, all externally developed Drupal modules must be compatible with the following:

- Drupal version 7.43 or 8.1.2
- Existing site projects
    - Apache httpd 2.2
    - PHP 5.3 (mod_php) with APC
    - Drush 6.0
- New sites
    - Nginx 2.0 (FastCGI)
    - PHP 5.5 (FPM) with Zend OPcache
    - Drush 6.0 *(7.0.0 upgrade pending)*

Modules must be developed under the following principles:

- Modules must adhere to Drupal coding standards, security standards and best practices as outlined here: http://drupal.org/node/360052.
- Modules must pass inspection by the latest 7.x version of the Drupal Coder module http://drupal.org/project/coder without any errors or warnings.
- Modules must utilize simpletest to verify all key module functionality.
- Modules and any libraries that they depend on should be releasable under a GPL2 or similar license http://www.gnu.org/licenses/gpl-2.0.html.
- Modules should provide an extensible API where possible and applicable.
- Modules should override core functionality using Drupal's hook system. Core Drupal code must never be modified directly.
- Modules must be designed and tested to work with Drupal multisite.
- All developed features should be in a default state and should not be in conflict with existing and enabled site modules.
- Content that is meant to be regularly edited (menus, block titles and bodies, etc) shall NOT be featurized. In order to create initial content, create the blocks, menus, etc via install hooks in the _deploy module. A content editor (ability to edit nodes, menus, block titles/bodies) shall have the right to edit whatever they please without causing any features to be overridden.
- Existing site roles should be used in place of custom roles whenever possible.
- All users (with the exception of user 1) should authenticate via the Linux Foundation CAS SSO system.
- PHP, JavaScript, and CSS used in a module should be namespaced to that specific module. This ensures that CSS or JS files with identical names don't have collisions.
- Use of Drupal's core PHP module is not allowed.
- All modules will be subject to review by the Linux Foundation web team to ensure that they conform to the aforementioned standards.
- Deployment of all deliverables should be achievable by pushing code and running hook_update_n hooks within a given site's deployment module. Any exceptions to this process must be agreed upon by Linux Foundation staff.
    - Content or user-editable items shall not be featurized.
    - Entire site should be in a production ready state by running the following commands and nothing else:
        - drush si -y
        - drush en alias_deploy -y
- Unless otherwise specified, if any contributed modules are used that are not currently part of the Linux Foundation code base, they must meet all of the above guidelines and have a full (not dev, alpha, beta, or rc) release. Any exceptions to this process must be agreed upon by Linux Foundation staff.
- Drupal core should never be patched.
- If a contrib module is to be patched, relevant .patch files should be added into a site's "patches" directory for reference.
- Sites should conform to the following directory structure:
    - files directory: sites/site-alias/files (ex. sites/agl/files)
    - private files directory: sites/site-alias/files/private (ex. sites/agl/files/private)
    - site directory: sites/URL/ (ex. site/www.automotivelinux.org/)
        - with symlink to files directory: (ex. ln -s ../site-alias/files)
- All views queries should be configured to hit slave databases. This can be set via advanced->other->query settings in views configuration.

All UI or theme layer elements must be developed under the following principles:

- Unless otherwise specified, all custom themes should use the Basic theme as a foundation.
- Themes must adhere to Drupal coding standards, security standards and best practices as outlined here: http://drupal.org/node/360052.
- Wherever possible PHP logic must be placed in modules and removed from the theme layer.
- All tpl.php PHP should be limited to print functions and if/else statements. More complicated theme layer PHP should reside in template.php (assuming it can node me be moved to the module layer).
- Drupal UI best practices should be followed whenever possible: http://drupal.org/node/501452.
- HTML5 http://www.w3.org/TR/2010/WD-html5-20100304/ should be used for all markup. Given that this is a developing standard steps should be taken to ensure that all supported browsers function correctly via a progressive enhancement approach.
- CSS 3 http://www.w3.org/TR/2001/WD-css3-roadmap-20010523/ should be used for all markup. Given that this is a developing standard steps should be taken to ensure that all supported browsers function correctly via a progressive enhancement approach.
- SASS should be used to generate all CSS.
    - We prefer to use either Grunt or compass to compile SASS. Please make sure to include all config files in repo and instructions to compile.
- In conformance with WAI http://www.w3.org/WAI/intro/accessibility.php WCAG2.0 http://www.w3.org/TR/WCAG20/ accessibility standards.
- In conformance wtih WAI http://www.w3.org/WAI/intro/accessibility.php ATAG1.0 http://www.w3.org/TR/ATAG10/ accessibility standards.
- jQuery or HTML5 solutions should always be used in place of Adobe Flash.
- JavaScript UI elements must degrade gracefully when JavaScript is disabled. Look and feel must remain consistent whether or not JavaScript is disabled.
- Theme code (Javascript and PHP) must be thoroughly commented in conformity with Doxygen formatting conventions.
- Content must display properly in all supported web browsers.
- Themes must be properly responsive and work intuitively on supported IOS and Android browsers.

SugarCRM-specific Development Guidelines

- The Linux Foundation has decided to use Drupal coding standards for all project development. SugarCRM coding standards follow a very similar pattern to Drupal. All code must should indent using 2 whitespaces with no tabs.
- If development requires new Libraries or Frameworks you must have approval from LF. New libraries and Frameworks will be installed by Operations onto our Vagrant Environment.
- All code must be developed in an upgrade safe way following SugarCRM recommendations.
- If core code modification is required for a customization it must first be approved by LF and properly documented so it can be reimplemented in future releases.
- Deployment of all deliverables should be achievable by pushing code and running Repair and Rebuild. Any exceptions to this process must be agreed upon by Linux Foundation staff.
    - Content or user-editable items shall be provided as a query.
- All UI modifications must conform with current SugarCRM themes.

WordPress-specific Development Guidelines

Best Practices Conformance

- All PHP code should conform to WordPress PHP coding standards.
- All JavaScript should conform to WordPress JS coding standards.
- All inline comments should conform to established WordPress standards.
- All CSS should conform to WordPress CSS coding standards. Compiled CSS should be generated via SASS. We prefer to use either Grunt or compass to compile SASS. Please make sure to include all config files in repo and instructions to compile.
- All HTML markup should conform to WordPress HTML standards.
- **All WordPress plugins and themes will be subject to review and final approval by the Linux Foundation web team.** A list of existing or custom plugins that will be used in a given project should be submitted to the Linux Foundation web team before development begins. Any changes to this list should be discussed with the Linux Foundation web team as they arise.
- All development should align with WordPress accessibility standards.
- If you are working on a site with the Salient Child Theme active, any custom functionality must reside in the child theme rather than the parent theme. If you do not do this, you will be unable to receive any theme updates from Salient, which could leave your website performing poorly, and vulnerable to being hacked.

Development and Approval Workflow

Please familiarize yourself (really! Their docs are good!) with developing on Pantheon if you are not already experienced with their platform. Please also see Lando for your local Wordpress development needs.

Access

- If you are a solo developer, your Linux Foundation contact/manager must submit a request for Pantheon access to the Linux Foundation helpdesk. They will need to provide your email address.
- If you are part of an agency, please do the following:
    - Create an Agency/Organization account on Pantheon.
    - Ask your Linux Foundation contact to submit an access request to helpdesk. They must include the name of your Pantheon Organization in the access request.

Guidelines

- **Do not** make changes to the parent theme/template, Salient. All changes should be made either in the child theme, or (if it makes sense to do so) implemented outside the theme entirely, in PageBuilder/VisualBuilder.
- **Do not** install new plugins.
- **Do not** change custom posts or any existing features.
- If you think you need to do any of these things, you are responsible for initiating a meeting request with the LF engineer(s) managing Wordpress to work out a solution.

Version 2.0 | Updated June 8, 2018

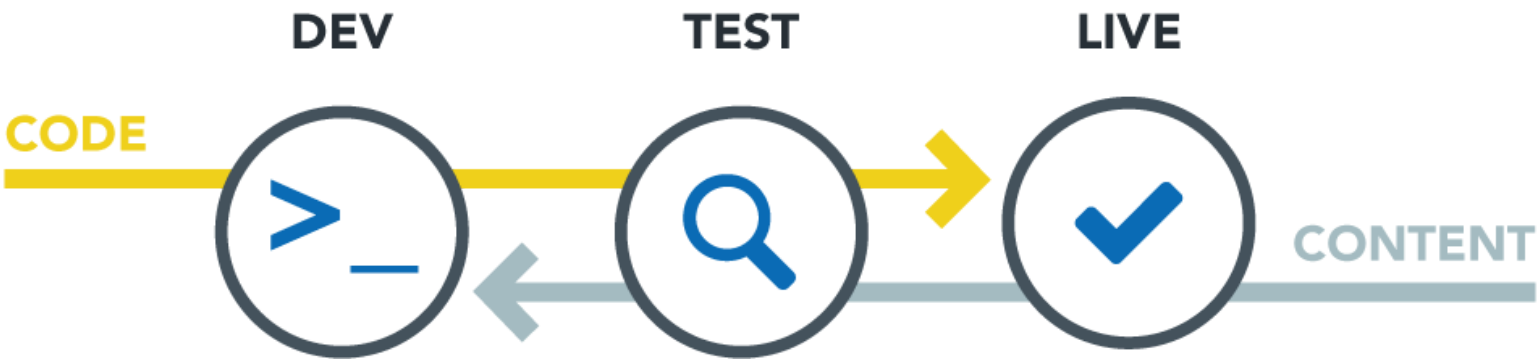
- For example, if Salient will not meet the project needs and a new template needs to be built, LF engineering must be given a minimum of 2 weeks' lead time to review and sign off on the completed new template before it is pushed live.

Workflow and Go-live

- All work should happen in the dev environment, using Pantheon's Multidev feature to contain your work to a separate branch.
- When your work is complete, it must be reviewed by a Linux Foundation Wordpress engineer.
- If there are no issues, the LF engineer will merge your changes to the test environment for stakeholder review.
- Once the stakeholder approves, the LF engineer will merge the changes to the live environment.

Important Notes

- Any content changes (e.g. image uploads, post/page creations or edits; anything stored in the database) made in the dev environments Wordpress dashboard will need to be redone in the live environment.
- **Code moves up, content moves down.**[1] Please plan for this when you approach your project.



[1] https://pantheon.io/docs/pantheon-workflow/
