www.phparch.com

April 2016
VOLUME 15 - ISSUE 4

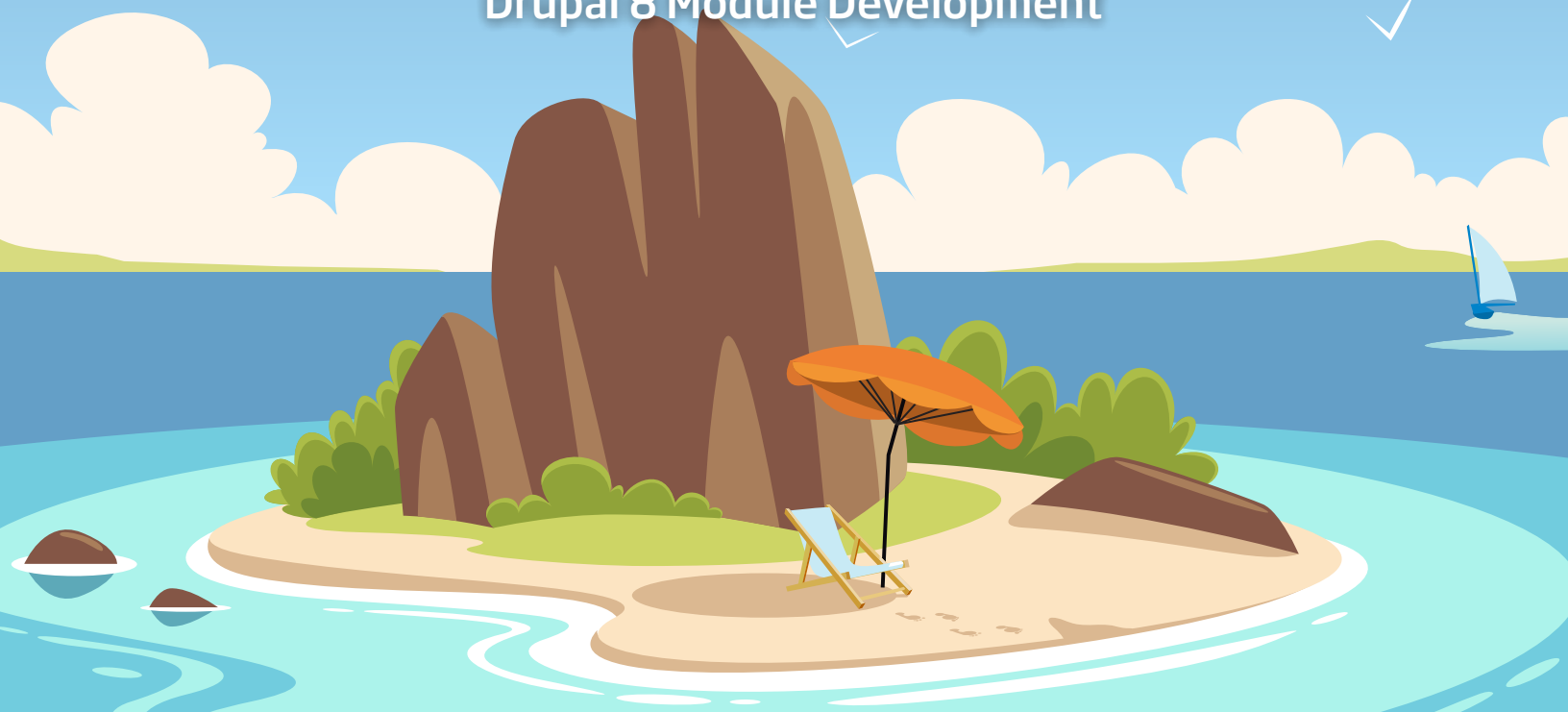
php[architect]

Off the Island: Drupal

Keep Your Drupal Composed

Drupal Security: How Open Source Strengths Manage Software

Drupal 8 Module Development



ALSO INSIDE

Living Documentation:
Generating Documentation from Source Code

Community Corner:
Community Leaders

Education Station:
PhpStorm Intentions for Improved Code Quality

finally{}:
Performance vs Scalability

Security Corner:
Securing Legacy Applications—Part 2

Leveling Up:
Learning a New Framework

BLACKMESH

Your Data Deserves the Highest Degree of Protection.

- Dynamic Scalability in High-Availability & Reliable Environment
- Leveraging OpenStack & OpenShift for Development & Deployment Capabilities
- Unlimited Support Across Entire Stack – From Hardware to Application Levels

FedRAMP

Contact us to learn more about our
FedRAMP Compliant SecureCloud PaaS Solution.

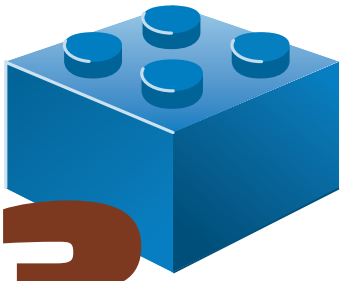
888-473-0854
blackmesh.com

php[architect]

CONTENTS

APRIL 2016
Volume 15 - Issue 4

Off the Island: Drupal



Columns

Subscriptions
Print, digital, and corporate subscriptions are available. Visit https://www.phparch.com/magazine to subscribe or email contact@phparch.com for more information.

Advertising
To learn about advertising and receive the full prospectus, contact us at ads@phparch.com today!

Managing Partners
Kevin Bruce, Oscar Merida, Sandy Smith, Eli White

php[**architect**] is published twelve times a year by:
musketeers.me, LLC
201 Adams Avenue
Alexandria, VA 22301, USA

Contact Information:
General mailbox: contact@phparch.com
Editorial: editors@phparch.com

Print ISSN 1709-7169
Digital ISSN 2375-3544

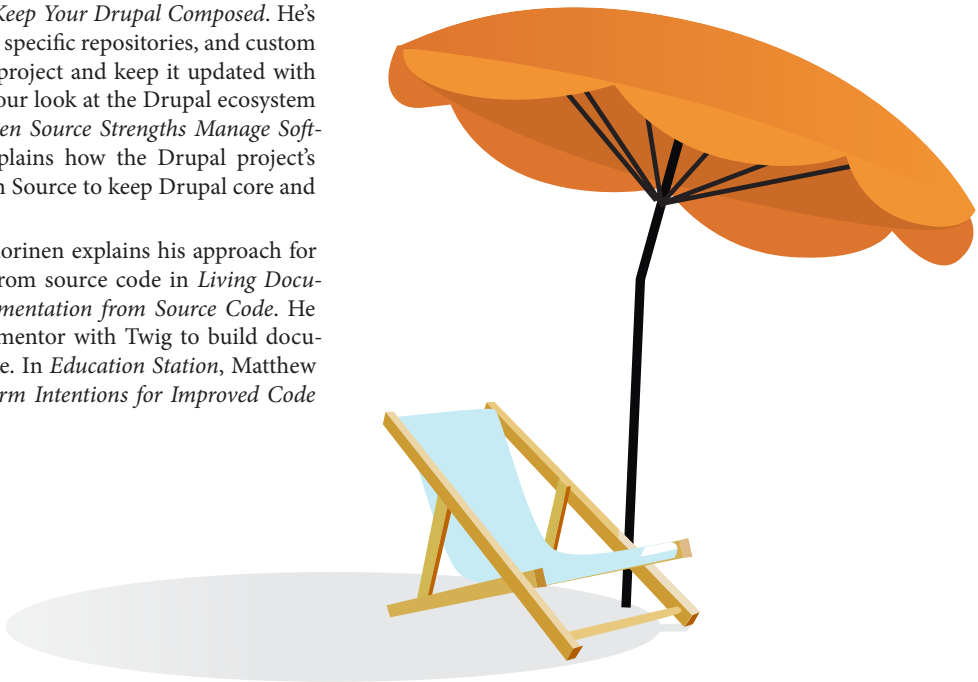
Off the Island: Drupal

In May, most of the php[architect] team will be at Drupalcon in New Orleans, including yours truly. I'm looking forward to it, as it'll be my first actual Drupalcon—even though I've worked with the CMS off-and-on since 2007. I'm particularly excited to see the Drupal and PHP communities building bridges to collaborate ("Getting off the island" as Larry Garfield likes to say) and also learn from each other.

Drupal 8 embraces Symfony, Composer, Guzzle, Twig, and many more components from the PHP ecosystem. This will make it more object-oriented, easier to understand, and overall more familiar to PHP developers. If you haven't evaluated recently, you should give the project another look to see if it might be a good fit for an upcoming project. Conversely, Drupal developers will need to become better versed in using object-oriented principles, understanding design patterns like Dependency Injection, and using Composer. If you find yourself in that position, I urge you to find your local PHP User Group, they can be a great source for help and education.

First off in this issue, Nicola Pignatelli details how the built a module to expire user password in *Drupal 8 Module Development*. He goes through and explains how to set up the various PHP and YAML files to make a working module. Luis Cruz will show you how to *Keep Your Drupal Composed*. He's leveraged Composer, Drupal specific repositories, and custom installers to build a Drupal project and keep it updated with ease. Cathy Theys wraps up our look at the Drupal ecosystem in *Drupal Security: How Open Source Strengths Manage Software Vulnerabilities*. She explains how the Drupal project's security team leverages Open Source to keep Drupal core and contributed modules secure.

Also in this issue, Carl Vuorinen explains his approach for generating documentation from source code in *Living Documentation: Generating Documentation from Source Code*. He shares how to use Phpdocumentor with Twig to build documentation for your code base. In *Education Station*, Matthew Setter looks at using *PhpStorm Intentions for Improved Code Quality*. Intentions can help improve your code quality with just-in-time suggestions for it. David Stockton provides advice on *Learning a New Framework* in *Leveling Up*. If you need to learn a new framework or have inherited an unfamiliar platform, he has some tips to make the learning curve easier to scale. In *Community Corner*, Cal Evans collects Brandon Savage's thoughts about *Forms of Leadership*. You don't have to be vocal or popular to be a leader in the PHP community. In *Security Corner*, Chris Cornutt has more advice on *Securing Legacy Applications*. In this second part, he looks at two more practices to integrate to harden your application. And *finally{}*, Eli White closes the issue by clarifying *Performance vs Scalability*. These terms are often used interchangeably, he'll guide you on using them correctly.



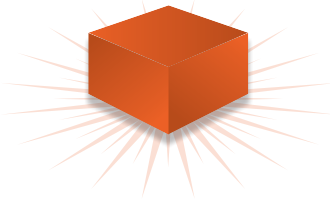
WRITE FOR US

If you would like to contribute, contact us, and one of our editors will be happy to help you hone your idea and turn it into a beautiful article for our magazine.

Visit https://phpa.me/write or contact our editorial team at write@phparch.com and get started!

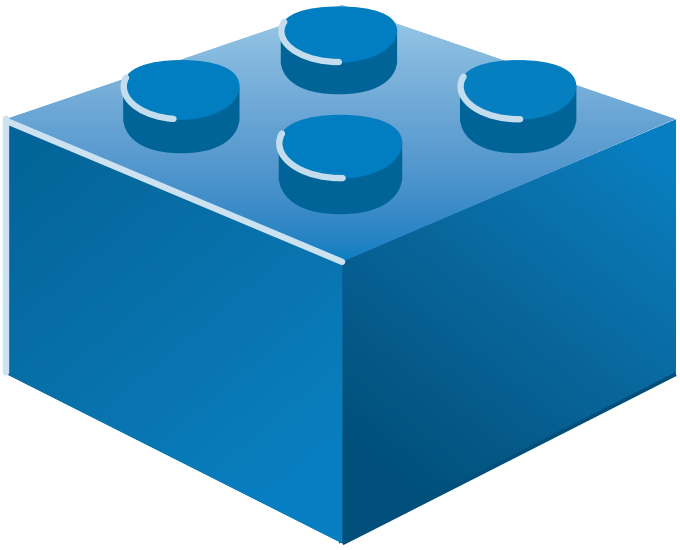
Download this issue's code package:

http://phpa.me/Apr2016_code



Drupal 8 Module Development

Nicola Pignatelli



Drupal is one of the most popular and most used open-source content management systems in the world. It has released a stable version for 8, which can be used on production servers. In this article, I explain how to develop a complete module for version 8.

In this version, the Drupal community reworked the codebase, enhancing it in the points that in the past have been its Achilles' heel. The main features under the microscope are these:

- Mobile ready
- Multilingual capabilities
- Transport configuration changes and managing versions with ease
- Support for standard accessibility technologies, including WAI-ARIA and semantic HTML5
- Built-in web services
- Building sites with Twig template engine
- More field types in core, including entity reference, link, date, email, telephone, etc.
- The page markup in Drupal 8 is now HTML 5-based.
- Non-Drupal developers can embrace object-oriented programming and proven technologies from the larger PHP community.
- Strong integration with Symfony

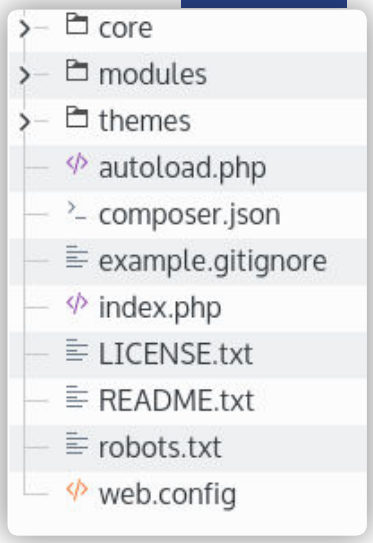
I assume that you have the following knowledge:

- Basic PHP knowledge, including syntax and the concept of PHP objects
- Basic understanding of database tables, fields, records, and SQL statements
- A working Drupal installation 8
- Web server access
- Knowledge of the inner workings of Drupal (hook, handler, etc.)

Drupal 8 installation

To install Drupal, go to the following URL https://www.drupal.org/node/3060/release and download the latest release of version 8. Now unpack the downloaded file with tar (in case you downloaded the .tar.gz file) or unzip (in case you have downloaded the .zip file) and copy all the contents into a subfolder (for example, drupal8demo). If you're using Apache, the subfolder can be under the root folder of web server (for example on Ubuntu in `/var/www/html/drupal8demo`). The file system structure should look like the one in Figure 1.

File system structure **FIGURE 1**



Assuming you have a virtual host configured, open your browser at the following URL http://localhost/drupaldemo8/install.php and follow the instructions. At the end, you will have a working installation of Drupal 8. You may also need to setup a database and db user for your Drupal site.

Creation of expire_user_password Module

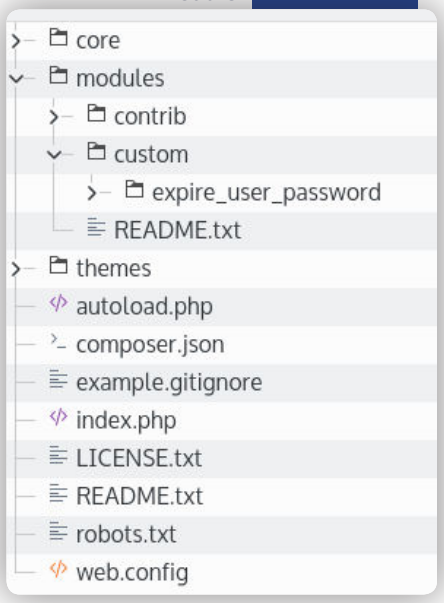
In the following sections, I will create forms that allow us to set the *Time to Live* (TTL) of the user's password for our test portal, configuring it in the various levels of access of the site.

Via the admin pages, we will manage the following features:

- Create a default TTL to apply it for each user when created, if he or she does not have the *Administator* role.
- Possibility to set the TTL when inserting/updating user on the user registration page.
- Possibility to set the TTL for a role.

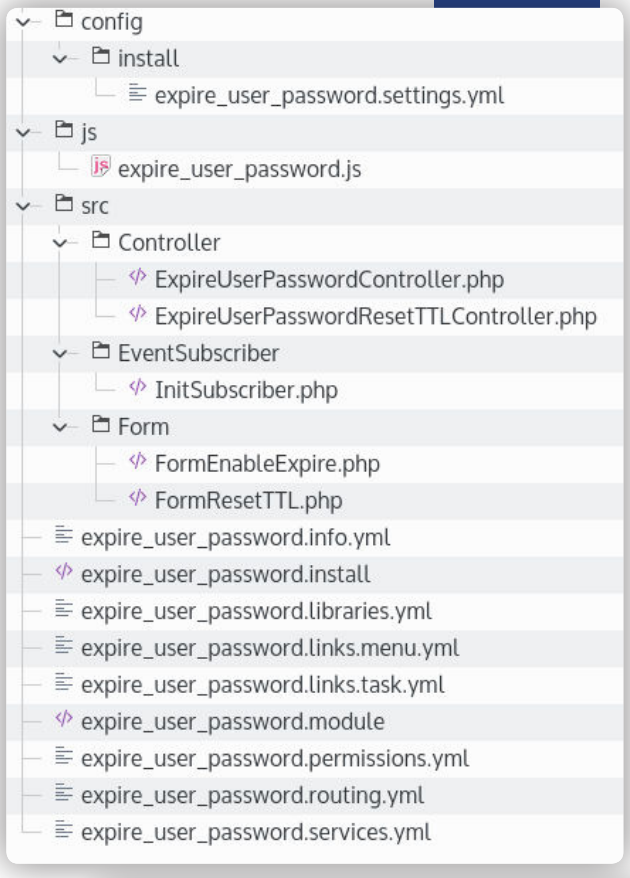
First of all, we need to create a folder under `modules`. Our module will be in a subfolder that is located in `custom/expire_user_password`, which you see in Figure 2.

Folder of custom module **FIGURE 2**



For configuration files, version 8 introduced an innovation module and the system configurations have been moved from the database to YAML (`.yml`) files to increase the portability of modules between environments (development, staging/QA, production) without having to manually repeat step-by-step configurations in order to store them in the database as with Drupal 7.

Folder structure of custom module **FIGURE 3**



In Figure 3, you can see the folder structure of the module and its files.

Module Configuration

This section covers an analysis of the files used to configure the module. In addition to the listing files available for download with the magazine, the code is available to clone via git at https://bitbucket.org/pignaz/expire-user-password and download at https://bitbucket.org/pignaz/expire-user-password/downloads

expire_user_password.info.yml

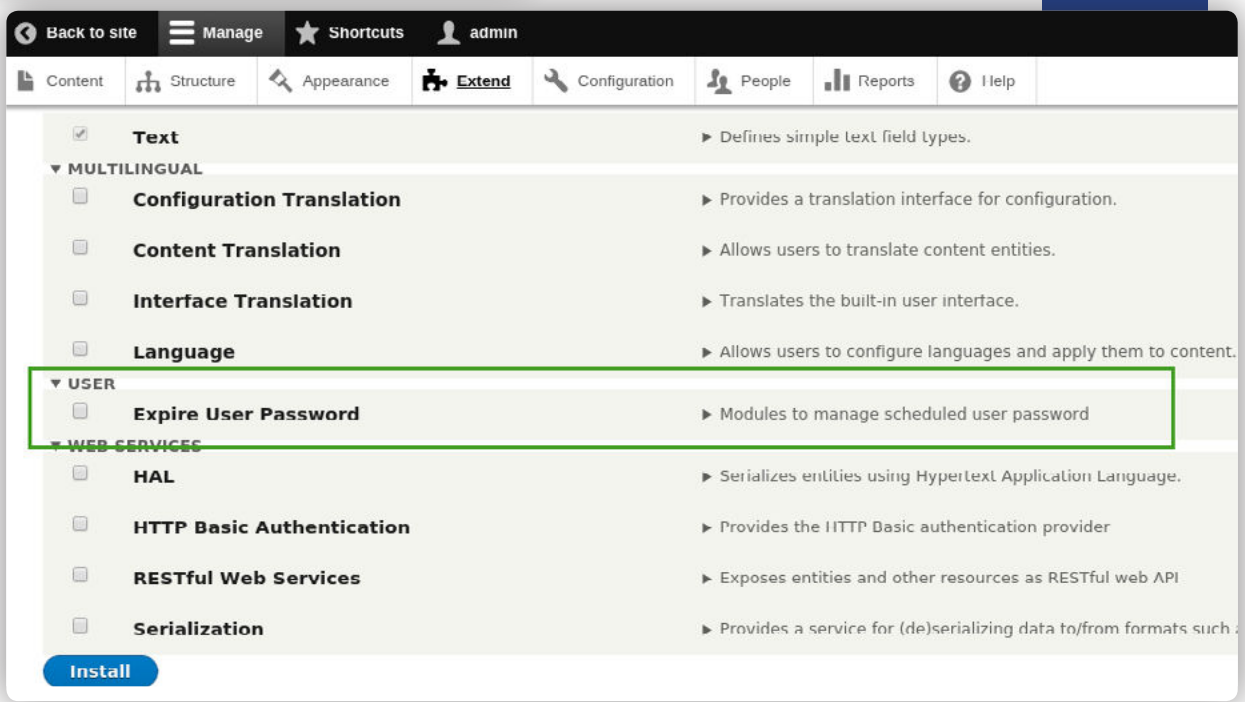
This file matches the `.info` file of the previous Drupal version. It contains information about the module. This will appear in the section `User` of the Module Admin configuration page at `admin/modules`.

The file content is pretty straightforward:

```
name: Expire User Password
description: 'Modules to manage scheduled expiration of
user password'
type: module
package: User
core: 8.x
```

The first line contains the name of the module. The second line contains the description of the module, which explains briefly what the module does. The third line defines the type of package, in this case a module rather than a theme or some other thing. The fourth line adds the module to the *User* section in the list of modules, as in Figure 4. The fifth line says which version of the Drupal core the module was developed for.

Page of activation of modules **FIGURE 4**



LISTING 1

```php
<?php

function expire_user_password_schema() {
   $schema['expire_user_password_users'] = array(
     'description' => t('Register TTL user password by user.'),
        'fields' => array(
           'uid' => array(
           'description' => t('primary key'),
           'type' => 'serial',
           'unsigned' => true,
           'not null' => true,
        ),
        'time_to_live' => array(
           'type' => 'int',
           'not null' => true,
           'default' => 0,
        ),
     ),
     'primary key' => array('uid'),
   );

   return $schema;
}
```

expire_user_password.install

The `.install` file is used to set the module. In particular, I use `hook_schema` to create the `expire_user_password_users` table that is useful to store the user's `uid` and define the expiration date for his or her password, as you can see in Listing 1.

expire_user_password.settings.yml

Here you can find configuration variables and text, which will be imported into the database during module installation, as you can see in Listing 2. Drupal 8 no longer uses the `variable_set` function to store the value of the variable. To retrieve the variable itself, you can see below the code used by this new version of Drupal.

Manage menu by YAML

In Drupal 7 to generate a URL we used `hook_menu` in `[module name].module` file and `hook_permission` to create any permissions to bind in the menu items. This doesn't happen in Drupal 8, where URLs are generated through various files:

- `expire_user_password.routing.yml`
- `expire_user_password.permissions.yml`
- `expire_user_password.links.menu.yml`
- `expire_user_password.links.task.yml`

LISTING 2

```yaml
enabled: 0
global_ttl_default: 15552000

message:
   enable_expire_ok: 'TTL for not admin users were resetted'
   reset_ttl_saved: 'Data saved'

error:
   ttl_user_time_error: 'TTL time entered has expired.'
   ttl_global_time_expired: 'TTL time entered has expired.'
   ttl_global_role_list: 'No role was selected.'
   expired_password_time: 'Your password is expired, reset it.'

label:
   ttl_user: 'TIME TO LIVE USER PASSWORD'
   ttl_user_date_field: 'Set TTL password.Blank NO expire.'
   expire_enabled: 'Enable/disable expire user password TTL.'
   expire_enabled_description: 'Flag to enable users TTL.'
   default_global: 'GLOBAL TIME TO LIVE'
   ttl_global_description: 'Select TTL password. Blank NO expire.'
   ttl_default_roles: 'TIME TO LIVE OF ROLES'
   ttl_role_list: 'time to live to roles.'
   ttl_role_list_description: 'Select TTL role. Blank NO expire.'

email:
   subject: 'Reset user password from !site'
   body: '!username\n\nYour password is expired.\n\n!url\n\n to reset it.'
```

expire_user_password.routing.yml

This file contains an item menu of the Drupal 7 `MENU_CALLBACK` type. The code in `expire_user_password.routing.yml` file is in Listing 3.

From line 1 to line 7, I define the administration page of the module, in which there is a checkbox to enable/disable the password expiration system.

Line 1 defines the routing key, which will be used in other configuration files to call down this page. Line 2 defines the path to go to the administration page. Line 4 defines the controller, which will be used to generate the page layout. Line 5 defines the page title. Lines 7 and 8 define the permission for users who need to access this page. From line 9 to line 15, I define the administration page of the module where we reset the password for selected roles.

Line 9 defines the routing key that will be used in other configuration files to call down this page. Line 10 defines the path to go to the administration page. Line 12 defines the controller, which will be used to generate the page layout. Line 13 defines the path to go to the administration page. Lines 14 and 15 define the permission for users who need to access this page.

LISTING 3

```yaml
expire_user_password.configure:
   path: '/admin/expire_user_password'
   defaults:
      _controller: '\Drupal\expire_user_password\Controller\ExpireUserPasswordController::configure'
      _title: 'Expire user password'
   requirements:
      _permission: 'expire user password configure'

expire_user_password.reset_ttl:
   path: '/admin/expire_user_password_reset_ttl'
   defaults:
      _controller: '\Drupal\expire_user_password\Controller\ExpireUserPasswordResetTTLController::configure'
      _title: 'Reset TTL'
   requirements:
      _permission: 'expire user password configure'
```

expire_user_password.permissions.yml

In Drupal 8, `hook_permission` has been replaced by this specific file, whose content appears in Listing 4.

LISTING 4

```
01. expire user password configure:
02.   title: 'Configure permissions to manage TTL user password'
03.   description: 'This allows to manage TTL user password'
04.   restrict access: true
05.   configure: expire_user_password.configure
```

In `expire_user_password.permissions.yml` I define the permission for pages that manage the expire user password time. Line 1 defines the permission name. Line 2 defines the title, which will appear in the page of permissions at `admin/people/permissions`. Line 3 defines the description that will appear in the permissions page. Line 4 is used to define if access to this resource is restricted or available to any user. Line 5 contains the routing key to the path to go to the administration page to configure the user expire password time. In particular, `expire_user_password.configure` is defined in `expire_user_password.routing.yml`, where it is associated with a real URL.

Permissions are displayed in Figure 5.

expire_user_password.links.menu.yml

This file contains the items menu that matches the old Drupal 7 `MENU_NORMAL_ITEM` type. The code contained in `expire_user_password.links.menu.yml` file appears in Listing 5.

LISTING 5

```
01. expire_user_password.configure:
02.   title: Expire user password
03.   description: 'Configure Expire user password'
04.   parent: system.admin_config_system
05.   route_name: expire_user_password.configure
06.
07. expire_user_password.reset_ttl:
08.   title: Reset TTL
09.   description: 'Reset TTL Default of Expire user password'
10.   parent: expire_user_password.configure
11.   route_name: expire_user_password.reset_ttl
```

From line 1 to line 5, I define the item menu for the configuration administration page. Line 1 contains the routing key, which was defined in `expire_user_password.routing.yml`. Line 2 defines the title of page. Line 3 defines the description of page. Line 4 contains the parent key used to define the position for the item in the menu hierarchy by referring to the parent menu link name. If you don't know the path of the parent menu item, you would need to search for it in all available `*.routing.yml` files and find the route name for that path. Then you would need to search for the route name in all available `*.links.menu.yml` files. The `menu_link` whose `route_name` matches the defined one is your link. Line 5 contains the `route_name` key to tie the menu link to a route.

From line 7 to line 11, we define the item menu for the reset TTL (Time to Live) page. Line 7 contains the routing key, which was defined in `expire_user_password.routing.yml`. Line 8 defines the title of the page. Line 9 defines the page description. Line 10 contains the parent key used to define the item position in the menu hierarchy by referring to the parent menu link name. If you don't know the path of the parent menu item, you would need to search for it in all available `*.routing.yml` files and find the route name for that path. Then you would need to search for the route name in all available `*.links.menu.yml` files. The `menu_link` whose `route_name` matches is your link. Line 11 contains the `route_name` key to tie the menu link to a route.

expire_user_password.links.task.yml

This file contains the item menu that matches the old Drupal 7 `MENU_DEFAULT_LOCAL_TASK` and `MENU_LOCAL_TASK` type. The code contained in `expire_user_password.links.task.yml` file appears in Listing 6.

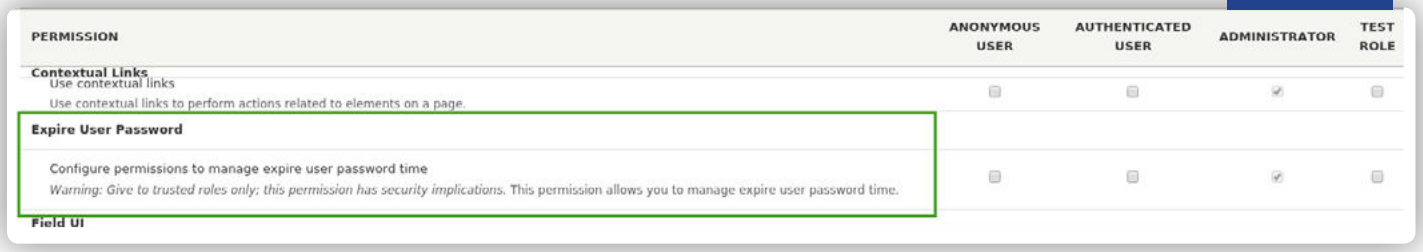
LISTING 6

```
01. expire_user_password.configure:
02.   title: Expire user password
03.   route_name: expire_user_password.configure
04.   base_route: expire_user_password.configure
05.   weight: 40
06.
07. expire_user_password.reset_ttl:
08.   title: Reset TTL
09.   route_name: expire_user_password.reset_ttl
10.   base_route: expire_user_password.configure
11.   weight: 41
```

From line 1 to line 5, I define the item task menu for the configuration administration page. Line 1 contains the routing key defined in `expire_user_password.routing.yml`. Line 2 defines the page title. Line 3 contains the routing key to the path to go to the administration page to configure the user expire password time. Line 4 contains the base routing key, which matches the old Drupal 7 `MENU_DEFAULT_LOCAL_TASK`. Line 5 contains the weight of the item menu in the menu object.

From line 7 to line 11, I define the item menu for the reset TTL (Time to Live) page. Line 7 contains the routing key defined in `expire_user_password.routing.yml`. Line 8 defines the title of page. Line 9 contains the routing key to the path to go to the administration page to configure the TTL user expire password time. Line 10 contains the base routing key, which matches the old Drupal 7 `MENU_DEFAULT_LOCAL_TASK`. Line 11 contains the weight of the item menu in the menu object.

permission page **FIGURE 5**



Add JavaScript and Stylesheet Files

To import the default or custom JavaScript and Stylesheet libraries, you must use the `expire_user_password.libraries.yml` file.

The `drupal_add_js()` and `drupal_add_css()` functions for manual inclusion of JS/CSS files have been removed in favor of library definitions that properly declare their dependencies. As a result of this, all modules and themes have to declare their libraries in order to include JS and/or CSS files in `[name module/name theme].libraries.yml`.

The code contained in `expire_user_password.libraries.yml` file appears in Listing 7.

Line 4 contains

LISTING 7

```
01. global:
02.    version: 1.0
03.    js:
04.       js/expire_user_password.js: {}
05.    dependencies:
06.       - core/jquery
07.       - core/drupal
08.       - core/drupalSettings
```

the custom JavaScript of the module. Between braces, you could insert the conditions of use for this file—for example, weight, browser type, and other.

Customize Actions on User Entity

In this section, I analyze the code that acts mainly on the User entity because we will need to work on some user forms and also trigger periodic actions to control the TTL of user passwords.

LISTING 9

```
01. function expire_user_password_form_alter(&$form,
02.               FormStateInterface $form_state, $form_id) {
03.
04.    switch ($form_id) {
05.
06.       case 'user_form':
07.       case 'user_register_form':
08.          $config = \Drupal::service('config.factory')
09.             ->getEditable('expire_user_password.settings');
10.          $ttl_user = date("Y-m-d", strtotime ("+6 months"));
11.
12.          if ($form_id == 'user_form') {
13.             $request = \Drupal::request();
14.             $path_args = parse_url(\Drupal::request()->getUri());
15.             $path_args = explode("/", substr($path_args['path'], 1));
16.             $ttl_user = _expire_user_password_get_user_ttl($path_args[1]);
17.             if ($ttl_user) {
18.                $ttl_user = date("Y-m-d", $ttl_user);
19.             } else {
20.                $ttl_user = '';
21.             }
22.          }
23.
24.          $account = User::load(\Drupal::currentUser()->id());
25.
26.          if ($config->get('enabled') &&
27.             $account->hasPermission('expire user password configure')) {
28.             $form['ttl_user'] = array(
29.                '#type'        => 'fieldset',
30.                '#title'       => t($config->get('label.ttl_user')),
31.                '#collapsible' => FALSE,
32.                '#collapsed'   => FALSE,
33.             );
34.
35.             $form['ttl_user']['eup_date_field'] = array(
36.                '#title'         => '',
37.                '#type'          => 'date',
38.                '#description'   => t($config->get('label.ttl_user_date_field')),
39.                '#default_value' => $ttl_user,
40.             );
41.          }
42.          $form['actions']['submit']['#submit'][] = '_expire_user_password_ttl_user_submit';
43.       break;
44.    }
45. }
```

LISTING 8

```
01. function expire_user_password_help($route_name, RouteMatchInterface $route_match) {
02.    switch ($route_name) {
03.       case 'help.page.expire_user_password':
04.          $output = '';
05.          $output .= '<h3>' . t('About') . '</h3>';
06.          $output .= '<p>' . t('Allow to set TTL user password.') . '</p>';
07.          $output .= '<h3>' . t('Uses') . '</h3>';
08.          $output .= '<dl>';
09.          $output .= '<dt>' . t('Expire user password') . '</dt>';
10.          $output .= '<dd>' . t('Active module on form') . '</dd>';
11.          $output .= '<dt>' . t('Reset TTL') . '</dt>';
12.          $output .= '<dd>' . t('Reset TTL for roles') . '</dd>';
13.          $output .= '</dl>';
14.          return $output;
15.    }
16. }
17. expire_user_password_form_alter
```

expire_user_password.module

This file contains hooks and callback functions to change the output and the actions on User entities.

expire_user_password_help

The first hook in our `.module` file, `hook_help`, provides online user help. The page-specific help information provided by this hook appears in the Help block (provided by the core Help module) if the block is displayed on that page. The `hook_help` code appears in Listing 8.

expire_user_password_form_alter

`hook_form_alter` performs alterations before a form is rendered. The `hook_form_alter` code appears in Listing 9.

Compared to Drupal 7, here the `$form` parameter is passed as a reference while

`$form_state` does not need it because it already references an object. The third parameter is the `form_id` on which I act.

In this hook, I will alter forms whose IDs are `user_register_form` and `user_form`. I only change the forms under certain conditions.

From line 8 to 22, retrieve the default value for a user's password TTL and write it in the `$ttl_user`. If we are in `user_form`, write in `$ttl_user` the value returned from the `_expire_user_password_get_user_ttl` function, using the ID present in the URL of the page.

From line 26 to line 41, if the user's password TTL is enabled and the user has permission to update the TTL, in the form a datetime field is added and here a new TTL is stored.

In line 42, the callback function is added for the submit to store the new TTL in the `expire_user_password_users` table.

In Figure 6, you see the field added to form.

_expire_user_password_get_user_ttl

This function is used to get the user's password TTL time from the `expire_user_password_users` table. The code appears in Listing 10.

_expire_user_password_ttl_user_submit

This callback is used with `user_form` and `user_register_form`. The code appears in Listing 11.

In line 3, I verify if the `pass` field is present in the form ; if yes, I set the `$timestamp` variable to the actual time plus the time in the `global_ttl_default` configuration parameter. Otherwise I set the `$timestamp` variable as value of the `eup_date_field` form.

Then if `$timestamp` is not equal to zero I insert/update the timestamp for ID user in the `expire_user_password_users` table. If `$timestamp` is equal to zero, I delete the record about this user ID from the `expire_user_password_users` table.

expire_user_password_entity_load

`hook_entity_load` acts on entities when loaded. Here I verify that the entity type is `user`, and if this is the case I add the `ttl_password_expired` property to the entity, entering the value returned by the `_expire_user_password_get_user_ttl` function.

expire_user_password_entity_delete

`hook_entity_delete` is an entity deletion. This hook runs once the entity has been deleted from storage. In Listing 13 I verify that the entity type is User, and in this case I delete the record on the entity user ID from the `expire_user_password_users` table.

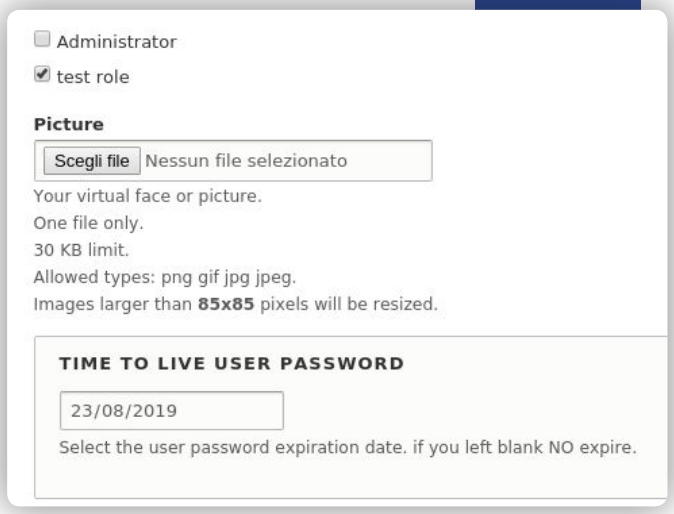
expire_user_password_cron

`hook_cron` performs periodic actions. Modules that require some commands to be executed periodically can implement `hook_cron()`.

This hook is used to alert users that their password is expired and they must change it.

The system extracts users with no administrator role from the database and sends them mail via the `expire_user_password_mail` function, which is called by `mail` in line 19.

FIGURE 6 user form



LISTING 10

```
01. function _expire_user_password_get_user_ttl($user_id=0) {
02.
03.    $eup = db_select('expire_user_password_users', 'eup')
04.       ->fields('eup', array('time_to_live'))
05.       ->condition('uid', $user_id)
06.       ->execute()
07.       ->fetchObject();
08.    if (!is_object($eup)) {
09.       return NULL;
10.    }
11.
12.    return $eup->time_to_live;
13. }
```

LISTING 11

```
01. function _expire_user_password_ttl_user_submit($form, FormStateInterface $form_state) {
02.    $timestamp = 0;
03.    if ($form_state->getValue('pass')) {
04.       $config = \Drupal::service('config.factory')
05.          ->getEditable('expire_user_password.settings');
06.       $timestamp = time() + $config->get('global_ttl_default');
07.    } elseif ($form_state->getValue('eup_date_field')) {
08.       $timestamp = strtotime($form_state->getValue('eup_date_field'));
09.    }
10.
11.    if ($timestamp) {
12.      $mergeObject = new Merge(Database::getConnection(), 'expire_user_password_users');
13.      $mergeObject->key('uid', $form_state->getValue('uid'));
14.      $mergeObject->fields(array('time_to_live'), array($timestamp));
15.      $mergeObject->execute();
16.    } else {
17.      $deleteObject = new Delete(Database::getConnection(), 'expire_user_password_users');
18.      $deleteObject->condition('uid', $form_state->getValue('uid'));
19.      $deleteObject->execute();
20.    }
21. }
```

LISTING 12

```
01. function expire_user_password_entity_load(array $entities, $entity_type_id) {
02.    if ($entity_type_id=='user') {
03.       foreach ($entities as $key => $entity) {
04.          $entity->ttl_password_expired = _expire_user_password_get_user_ttl($entity->id());
05.       }
06.    }
07. }
```

expire_user_password_mail

This function sends email to users to alert them that their password is expired and that they must retype.

expire_user_password_page_attachments_alter

`hook_page_attachments_alter` alters the attachments to a page before it is rendered. If the `alerting_expired` parameter is present in the query string, I store the value of the `alerting_expired` parameter in `drupalSettings`, which is restored on the client side by JavaScript.

To attach the `alerting_expired` parameter to the query string, you must see the Service section illustrated below.

LISTING 13

```
01. function expire_user_password_entity_delete(EntityInterface $entity) {
02.    if ($entity->getEntityTypeId()=='user') {
03.       $deleteObject = new Delete(Database::getConnection(), 'expire_user_password_users');
04.       $deleteObject->condition('uid', $entity->id());
05.       $deleteObject->execute();
06.    }
07. }
```

LISTING 14

```
01. function expire_user_password_cron() {
02.
03.    $config = \Drupal::service('config.factory')
04.              ->getEditable('expire_user_password.settings');
05.
06.    $query = \Drupal::entityQuery('user')
07.              ->condition('status', 1)
08.              ->condition('roles', 'administrator', '<>')
09.              ->condition('uid', 0, '>');
10.    $uids = $query->execute();
11.    $accounts = User::loadMultiple($uids);
12.
13.    foreach ($accounts as $account) {
14.       if ($config->get('enabled') && !$account->isAnonymous() &&
15.          !in_array('administrator', $account->getRoles()) &&
16.          $account->ttl_password_expired &&
17.          time() > $account->ttl_password_expired) {
18.          $params = array('account' => $account);
19.          \Drupal::service('plugin.manager.mail')->mail(
20.                   'expire_user_password', 'expire_password',
21.                   $account->getEmail(),
22.                   $account->getPreferredLangcode(), $params);
23.       }
24.    }
25. }
```

LISTING 15

```
01. function expire_user_password_mail($key, &$message, $params) {
02.    $config_site = \Drupal::service('config.factory')->
                                        getEditable('system.site');
03.    $config = \Drupal::service('config.factory')
04.                 ->getEditable('expire_user_password.settings');
05.
06.    $variables = array(
07.       '!username' => $params['account']->getUsername(),
08.       '!site'     => $config_site->get('name'),
09.       '!url'      => user_pass_reset_url($params['account'])
10.    );
11.
12.    switch ($key) {
13.       case 'expire_password':
14.          $langcode = $message['langcode'];
15.          $message['subject'] = t($config->get('email.subject'),
16.                $variables, array('langcode' => $langcode));
17.          $message['body'][] = t($config->get('email.body'),
18.                $variables, array('langcode' => $langcode));
19.       break;
20.    }
21. }
```

LISTING 16

```
01. function expire_user_password_page_attachments_alter(&$page) {
02.    $path_args = parse_url(\Drupal::request()->getUri());
03.    $path_args = explode("/", substr($path_args['path'], 1));
04.
05.    $page['#attached']['library'][] = 'expire_user_password/global';
06.    if (\Drupal::request()->query->get('warning_expired')) {
07.       $page['#attached']['drupalSettings']['expire_user_password']
08.             ['global']['warning'] = \Drupal::request()->query->get('warning_expired');
09.    }
10. }
```

expire_user_password.js

This code allows us to display an alert JavaScript on web page load if `drupalSettings.expire_user_password` is defined server side, as in Listing 16. This file code appears in Listing 17.

LISTING 17

```
01. (function ($, Drupal, drupalSettings) {
02.
03.    "use strict";
04.
05.    Drupal.behaviors.warning_expire = {
06.       attach: function (context, settings) {
07.          if(drupalSettings.expire_user_password!=undefined) {
08.             alert(drupalSettings.expire_user_password.global.warning);
09.          }
10.       }
11.    };
12.
13. })(jQuery, Drupal, drupalSettings);
```

Build Forms

In previous paragraphs, you have seen how to create a URL to attach a page of our installation.

To generate content in the module configuration page, I will create a `Controller` class that will contain our configuration form. To map a controller to a URL, recall Listing 3.

In line 2, I defined a URL and in line 4 I attached a controller to the configuration page:

```
_controller: '\Drupal\expire_user_password\Controller \
    \ExpireUserPasswordController::configure'
```

In line 9, I defined a URL and in line 12 I attached a controller to reset the TTL configuration page:

```
_controller: '\Drupal\expire_user_password\Controller \
    \ExpireUserPasswordResetTTLController::configure'
```

ExpireUserPasswordController.php

The implementation code for `ExpireUserPasswordController` is found in `src/Controller/ExpireUserPasswordController.php` as shown in Listing 18.

Line 1 defines the namespace to which controller belongs. Line 3 defines the controller class. Line 6 defines the public member function used to render the page. Lines 7 and 8 contain the following code:

```
$form = \Drupal::formBuilder()->getForm(
    'Drupal\expire_user_password\Form\FormEnableExpire'
);
```

The `$form` variable includes the form object for rendering.

FormEnableExpire.php

The code which implements a form to enable/disable the TTL time configuration is included in `src/Form/FormEnableExpire.php` as shown in Listing 19.

Line 2 defines the namespace to which Form belongs. From line 4 to line 9, I define the libraries to use in this class. In the class, the `getFormId` method defines the Form ID that will be rendered in the HTML form page.

In Drupal 8, the function `drupal_get_form` is no longer used to generate a form. It has been replaced by the `buildForm` method that you can see in line 17. This function contains the form definition. In line 18, we get the configuration of the module from the `expire_user_password.settings` key, which matches `expire_user_password.settings.yml`.

```
$config = \Drupal::config('expire_user_password.settings')
```

`$config` has the `get` function member that we use to retrieve a string from settings—for example, to line 21.

```
$form['expire_enabled'] = array(
  '#title'=> $this->t($config->get('label.expire_enabled')),
  '#type' => 'checkbox',
```

`t` is the member function used to translate a string in a selected language.

In Drupal 8, if you want to execute some code on form submission, you must use the `submitForm` member function.

In line 36, I retrieve settings but, compared to line 14, the object that I insert in the the `$config` variable is writable, so I can retrieve strings by using the `get` member function and overwrite strings by using the `set` member function. From line 41 to line 60, I verify if a checkbox is checked and if it is `true`, we insert/update the TTL user expire password time in the `expire_user_password_users` table, but only for non-administrator users.

LISTING 18

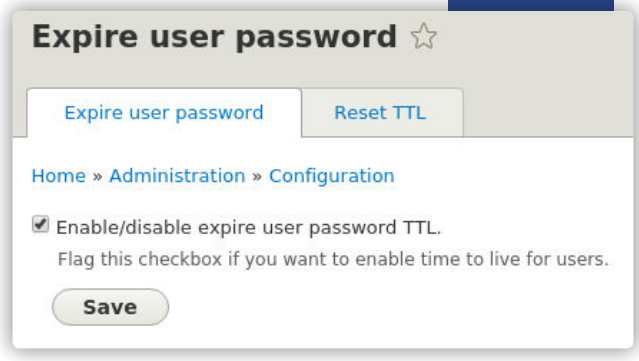
```
01. <?php
02. namespace Drupal\expire_user_password\Controller;
03.
04. class ExpireUserPasswordController
05. {
06.    public function configure() {
07.       $form = \Drupal::formBuilder()
08.          ->getForm('Drupal\expire_user_password\Form\FormEnableExpire');
09.       return $form;
10.    }
11. }
```

LISTING 19

```
01. <?php
02. namespace Drupal\expire_user_password\Form;
03.
04. use Drupal\Core\Form\FormBase;
05. use Drupal\Core\Form\FormStateInterface;
06. use Drupal\Core\Database\Database;
07. use Drupal\Core\Database\Connection;
08. use Drupal\Core\Database\Query\Merge;
09.
10. class FormEnableExpire extends FormBase {
11.
12.    public function getFormId() {
13.       return 'expire_user_password_enable_form';
14.    }
15.
16.    public function buildForm(array $form, FormStateInterface $form_state) {
17.       $config = \Drupal::config('expire_user_password.settings');
18.
19.       $form['expire_enabled'] = array(
20.          '#title' => $this->t($config->get('label.expire_enabled')),
21.          '#type' => 'checkbox',
22.          '#description' => $this->t($config
23.                            ->get('label.expire_enabled_description')),
24.          '#default_value' => $config->get('enabled'),
25.       );
26.       $form['save'] = array(
27.          '#type'   => 'submit',
28.          '#value'  => t('Save'),
29.       );
30.       return $form;
31.
32.    }
33.
34.    public function submitForm(array &$form, FormStateInterface $form_state) {
35.       $config = \Drupal::service('config.factory')
36.                ->getEditable('expire_user_password.settings');
37.       $config->set('enabled', $form_state->getValue('expire_enabled'));
38.       $config->save();
39.
40.       if ($form_state->getValue('expire_enabled')) {
41.          $timestamp = time() + $config->get('global_ttl_default');
42.          $query = \Drupal::entityQuery('user')
43.                   ->condition('status', 1)
44.                   ->condition('uid', 0, '>');
45.          $uids = $query->execute();
46.
47.          foreach ($uids as $uid) {
48.             $account = \Drupal\user\Entity\User::load($uid);
49.             if (!in_array('administrator', $account->getRoles())) {
50.                $mergeObject = new Merge(Database::getConnection(),
51.                               'expire_user_password_users');
52.                $mergeObject->key('uid', $account->id());
53.                $mergeObject->fields(array('time_to_live'), array($timestamp));
54.                $mergeObject->execute();
55.             }
56.          }
57.
58.          drupal_set_message(t($config->get('message.enable_expire_ok')));
59.       }
60.    }
61. }
```

This form is displayed in Figure 7.

Enable Expire Form **FIGURE 7**



ExpireUserPasswordResetTTLController.php

The code that implements `ExpireUserPasswordResetTTLController` is contained in `src/Controller/ExpireUserPasswordResetTTLController.php` in Listing 20.

LISTING 20

```
01. <?php
02. namespace Drupal\expire_user_password\Controller;
03.
04. class ExpireUserPasswordResetTTLController
05. {
06.    public function configure() {
07.       $form = \Drupal::formBuilder()->getForm(
08.             'Drupal\expire_user_password\Form\FormResetTTL');
09.       return $form;
10.    }
11. }
```

Line 2 defines the namespace to which controller belongs. Line 4 defines the controller class. Line 6 defines the public member function that is used to render the page. Lines 7 and 8 contain the next code which retrieves the form we defined earlier:

```
$form = \Drupal::formBuilder()->getForm(
    'Drupal\expire_user_password\Form\FormResetTTL'
);
```

The `$form` variable gets the form object to rendering.

FormResetTTL.php

The code that implements form to reset the TTL time, based on the user's role, appears in `src/Form/FormResetTTL.php` in Listing 21.

Line 2 defines the namespace to which form belongs. From line 4 to line 8, I define the libraries to use in this class. From line 12 to line 14, we define the form ID that will be rendered in the HTML form page.

The `buildForm` member function is in line 16. This function contains the definition of form. In particular, in line 17 we get the configuration of the module from the `expire_user_password.settings` key, which matches `expire_user_password.settings.yml`.

```
$config = \Drupal::
          config('expire_user_password.settings')
```

`$config` has the `get` function member that we use to retrieve a string from settings—for example, in line 23.

```
$form['ttl_default_global'] = array(
   '#type'  => 'fieldset',
   '#title' => $this->t($config
               ->get('label.default_global')),
```

To maintain the form field hierarchy, we must write instructions as in line 19.

```
$form['#tree'] = TRUE;
```

Now if you want to execute some code on form submit, you must use the `submitForm` member function.

Our form contains two fields:

Field at line 31 is used to set the timestamp for the user's password TTL time and, at line 53, to select roles whose users must reset passwords.

From the roles list, we exclude the administrator and authenticated roles (lines 47 and 48).

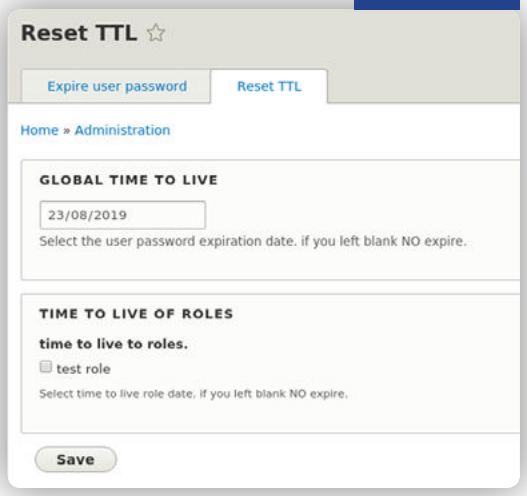
The `validateForm` member function is used to validate form, in particular to verify if the timestamp inserted in the `ttl_global` field is not older than now and to control if at least one role is selected.

The `submitForm` member function is used to select all active users who have roles selected and then insert/update the user's password TTL time in the `expire_user_password_users` table to users selected (lines from 88 to 101).

As the last action, I update the global default TTL value time (line 104).

This form is displayed in Figure 8.

Reset TTL Form **FIGURE 8**



Implement a Service

How can we add our warning parameter to querystring to alert the user that his password has expired?

For this action, I can use a service. First of all, I must define the service I need in `expire_user_password.services.yml`.

expire_user_password.services.yml

The content of `expire_user_password.services.yml` is shown in Listing 22.

LISTING 22

```
12. services:
13.    expire_user_password.init:
14.       class: Drupal\expire_user_password  \
15.              \EventSubscriber\InitSubscriber
16.       tags:
17.          - { name: event_subscriber }
```

In line 2, I define the name of the service: `expire_user_password.init`. It can be any string.

In line 3, I define the class that is able to manage the event service.

In line 4, I have tags. When reading the data in `{MYMODULE}.services.yml` file, one of the things Drupal looks for is service tags. These indicate to the compiler that this particular service should be registered, or used, in a special way. When adding an event subscriber, we tag our service(s) with the aptly named `event_subscriber` tag. When the tag at line 4 is encountered, the compiler instantiates a copy of the tagged service class, calls the `::getSubscribedEvents()`, and retains a list of all their combined responses.

Now I pass to the container class that handles the service that I defined.

LISTING 21

```
01. <?php
02. namespace Drupal\expire_user_password\Form;
03.
04. use Drupal\Core\Form\FormBase;
05. use Drupal\Core\Form\FormStateInterface;
06. use Drupal\Core\Database\Database;
07. use Drupal\Core\Database\Connection;
08. use Drupal\Core\Database\Query\Merge;
09.
10. class FormResetTTL extends FormBase
11. {
12.    public function getFormId() {
13.       return 'expire_user_password_reset_ttl_form';
14.    }
15.
16.    public function buildForm(array $form, FormStateInterface $form_state) {
17.       $config = \Drupal::config('expire_user_password.settings');
18.       $form['#tree'] = TRUE;
19.
20.       $form['ttl_default_global'] = array(
21.          '#type' => 'fieldset',
22.          '#title' => $this->t($config->get('label.default_global')),
23.          '#collapsible' => FALSE,
24.          '#collapsed'   => FALSE,
25.       );
26.
27.       $ttl = $config->get('global_ttl_default') ?
28.                   $config->get('global_ttl_default') : strtotime ("+6 months");
29.       $form['ttl_default_global']['ttl_global'] = array(
30.          '#title'         => '',
31.          '#type'          => 'date',
32.          '#description'   => $this->t($config->get('label.ttl_global_description')),
33.          '#default_value' => date("Y-m-d", $ttl),
34.       );
35.       $form['ttl_default_roles'] = array(
36.          '#type'        => 'fieldset',
37.          '#title'       => $this->t($config->get('label.ttl_default_roles')),
38.          '#collapsible' => FALSE,
39.          '#collapsed'   => FALSE,
40.          '#tree'        => TRUE,
41.       );
42.
43.       $roles    = user_roles(TRUE);
44.
45.       unset($roles['administrator']);
46.       unset($roles['authenticated']);
47.       foreach ($roles as $key => $oRole) {
48.          $roles[$key] = $oRole->get('label');
49.       }
50.
51.       $form['ttl_default_roles']['role_list'] = array(
52.          '#title'         => $this->t($config->get('label.ttl_role_list')),
53.          '#type'          => 'checkboxes',
54.          '#description'   => $this->t($config->get('label.ttl_role_list_description')),
55.          '#options'       => $roles,
56.       );
57.       $form['save'] = array(
58.          '#type'   => 'submit',
59.          '#value'  =>$this->t('Save'),
60.       );
61.       return $form;
62.    }
63.
64.    public function validateForm(array &$form, FormStateInterface $form_state) {
65.       $config = \Drupal::config('expire_user_password.settings');
66.       if ($form_state->getValue('ttl_global') &&
67.            strtotime($form_state->getValue('ttl_global')) < time()) {
68.          $form_state->setError(
69.             $form['ttl_default_global']['ttl_global'],
70.             $this->t($config->get('error.ttl_global_time_expired')));
71.       }
72.
```

continued next page

LISTING 21 (CONT)

```
73.       $roles = $form_state->getValue('ttl_default_roles');
74.       $roles = array_diff($roles['role_list'], array(0));
75.       if (empty($roles)) {
76.          $form_state->setError($form['ttl_default_roles']['role_list'],
77.             $this->t($config->get('error.ttl_global_role_list'))
78.          );
79.       }
80.    }
81.
82.    public function submitForm(array &$form, FormStateInterface $form_state) {
83.       set_time_limit(0);
84.
85.       $values = $form_state->getValues();
86.       $roles = array_diff($values['ttl_default_roles']['role_list'], array(0));
87.       $query = \Drupal::entityQuery('user')
88.                   ->condition('status', 1)
89.                   ->condition('roles', $values['ttl_default_roles']['role_list'], 'IN')
90.                   ->condition('roles', 'administrator', '<>')
91.                   ->condition('uid', 0, '>');
92.       $uids = $query->execute();
93.       foreach ($uids as $uid) {
94.          $mergeObject = new Merge(Database::getConnection(), 'expire_user_password_users');
95.          $mergeObject->key('uid', $uid);
96.          $mergeObject->fields(array('time_to_live'),
97.             array(strtotime($values['ttl_default_global']['ttl_global'])));
98.          $mergeObject->execute();
99.       }
100.
101.       $config = \Drupal::service('config.factory')
102.             ->getEditable('expire_user_password.settings');
103.       $config->set('global_ttl_default',
104.          strtotime($values['ttl_default_global']['ttl_global']
105.       ));
106.       $config->save();
107.
108.       drupal_set_message($this->t($config->get('message.reset_ttl_saved')));
109.    }
110. }
```

InitSubscriber.php

The class contained in this file is the container for the service that I defined. Listing 23 shows the class definition saved as `src/EventSubscriber/InitSubscriber.php`.

Line 2 defines the namespace to which the service belongs. From line 4 to line 11, I define the libraries to use in this class. From line 37 to line 46, I implement the `getSubscribedEvents` member function, which I use to register the `checkExpiredUserPassword` function on the request event. From line 16 to line 37, I implement the `checkExpiredUserPassword` member function, where I get the password TTL time of the user and verify some conditions at lines 24–28:

- If the user is not anonymous
- If the user is not administrator
- If the user's password timestamp is not `null`
- If the user's password timestamp is less than now
- If the URL is not one of these: `/user/[user id]/edit`, `/user/[user id]/password`, `/user/[user id]/reset` where `[user id]` must be changed with ID user

If all previous conditions are `true`, then the Drupal system logs out the user and redirects him to the `/user/password` page with the alerting_expired parameter in the querystring.

Conclusion

In this tutorial I have created a module and showed how to use hooks, create forms, and schedule periodic actions. You saw that many functions and hooks were replaced by YAML configuration files (`hook_permission`, for example) and Object methods (`\Drupal\User::load` for `user_load`).

I think the Drupal community will be divided between fans of these new features and people who would have preferred to remain faithful to the hook events and actions that are typical of earlier Drupal versions. I believe that everything evolves and nothing is immutable, so let's start this new challenge given by the best open-source CMS in the world.

LISTING 23

```
01. <?php
02. namespace Drupal\expire_user_password\EventSubscriber;
03.
04. use Symfony\Component\HttpFoundation\RedirectResponse;
05. use Symfony\Component\HttpKernel\KernelEvents;
06. use Symfony\Component\HttpKernel\Event\FilterResponseEvent;
07. use Symfony\Component\EventDispatcher\EventSubscriberInterface;
08.
09. use Drupal\Component\Utility\Unicode;
10.
11. use Drupal\user\Entity;
12.
13. use Symfony\Component\HttpKernel\Event\GetResponseEvent;
14.
15. class InitSubscriber implements EventSubscriberInterface {
16.
17.    public function checkExpiredUserPassword(GetResponseEvent $event) {
18.       $path_args = parse_url($event->getRequest()->getUri());
19.
20.       $config = \Drupal::service('config.factory')
21.             ->getEditable('expire_user_password.settings');
22.       $account = \Drupal::currentUser();
23.       $ttl_user = _expire_user_password_get_user_ttl($account->id());
24.       if (!$account->isAnonymous() &&
25.          !in_array('administrator', $account->getRoles()) &&
26.          $ttl_user && time() > $ttl_user &&
27.          !preg_match('/^\/user\/' . $account->id() . '\/(edit|password|reset)/',
28.                      $path_args['path'])) {
29.          user_logout();
30.          drupal_set_message(t($config->get('error.expired_password_time')));
31.          $event->setResponse(new RedirectResponse(
32.             '/user/password?warning_expired=' .
33.             t($config->get('error.expired_password_time'))));
34.       }
35.    }
36.
37.    static function getSubscribedEvents() {
38.       $config = \Drupal::service('config.factory')
39.             ->getEditable('expire_user_password.settings');
40.
41.       $events = array();
42.       if ($config->get('enabled')) {
43.          $events[KernelEvents::REQUEST][] = array('checkExpiredUserPassword');
44.       }
45.       return $events;
46.    }
47. }
```

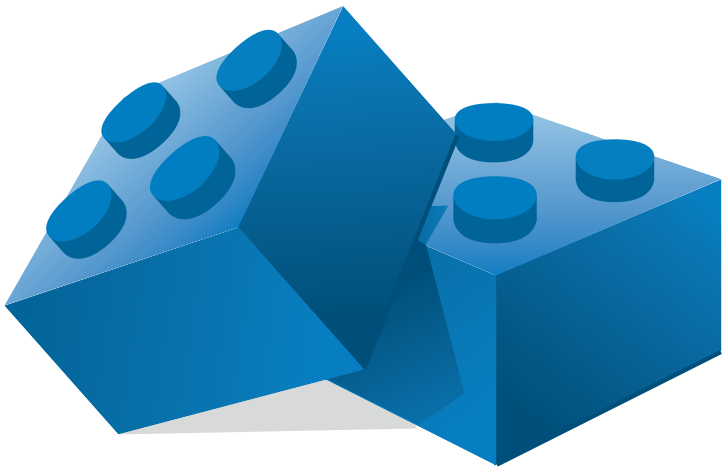
Nicola Pignatelli has been building PHP applications since 2001 for many large organizations in the fields of publishing, mechanics and industrial production, startups, banking and teaching. Currently he is a Senior PHP Developer and Technical Leader at a financial company that provides services for banking products. @pignatellicom

Requirements:

- Drupal 8.0+—http://www.drupal.org

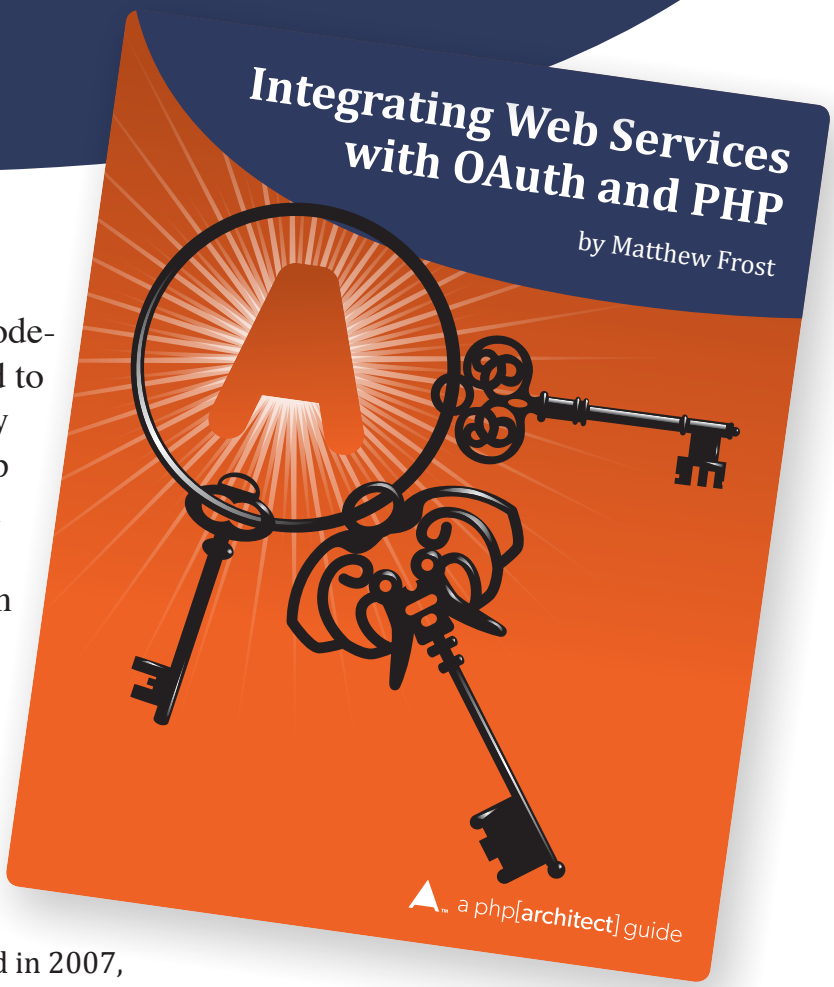
Other Software:

- Apache 2.2+
- MySQL 5.0+
- SMTP module of Drupal 8.0– http://www.drupal.org/project/smtp



Integrating Web Services with OAuth and PHP

by Matthew Frost



Modern web applications are no longer standalone, monolithic codebases. Instead, they are expected to integrate with external, 3rd party applications to allow users to tap into new features, integrate with their social networks, and to easily migrate their data between systems. Many services afford these integrations by building web services that use the OAuth standard to authenticate users and allow "secure delegated access" on their behalf.

There are two versions of OAuth. Version 1.0 as introduced introduced in 2007, and OAuth 2.0 was released in 2012. **Integrating Web Services with OAuth and PHP** describes the differences between the two versions, explains the jargon associated with each, and–most importantly–provides working PHP examples for integrating with popular web services such as Twitter, Tumblr, Instagram, and others. This book also includes a primer on the HTTP protocol, highlights open-source resources for OAuth clients and servers, and discusses issues with OAuth and application security.

Written by PHP professional Matt Frost, this book is an indispensable resource for any PHP developer that builds or integrates with online applications.

Purchase
http://phpa.me/oauthbook

Drupal Security: How Open Source Strengths Manage Software Vulnerabilities

Cathy Theys

It is a frequent topic of discussion whether open-source software (Drupal is under GPL) is not secure because it is open source. Some people worry that if a source is "open," publicly available, and accessible, malicious hackers can find vulnerabilities to exploit. Some think private or closed-source applications would prevent these threats. In this article, I'll review the actions the Drupal project has taken to improve security and handle vulnerabilities.

Any software—whether open- or closed-source—is at risk of cyber threats, just in different ways. However, the collaborative open-source aspect makes software stronger, more defensive, and able to react to any potential issues.

Drupal 8

I have been heavily involved in Drupal Core development for years and have seen progress in making smart defaults in Drupal 8 to make it more secure, and fixes that increase security hardening.

Compared to Drupal 7, Drupal 8 had a significant amount of code refactoring, and included third-party components. In addition to our usual security efforts, the Drupal 8 pre-release Security Bug Bounty program[1] was launched starting June 2015 to crowdsource the discovery of security bugs. Previous Drupal contributors and people new to Drupal participated. Drupal 8 was released in November 2015.

Keeping a Drupal Site Secure

There are many community procedures in place to help Drupal keep pace with security. For site maintainers, the best practical security advice can be found in Greg Knaddison's (greggles) book: *Cracking Drupal*. Another good resource is the handbook pages on Drupal.org[2].

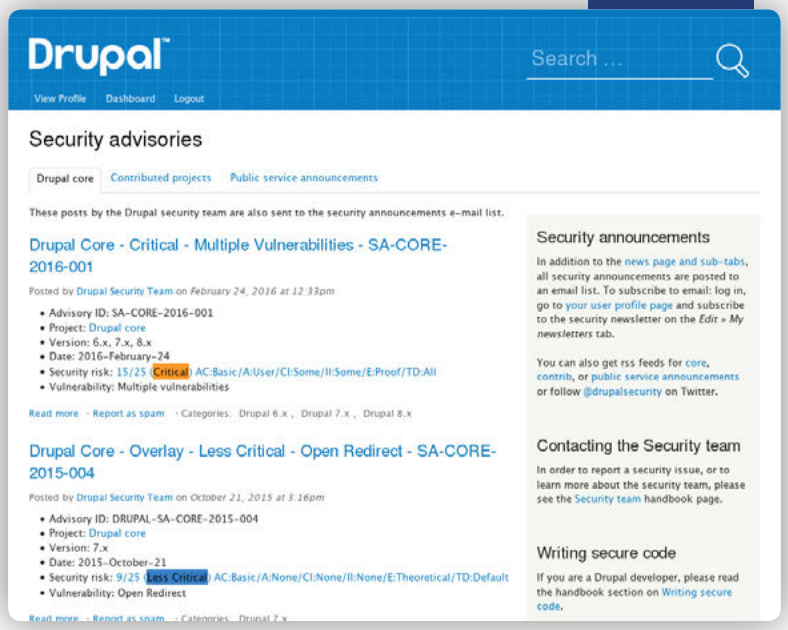
The most important advice is to keep software up to date—both Drupal and your server. *Cracking Drupal* goes into greater depth about common vulnerabilities in custom code, while the book *Drupal Security Best Practices* wisely advises you to write as little custom code as possible. If you do have custom modules or themes, the most common (and very serious) vulnerability is known as Cross-site Scripting (XSS). The most common manifestation of XSS is when user input (such as the title of a piece of content) is printed to the screen without HTML tags being escaped. This could allow a site user to inject JavaScript that would be executed by other visitors of the page. Since JavaScript can cause your browser to take any action you have permission for (create accounts, change settings, etc.), this leads to the site being completely compromised.

Drupal Security Team

The Drupal security team has almost 40 members. I joined the security team as a provisional member July 2015, and became a full member February 2016. The team coordinates reported security issues, makes security advisories, provides assistance for contributed module maintainers in resolving security issues, coordinates with the infrastructure team to keep the drupal.org infrastructure secure, and works to prevent security problems.

To help prevent security problems the security team provides education, including

FIGURE 1



1 *Security Bug Bounty program:* https://www.drupal.org/drupal8-security-bounty

2 *Security Configuration:* https://www.drupal.org/security/secure-configuration

providing documentation on writing secure code[3] and providing documentation on securing sites via the handbook pages mentioned previously.

Drupal projects are made up of *Drupal Core* and also *contributed projects*, referred to as "contrib," that are hosted on Drupal.org. Contributed projects on Drupal.org by first-time contributors are screened for security before the author of the project gets permission to create full projects. The security team facilitates security issues for all full projects with a current 1.0 or greater supported release.

Software Vulnerabilities

All software have bugs, some of which lead to security vulnerabilities. A part of any healthy open-source project is a history of security advisories and fixes. If a project has no advisories, this could indicate security is not getting enough attention. Sometimes the difference between a dangerous security problem and a non-dangerous one is who finds it. If someone finds it and reports it so it can be fixed before it is exploited, it is a better situation. With an open-source project, many people are reporting and fixing things.

Reporting a Drupal Security Issue

The process for reporting a Drupal security issue, potential error, weakness, or threat is to keep it confidential and submit the concern to the Drupal security team[4].

Vulnerabilities are reported from a variety of sources. Sometimes they come from organizations who are performing internal Drupal security audits. Drupal contributors will also report things they notice while working on other issues or tasks for a client. Other open-source projects will sometimes publicly report a vulnerability, and someone will check to see if something similar can happen with Drupal. Other open-source projects will also privately contact the Drupal security team and coordinate security releases when they know Drupal will be affected by something they are also working on.

Sometimes someone might make a public security issue, or a comment on a public issue, if they are not aware of the policy of privately contacting the security team. In those cases, an experienced contributor might notice, unpublish the information, and notify the security team.

Handling Drupal Security Issues

Security issues created (either by going to a project page and using the link "Report a security vulnerability" or by submitting an issue[5] go into a private Drupal security team issue tracker. We gather more info, such as if it effects a current stable release of a project on Drupal.org. People are added to the issue who are not official members of the security team, such as the maintainer, if it is a contributed project. Someone then attempts to reproduce the problem. If it turns out to be an issue that does not need to stay private, a member of the team replies to the reporter and asks them to create a public issue.

Once the issue is verified to be a valid security issue, all the maintainers of the project are also added to the private issue.

The security team and the people added to the issue collaborate to make patches to address the issue. People working on the issue might run tests locally and post test results in the comments on the issue. Once the issue nears consensus, a member of the security team initiates a private full test run on the Drupal CI system and posts the complete test results on the issue.

When consensus is achieved and the test results are good, a release is scheduled, coordinated with contributed project maintainers if it affects contrib projects. And a security team member drafts a Drupal security advisory.

Security Advisory

The Drupal security advisory has an ID, which specifies the project, version, date, and risk level and contains a description of the vulnerability and factors that might mitigate it. An example is https://www.drupal.org/SA-CORE-2015-004, shown in Figure 2.

Part of making the security advisory is using the Drupal Security Risk Calculator. The risk level is calculated using these factors:

- Access complexity: How difficult is it for the attacker to leverage the vulnerability?
- Authentication: What privilege level is required for an exploit to be successful?
- Confidentiality impact: Does this vulnerability cause non-public data to be accessible?
- Integrity impact: Can this exploit allow system data (or data handled by the system) to be compromised?
- Zero-day impact: Does a known exploit exist?
- Target distribution: What percentage of module users is affected?

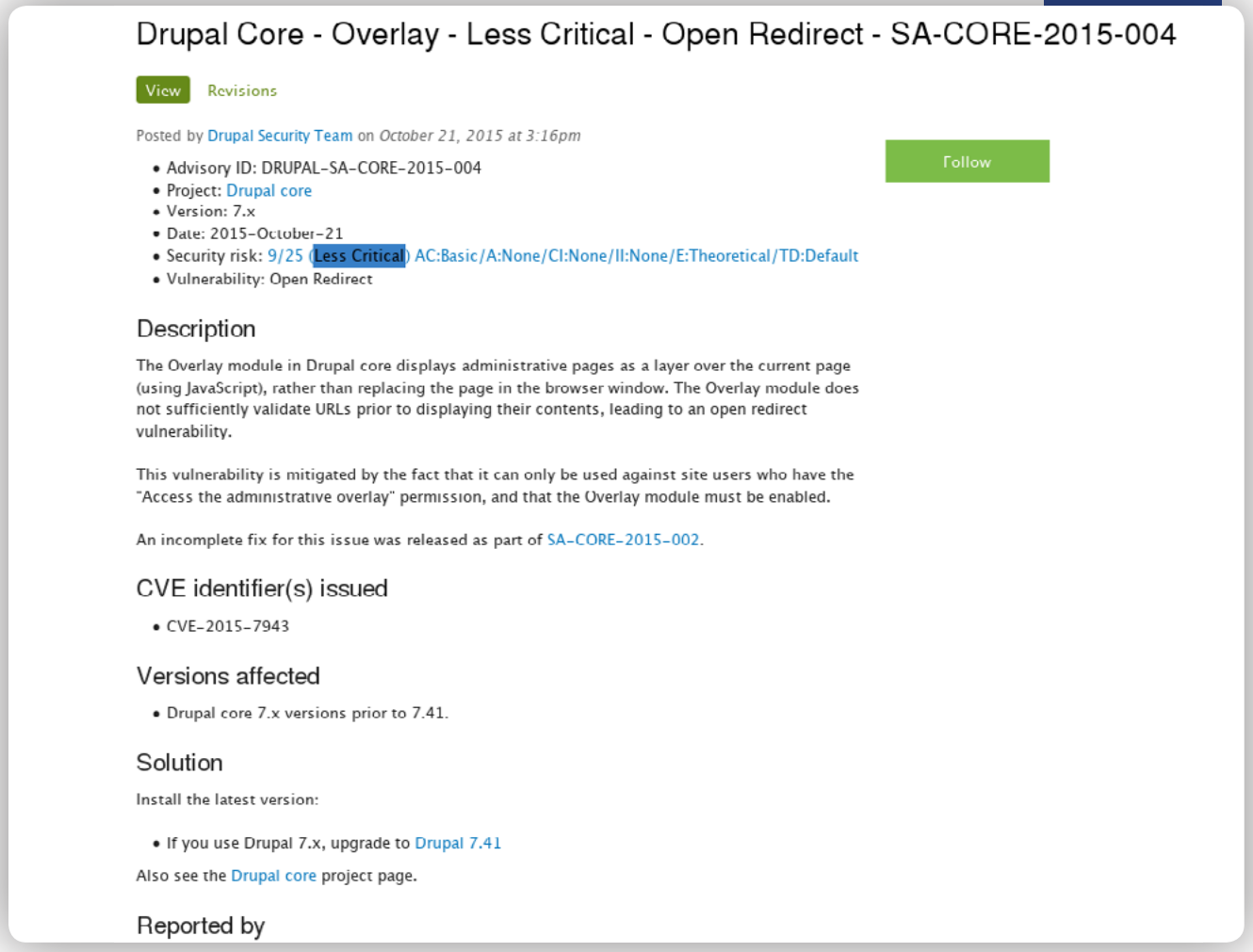
The answers help the team determine if the risk level is *Highly Critical*, *Critical*, *Moderately Critical*, *Less Critical*, or *Not Critical*.

3 *Writing Secure Code:* https://www.drupal.org/writing-secure-code

4 *Reporting Issues:* https://www.drupal.org/security-team/report-issue

5 *Drupal Security, Submit Issue:* https://security.drupal.org/node/add/project-issue

FIGURE 2



The security advisory credits the original reporter and the people who reviewed and worked on the fix. On the day of the release, the fix is committed, and the security advisory is published. After an advisory is published, a CVE (Common Vulnerabilities and Exposures)[6] ID is applied for.

Core security advisories are listed on the Drupal.org security page, https://www.drupal.org/security, and security advisories for contrib projects are listed at https://www.drupal.org/security/contrib.

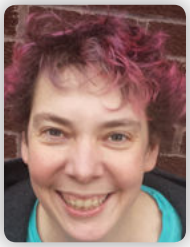
Some issues do not get security advisories. Only problems affecting stable releases get advisories. Advisories are not issued for development releases: dev, alpha, beta, RCs, or sandboxes. If an exploit requires the use of elevated permissions, then there also is no advisory. For example, if a user has to have the "administer users" permission to exploit a vulnerability, there would be no advisory, since someone with advanced permission could already take over a site. The decision to have an advisory or not is made according to the security advisory policy[7].

The Drupal Security Team Welcomes New Members

Members of the security team coordinate with security researchers who deliver reports and project maintainers responsible for fixing security issues. They follow Drupal's internal process for creating security announcements. Members provide advice to project maintainers as they work through security issues, and educate the Drupal community on security topics to improve the overall security stance of the project. Members also use their experience to identify vulnerabilities and make enhancements related to security in Drupal Core and contributed projects. The team is open to new members: https://www.drupal.org/node/1760866.

Open Source

When the source is open, more people can identify issues and privately report them so they can be fixed before they are exploited. We work together, sometimes with other projects, to make sure we handle issues in a responsible way.



Cathy Theys is the Drupal Community Liaison for BlackMesh, a FedRAMP-moderate PaaS certified managed hosting and solutions provider. In her role, Cathy works as a member of the Drupal Security team, contributes to Drupal 8 Core, participates in and presents at Drupal conferences, and organizes the Drupal Mentoring program. You can find Cathy online as @YesCT.

6 *CVE: https://cve.mitre.org*

7 *Security Advisory Policy: https://www.drupal.org/security-advisory-policy*

BLACKMESH

How secure is your Drupal site?

Optimizing your Drupal infrastructure with BlackMesh allows you to easily deploy and manage site content, boost application quality, and eliminate the stress of the underlying infrastructure operations.

With high performance technologies, managed integration and advanced security assessments, BlackMesh is committed to keeping your data protected.

Contact us to schedule a free solution consultation and security assessment.
888-473-0854
blackmesh.com
@blackmesh

Keep Your Drupal Composed

Luis Cruz

Have you ever thought, "Legacy software… Oh, how we wish Composer could manage you…"? I thought that was the case for Drupal 7 until I scratched beyond Composer's surface and found a plethora of tools waiting to prove me wrong. In this article, we will learn how to use custom installers, our own private Packagist repository, and scripts to install Drupal into a project and update it when modules and themes are updated. Keep Drupal Composed? Yeah, we'll be able to do that…

Composer[1] changed the way our web team built applications and allowed us to manage our third-party software dependencies with ease. Except for Drupal[2]… We had to build and maintain a variety of Drupal-based sites; each one had its own code repository with all of the necessary modules and themes checked into it. Each update to the Drupal core or to a module required a tedious update of all the repositories.

As our latest hack-fest was drawing near, I decided to see if Drupal could join the Composer revolution. A week later, I unveiled that the answer was "Yes, yes it can." Using the advanced features of Composer, I created a project that composed in the core Drupal distribution, various Drupal modules and themes, and ran the command-line Drupal site installer with only `composer install`.

Let's dive into how you can keep your Drupal, or other legacy software, composed…

The Final Product

Let's start by looking at a slimmed-down version of the project's `composer.json`, see Listing 1. The `require` attribute starts with an entry to install the Drupal 7.33 core files (`"sprak3000/drupal": "7.33.x-dev"`). It then includes entries to install modules (`"sprak3000-drupal/date": "7.2.8"`) and entries to install themes (`"sprak3000-drupal/rubiktheme": "7.4.0"`). However, these packages are not hosted on Packagist, the default repository for Composer.

Composer knows how to install these requirements from the two entries in the `repositories` attribute. The first defines a version control system repository; this repository will provide the Drupal core files for our project. The second defines a `composer` repository specified by

1 *Composer:* https://getcomposer.org

2 *Drupal:* https://www.drupal.org

LISTING 1

```
01. {
02.   "name": "sprak3000/drupal-composed",
03.   "type": "project",
04.   "homepage": "https://github.com/sprak3000/drupal-composed",
05.   "license": "MIT",
06.   "repositories": [
07.     {
08.       "type": "vcs",
09.       "url": "https://github.com/sprak3000/drupal"
10.     },
11.     {
12.       "type": "composer",
13.       "packagist": "false",
14.       "url": "https://raw.githubusercontent.com/sprak3000 \
15.               /drupal-packagist/master/packages-7.x.json"
16.     }
17.   ],
18.   "require": {
19.     "sprak3000/drupal": "7.33.x-dev",
20.     "sprak3000-drupal/date": "7.2.8",
21.     "sprak3000-drupal/entity_view_mode": "7.1.0-rc1",
22.     "sprak3000-drupal/filefield_paths": "7.1.0-beta4",
23.     "sprak3000-drupal/media": "7.1.4",
24.     "sprak3000-drupal/rubiktheme": "7.4.0",
25.     "sprak3000-drupal/taotheme": "7.3.1",
26.     "sprak3000-drupal/bootstraptheme": "7.3.0"
27.   },
28.   "extra": {
29.     "installer-paths": {
30.       "public/sites/all/modules/{$name}/": ["type:drupal-module"],
31.       "public/sites/all/themes/{$name}/": ["type:drupal-theme"]
32.     }
33.   },
34.   "scripts" : {
35.     "post-install-cmd" : [
36.       "drush --yes --root=$DRUPAL_ROOT
37.         \ site-install linguo --db-url=$DB_URL_DRUPAL
38.         \ | grep -v 'sendmail: not found|
39.         \ Error sending e-mail|Unable to send e-mail'",
40.       "echo 'Setting file upload directory
41.         \ permissions ...';
42.         \ chmod -R ug+w public/sites/default/files"
43.     ],
44.     "post-update-cmd" : [
45.       "drush --yes --root=$DRUPAL_ROOT updatedb",
46.       "drush --yes --root=$DRUPAL_ROOT cache-clear all"
47.     ]
48.   }
49. }
```

a JSON file at a particular URL. Specifying `"packagist": "false"` for this repository tells Composer to disable the default behavior of checking the main Packagist repository.

We override the default paths for installing Drupal modules and themes through the `installer-paths` entry in the `extra` attribute. If a package in the `require` attribute is of the type `drupal-module` or `drupal-theme`, they will be installed into the appropriate directories.

Finally, the `scripts` attribute defines post-install scripts to install Drupal from the command line and post update scripts to run module and theme migrations after they have been updated to new versions. At first glance, this is a fairly standard file, but behind lies some powerful features provided by Composer.

Using Scripts and Events to Install and Upgrade Drupal

Composer allows you to run scripts at various points during its operation. For our project, we want to install Drupal after `composer install` has installed the Drupal files into our project. We also want to run any module or theme upgrades after `compose update` has installed the latest versions of our dependencies. The `post-install-cmd` and `post-update-cmd` events fit the bill; however, Drupal's installation instructions point you to an installation wizard in your web browser. We need something to run from the command line.

Enter Drush[3], the command-line utility for running a wide variety of Drupal administration functions. Let's look at our post-install scripts:

```
"post-install-cmd" : [
    "drush --yes --root=$DRUPAL_ROOT
       \ site-install linguo --db-url=$DB_URL_DRUPAL
       \ | grep -v 'sendmail: not found|
       \ Error sending e-mail|Unable to send e-mail'",
    "echo 'Setting file upload directory permissions ...';
       \ chmod -R ug+w public/sites/default/files"
]
```

We first run the Drush command `site-install` providing the installation profile `linguo`; installation profiles allow you to define what modules and themes your installation of Drupal depends on along with any custom work that needs to be performed during installation. The `--yes` flag will auto respond to any command-line prompts during the process. The `--root` flag points to the root directory for the Drupal files, and the `--db-url` flag points to the database that will host the Drupal installation. The remainder of the scripts filter out messages we don't care to see in the output and ensures that the proper permissions are set on the directory that will hold any files uploaded through the Drupal CMS.

With these scripts in place, our command to install Drupal only requires two environment variables be defined for installation:

```
$ DRUPAL_ROOT='/path/to/your/site/public/'
   \ DB_URL_DRUPAL='mysql://user:pass@server/database'
   \ composer install
```

When you upgrade a module or theme in Drupal, they could require one or more migrations to upgrade your installation to the latest version. To do this automatically, we define these post-update scripts:

```
"post-update-cmd" : [
    "drush --yes --root=$DRUPAL_ROOT updatedb",
    "drush --yes --root=$DRUPAL_ROOT cache-clear all"
]
```

The first script has Drush run the `updatedb` command to tell all the updated modules and themes to run their migrations. The second script clears all the Drupal caches to ensure that any content that depends on the upgraded modules will be refreshed and take advantage of the upgrades. We only need one environment variable setup to run our update command:

```
$ DRUPAL_ROOT='/path/to/your/site/public/' composer update
```

NOTE: *As of Drupal 7, there is no enforced order for running migrations. In practice, our team came across instances where running one module's migration first caused other module migrations to fail. We made it a habit to update modules one at a time or in known "good batches" to avoid this.*

But Where Does Composer Get the Files?

Composer manages dependencies by installing packages from repositories into your project directory. A package is a directory with "something" in it, typically source files. A repository tells Composer what packages it can provide, and their versions. By default, Composer uses Packagist[4] as its repository. When you specify `compose install vendor/package`, Composer queries Packagist to find a match for `vendor/package`.

In my Drupal project's `composer.json`, you will find entries to install Drupal core, modules, and themes (e.g., `sprak3000/drupal`, `sprak3000-drupal/date`, etc.). If you were to search Packagist, it would not list them as packages it knows about. Packages listed there are typically GitHub repositories; Composer would pull these down via `git clone` into our project's directory.

However, the Drupal core distribution, modules, and themes exist as zip files on a server, one file per version. We need a way to download the appropriate zip file and extract it into a specific directory in our project. To solve this, Composer allows us to pull packages from other version control systems along with using custom installers to ensure the files are fetched and placed into the project appropriately.

Solving the Core Dependency with a Custom Installer

Before we can install any modules or themes, we need the core Drupal distribution files. Recall that the project's `composer.json` contained two entries in the `repositories` attribute. The first was this:

```
{
    "type": "vcs",
    "url": "https://github.com/sprak3000/drupal"
}
```

We use this repository by including the sole package available in it:

```
"require": {
    "sprak3000/drupal": "7.33.x-dev"
}
```

3 Drush: http://drupal.org/project/drush

4 Packagist: https://www.packagist.org

The `vcs` repository type tells Composer to pull the contents of the `url` down as a package. If you look at that repository, the only file of note is its own `composer.json`, see Listing 2. Composer will download this file, parse it, and install any dependencies it contains. The only dependency this file specifies outside of the PHP version is `mouf/archive-installer`, a package containing a custom installer called the Archive Installer[5].

LISTING 2

```
01. {
02.   "name": "sprak3000/drupal",
03.   "homepage": "http://tinyurl.com/j3ngr94",
04.   "type": "archive-package",
05.   "license": "MIT",
06.   "authors": [
07.     {
08.       "name": "David N·grier",
09.       "homepage": "http://mouf-php.com"
10.     }
11.   ],
12.   "require": {
13.     "php": ">=5.3.0",
14.     "mouf/archive-installer": "~1.0"
15.   },
16.   "extra": {
17.     "url": "http://ftp.drupal.org/files/projects/drupal-7.33.zip",
18.     "target-dir": "public",
19.     "omit-first-directory": "true"
20.   }
21. }
```

When Composer installs this dependency, the Archive Installer is registered as able to install packages of type `archive-package`. Composer then notices that the package was given the type `archive-package` and invokes the Archive Installer to handle the installation. The Archive Installer expects the following information in the `extra` attribute of the package:

1. `url`: The URL of the archive (`zip`, `tar`, etc.) to be downloaded and extracted into the project
2. `target-dir`: The directory, relative to the project's `composer.json`, where it will be installed. In our project, we specified it to be under `./public`
3. `omit-first-directory`: (Optional, default `false`) Most archives typically bundle the files into a single top-level directory. Setting this to `true` will install the files without this directory. For our project, if the archive contains `./drupal/install*`, our project directory structure will be `./public/install*` instead of `./public/drupal/install*`.

This core Drupal repository has a branch for each version of Drupal required by our various Drupal sites. We reference the branch in our package requirement via the appropriate branch alias (`7.33.x-dev`, `7.31.x-dev`). We went with branch aliases rather than semantic versioning to keep more in line with how Drupal kept its version numbers at the time.

A Brief Anatomy of a Custom Installer

Building a custom installer could be an article in and of itself. Briefly, here is how you create one.

You start by creating a class that either implements or extends a class that implements `Composer\Installer\InstallerInterface`. This interface requires the following methods be defined:

1. `supports($packageType)`: Returns a `boolean` indicating if this installer can handle the given package type.
2. `isInstalled(InstalledRepositoryInterface $repo, PackageInterface $package)`: Returns a `boolean` indicating if the given package from the given repository is already installed in the project.
3. `install(InstalledRepositoryInterface $repo, PackageInterface $package)`: Handles installing the package when `composer install` is invoked.
4. `update(InstalledRepositoryInterface $repo, PackageInterface $initial, PackageInterface $target)`: Handles updating the package when `composer update` is invoked.
5. `uninstall(InstalledRepositoryInterface $repo, PackageInterface $package)`: Handles any steps you wish to take when `composer remove` is invoked for the package.
6. `getInstallPath(PackageInterface $package)`: Returns the directory, relative to the project's `composer.json`, where the package will be installed.

If you are not extending an existing class that implements the interface, you will need to bundle the installer into a Composer plugin package. This is identical to any other Composer package; it contains a `composer.json` but has the following requirements:

1. The `type` attribute must be `composer-plugin`.
2. The `extra` attribute must contain a `class` element that points to the namespaced class name of the plugin.
3. It must require the special `composer-plugin-api` package and define what Plugin API versions the plugin is compatible with.

The plugin class must implement `Composer\Plugin\PluginInterface` by defining `activate(Composer $composer, IOInterface $io)`. This is

5 Archive Installer: http://phpa.me/archive-installer

LISTING 3

```
{
  "packages": {
    "sprak3000-drupal/i18n": {
      "7.1.11": {
        "name": "sprak3000-drupal/i18n",
        "version": "7.1.11",
        "version_normalized": "7.1.11.0",
        "dist": {
          "type": "tar",
          "url": "http://ftp.drupal.org/files/projects/i18n-7.x-1.11.tar.gz",
          "reference": "b7949e640740e34be4be15a22281c5cf",
          "shasum": null
        },
        "require": {
          "composer/installers": "*"
        },
        "type": "drupal-module"
      }
    },
    "sprak3000-drupal/uuid": {
      "7.1.0-alpha5": {
        "name": "sprak3000-drupal/uuid",
        "version": "7.1.0-alpha5",
        "version_normalized": "7.1.0.0-alpha5",
        "dist": {
          "type": "tar",
          "url": "http://ftp.drupal.org/files/projects/uuid-7.x-1.0-alpha5.tar.gz",
          "reference": "91c830af4e0cc9f1af96045d35abe2f8",
          "shasum": null
        },
        "require": {
          "composer/installers": "*"
        },
        "type": "drupal-module"
      },

      "7.1.0-alpha6": {
        "name": "sprak3000-drupal/uuid",
        "version": "7.1.0-alpha6",
        "version_normalized": "7.1.0.0-alpha6",
        "dist": {
          "type": "tar",
          "url": "http://ftp.drupal.org/files/projects/uuid-7.x-1.0-alpha6.tar.gz",
          "reference": "629083e006338bede48093d2ed38a34f",
          "shasum": null
        },
        "require": {
          "composer/installers": "*"
        },
        "type": "drupal-module"
      }

    },
    "sprak3000-drupal/bootstraptheme": {
      "7.3.0": {
        "name": "sprak3000-drupal/bootstraptheme",
        "version": "7.3.0",
        "version_normalized": "7.3.0.0",
        "dist": {
          "type": "tar",
          "url": "http://ftp.drupal.org/files/projects/bootstrap-7.x-3.0.tar.gz",
          "reference": "633ba721e7d0f52627bc665d27cea002",
          "shasum": null
        },
        "require": {
          "composer/installers": "*"
        },
        "type": "drupal-theme"
      }
    }
  }
}
```

typically boilerplate code to register the installers in your plugin:

```
public function activate(Composer $composer, IOInterface $io)
{
    $installer = new MyCustomerInstaller($io, $composer);
    $composer->getInstallationManager()
             ->addInstaller($installer);
}
```

Private Packagist Reporting for Duty!

With the core Drupal distribution installed, we can now focus our attention on how to get modules and themes installed into our project. Modules and themes typically reside under the `sites/all/modules` and `sites/all/themes` directories within the core Drupal distribution directory. Now, where is our project going to find the modules and themes we want?

Here is the second repository in our project's `composer.json`:

```
{
  "type": "composer",
  "packagist": "false",
  "url": "https://raw.githubusercontent.com/sprak3000 \
/drupal-packagist/master/packages-7.x.json"
}
```

A `composer` repository uses a single `packages.json` file to define all the packages and their versions the repository knows about. The `url` element is the URL, minus `packages.json`, for the repository file. Where we currently host only a repository for Drupal 7 items and might later support future Drupal versions, we want our package file to read `packages-7.x.json` (`packages-8.x.json`, `packages-9.x.json`) to reflect this. By setting the `packagist` element to `false`, Composer expects the `url` element to contain the complete path to the `packages.json` file—in our case, a file in another GitHub repository, as in Listing 3.

The attributes of the file are the packages available to use in the `require` section of `composer.json`. The elements in each attribute are the versions of the package made available. For each version, we then define the following data:

1. `name`: This is a duplicate of the attribute, the `vendor/package` given for this particular package.
2. `version`: This is the version number of the package and should match the element it is inside.
3. `version_normalized`: This is a normalized version number; you typically append another `.0` to the version number listed in `version`.
4. `dist`: The information on how to fetch the package contents.
5. `require`: Any dependencies the package requires.
6. `type`: What type of package it is.

If you look at Listing 3, you might notice a familiar pattern. For each module and theme package, we give it the type `drupal-module` or `drupal-theme` and require the `composer/installers` package. The `composer/installers` package provides us with installers that know how to handle `drupal-module` and `drupal-theme` packages.

The custom installers uses the `dist` element to fetch the package. For our modules and themes, they are compressed `tar` archives available via the URL in the `url` element. The `reference` element is the MD5 hash of the archive file. By default, the custom installers will place the modules and themes into the standard `sites/all/modules` and `sites/all/themes` directories relative to our project's `composer.json`. To have them placed in the `public`

directory, our project's `composer.json` provides overrides in the `extra` attribute:

```
"extra": {
    "installer-paths": {
        "public/sites/all/modules/{$name}/":
            ["type:drupal-module"],
        "public/sites/all/themes/{$name}/":
            ["type:drupal-theme"]
    }
}
```

NOTE: *The download pages for Drupal modules and themes do not typically contain the MD5 hash for the files. You will need to download them and generate the hash for the* `reference` *element.*

Drupal 8, Building a Better Composer Tomorrow... Today!

While our project was geared toward Drupal 7, Drupal 8 has been making significant strides to embrace Composer. The core distribution provides two packages—`drupal/core` for the core and its dependencies and `drupal/drupal` for installing a complete distribution for building an application (what would be bundled in the zip file you typically download). Likewise, a Drupal-focused Packagist site[6] allows you to discover and include Drupal 8 (and 7!) modules and themes via Composer. The blog post *Building a Drupal 8 Website with Composer*[7] is a great starting point for building Drupal 8 applications with Composer support. Along with hosting the Drupal Packagist site, the Drupal Composer group[8] also lists numerous discussions and projects related to using Composer with Drupal.

Conclusion

We can keep our Drupal Composed[9], but we can also use these same tools to keep other legacy applications and libraries Composed. Have to pull files from some little-known version control system? Write a custom installer to handle it. Want to manage versions on a project that versions by using file names? You can setup your own custom Packagist repository to manage it. Scripts can help fill in the gaps by running migrations, command-line installers, and more. Add these tools to your Composer belt and a `composer install` of that legacy code can be yours.

Luis Cruz is a senior engineer at Traackr, the leading platform for influencer marketing. He has been working with PHP since 2000 and programming long enough to see his first computer behind glass at the Smithsonian. Since 2004, his passion has been the web and everything around it. His second job is entertaining his daughter and making sure she can identify super heroes from a great distance. @sprak

Related URLs:

- Composer: Setting up and using custom installers - https://getcomposer.org/doc/articles/custom-installers.md
- A Multi-Framework Composer Library Installer - https://github.com/composer/installers

6 *Drupal Composer Packages: https://packagist.drupal-composer.org*

7 *Building a Drupal 8 Website with Composer: http://www.vmdoh.com/node/373*

8 *Drupal Composer Group: http://drupal-composer.org*

9 *Drupal Composed: Linguo: https://github.com/sprak3000/drupal-composed*

Living Documentation: Generating Documentation from Source Code

Carl Vuorinen

Writing documentation can be boring, and even good documentation falls out of date easily over time. This article will demonstrate an approach to generating documentation from source code, so that it is easier to write and keep up to date.

The problem with writing documentation is that it's considered boring by most developers. Even when good documentation is written for a project up front, it can quickly fall out of sync with actual development and become out of date, and even misleading. Most bigger open-source projects have pretty decent documentation, because they have people focusing mostly on the documentation or have set up processes and guidelines for how the documentation should be kept up to date. Smaller projects, be it open-source or client work, don't usually have such focus on documentation, and writing the documentation relies on the motivation of a few developers who would probably rather be writing code.

I <3 writing documentation!

– No one, ever

This article will focus on some strategies that can be used with a small library, or other similar projects, to automatically generate usage documentation from source code. The aim of this kind of approach is to be able to keep the documentation up to date easily when the code changes. This specific approach works best for rather small libraries that have a simple public API. The concepts covered in this article can also be extended to cover different types of documentation for different kinds of projects, or used alongside other manually written documentation.

Code as Documentation

Using source code as a form of documentation is nothing new, and most of you are probably familiar with the *DocBlock* type of comments where you define a method's parameter and return types. Some developers also consider tests as documentation, and others think that code should be self-documenting. All of these approaches have some value, but in this article we are going to focus on the DocBlock style of documentation, as it is easy to parse programmatically.

When your code is your documentation, it's so much easier to keep it in sync when introducing changes or new features. Modern IDEs will even notify you if you have undocumented or mistyped parameters or return types. And when you keep your methods small enough (which you should), you can see the whole method and the DocBlock at the same time on screen, so it will literally be right in front of your eyes.

And if you work on a project where other people look at your code, be it formal code review or just reviewing pull requests in GitHub, it's much more likely that someone will notice if you missed updating the documentation and inform you or make a correction themselves. The closer the documentation is to the code, the more likely it will stay relevant.

Introduction to phpDocumentor

phpDocumentor[1] is a tool used to generate API documentation from DocBlock comments. phpDocumentor has templates which can be used to customize the output. The templates in phpDocumentor are actually a lot more than just templates in the traditional sense, as they can be used to completely modify how the output is generated. A phpDocumentor template defines one or more transformations, which can copy resource files, generate HTML files using XSL or Twig[2], and generate SVG graphs.

Because the templates in phpDocumentor are so powerful that they can be used to control the whole transformation process, it means we can use them to generate other types of documentation besides just API docs. And Twig can be used to generate other types of text files besides just HTML. By leveraging these possibilities, we are going to create a template that generates a Markdown document using a Twig transformation, that only outputs information about the public API of the classes.

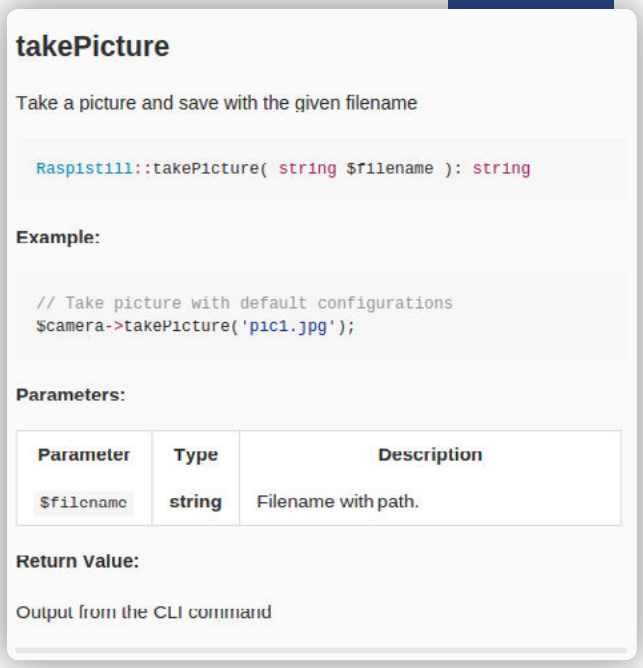
We only output information about the public API because we are generating *external documentation*, which is meant for developers who are using the library, rather than a full API documentation that is *internal documentation* meant for developers working on the code.

The goal is to create a listing of the classes and methods of a library, where each method and their parameters and return

1 *phpDocumentor:* http://www.phpdoc.org

2 *Twig:* http://twig.sensiolabs.org

Final output **FIGURE 1**



takePicture

Take a picture and save with the given filename

```
Raspistill::takePicture( string $filename ): string
```

Example:

```
// Take picture with default configurations
$camera->takePicture('pic1.jpg');
```

Parameters:

Parameter	Type	Description
$filename	string	Filename with path.

Return Value:

Output from the CLI command

LISTING 1

```
01. <?php
02. class Raspistill
03. {
04.     /**
05.      * Take a picture and save with the given filename
06.      *
07.      * **Example:**
08.      * ···php
09.      * // Take picture with default configurations
10.      * $camera->takePicture('pic1.jpg');
11.      * ···
12.      *
13.      * @param string $filename Filename with path.
14.      *
15.      * @return string Output from the CLI command
16.      */
17.     public function takePicture($filename) {
18.         // code here
19.     }
20. }
```

values are documented in an easily readable form with a table of contents at the top. I have taken inspiration from the official PHP documentation for the format, and the final output, as rendered by GitHub, can be seen in Figure 1 and the code from which it was generated in Listing 1. You can also browse the output on GitHub, http://phpa.me/raspicam-php-docs

Defining a phpDocumentor Template

A phpDocumentor template is defined in an XML file called `template.xml`. This file describes the transformations that the template should apply and some additional parameters. An example of a very basic template definition can be found in Listing 2. This template has a single transformation that uses the Twig writer, where all the parsed data will be passed on to a single Twig template and the output will be a single Markdown file with name `README.md`.

LISTING 2

```
01. <?xml version="1.0" encoding="utf-8"?>
02. <template>
03.     <author>Carl Vuorinen</author>
04.     <version>1.0.0</version>
05.     <description>Generate usage documentation in Markdown.</description>
06.     <transformations>
07.         <transformation writer="twig"
08.                         source="templates/markdown-public/index.md.twig"
09.                         artifact="README.md" />
10.     </transformations>
11.     <parameters>
12.         <parameter key="twig-debug">true</parameter>
13.     </parameters>
14. </template>
```

It also defines one parameter with the name `twig-debug` to be true. Parameters are values that the writers can use; in this case, the Twig writer will check for this parameter, set debug mode on, and enable template auto-reload (which is very convenient when developing a template).

Twig Template

Twig is a template engine primarily geared toward generating HTML files, but since it just outputs a text file we can use it to generate any kind of text files. The only downside for using it to generate Markdown is that we can't indent our template file similarly as with HTML templates. Markdown is very specific about whitespace, unlike HTML, where whitespace between tags is collapsed. This affects the readability of the templates somewhat, but we can remedy that by, for example, splitting templates into smaller parts and using includes. Note that going forward we are going to be talking about two different kinds of templates: the phpDocumentor template and Twig templates. I will try to be explicit about which kind of template I am referring to as they are completely different things despite the similarity in terminology.

phpDocumentor assigns a variable named `project` for Twig, which holds all the parsed information of the whole project and all its classes. It is an instance of the class `phpDocumentor\Descriptor\ProjectDescriptor`. The best way to see everything that is available from that variable is to look at the existing templates that come with phpDocumentor or read the source code of all the different *descriptor* classes, since it (ironically) is not very well documented in the phpDocumentor documentation.

In Listing 3 we can see a simple Twig template that outputs the project name as `title`, includes a separate template for Table of Contents, loops over all the classes of the project, and includes a separate class template for each class that is not abstract.

LISTING 3

```
01. # {{ project.name|raw }}
02.
03. {% include 'toc.md.twig' %}
04.
05. {% for class in project.indexes.classes|sort_asc %}
06. {% if not class.abstract %}
07. {% include 'class.md.twig' %}
08. {% endif %}
09. {% endfor %}
```

Twig Extensions

Twig extensions can be used to create filters and functions that are then available inside Twig templates. phpDocumentor allows you to register custom Twig extensions through template parameters. This is an easy way to further modify the output generated by the phpDocumentor template.

In this case we are interested in listing the public methods for a class. Although we could write the logic inside a Twig template, it keeps the template cleaner and easier to read if we extract it to a separate extension. Listing 4 shows a Twig extension function (whole class available in code package) that takes a phpDocumentor `ClassDescriptor` instance and outputs an array of `MethodDescriptor` instances.

LISTING 4

```
01. /**
02.  * @param ClassDescriptor $class
03.  *
04.  * @return MethodDescriptor[]
05.  */
06. public function getPublicMethods($class) {
07.     if (!$class instanceof ClassDescriptor) {
08.         return [];
09.     }
10.
11.     $methods = $class->getMagicMethods()
12.         ->merge($class->getInheritedMethods())
13.         ->merge($class->getMethods());
14.
15.     return array_filter(
16.         $methods->getAll(),
17.         function (MethodDescriptor $method) {
18.             return $method->getVisibility() === 'public';
19.         }
20.     );
21. }
```

To register a Twig extension to a phpDocumentor template, we just need to add a new parameter with the key `twig-extension` like this:

```
<parameter key="twig-extension">
    My\PhpdocTemplate\Extension\TwigClassPublicMethods
</parameter>
```

After that, the new function can be used from a Twig template like so:

```
{% for method in publicMethods(class) %}
    * {{ method.name }}
{% endfor %}
```

Putting it All Together

To get started using this approach of generating usage documentation, you can either use my package[3] from Packagist or create your own template. To install from Packagist, use the following Composer command in your project root directory:

```
composer require cvuorinen/phpdoc-markdown-public
```

The above command will also install phpDocumentor as it is listed as a dependency. To run the actual document generation, you can execute phpDocumentor and tell it to use the correct template by pointing to the directory where the `template.xml` file is located. When you are using the phpDocumentor built-in templates, it's enough to just use the template name, but with custom templates you have to specify the path to the directory so that phpDocumentor can find it. You have to also pass in the source directory that phpDocumentor should analyze and the target output directory where the generated document should be saved to. You can also pass in other optional parameters, such as a title for the project or exclude some files from the analysis, for example.

Here is an example command that uses the template that was installed with the previous command (the following command is broken down into multiple lines so it fits better, but should be executed as a single command):

```
./vendor/bin/phpdoc \
    --directory=src/ \
    --target=docs/   \
    --template="vendor/cvuorinen/phpdoc-markdown-public/ \
                data/templates/markdown-public"          \
    --title="My Project Documentation"
```

As you can see, it can quickly become rather tedious to type, let alone remember, such commands. Especially when you want to add more parameters, such as `--ignore '*Exception.php'` to skip all exception classes from analysis. Luckily there is a way to define all of these in a configuration file. You can create a file with the name `phpdoc.xml` in the root of the project (or tell phpDocumentor where the file is located with the `--config` parameter, if you store it somewhere else) and store all of the configuration options there. An example configuration file can be seen in Listing 5. With the configuration file in place, it's enough to just execute phpDocumentor with the `./vendor/bin/phpdoc` command (or simply `phpdoc` if you have it installed globally).

LISTING 5

```
10. <?xml version="1.0" encoding="UTF-8" ?>
11. <phpdoc>
12.     <title>My Project Documentation</title>
13.     <parser>
14.         <target>build</target>
15.     </parser>
16.     <transformer>
17.         <target>docs</target>
18.     </transformer>
19.     <transformations>
20.         <template name="vendor/cvuorinen/phpdoc-markdown-public/
21.                         data/templates/markdown-public" />
22.     </transformations>
23.     <files>
24.         <directory>src</directory>
25.         <ignore>*Exception.php</ignore>
26.     </files>
27. </phpdoc>
```

Another great way to save the command parameters somewhere so you don't have to type or remember them is to add it as a Composer script. You can add a section with the key `scripts` in your `composer.json` file and inside it define commands where key is the command name and value is the actual command. These commands are run with the composer `bin` directory in path, so it's not necessary to point to it. So you could add, for example, the following scripts section to be able to execute the command by simply running `composer docs`.

```
{
    ...
    "scripts": {
        "docs": "phpdoc -d src -t docs"
    }
}
```

3 *markdown-public template:* https://github.com/cvuorinen/phpdoc-markdown-public

We now have in place a documentation-generation process that can be executed with one simple command. If you want to further automate the process, you can integrate with an existing build process your project might have, or you could use git hooks, for example, but those are beyond the scope of this article. The main thing is that you should now be able to keep your documentation up to date more easily and reliably.

Going Further with Living Documentation

So far we have only used information in the "standard" DocBlock tags, but you don't have to stop there. In the book *Living Documentation*, Cyrille Martraire[4] describes many different ways of generating living documentation. Many of the techniques he describes use custom tags (or Java annotations). We don't have native annotations in PHP, but we can use custom DocBlock tags with phpDocumentor very easily. In fact, phpDocumentor will parse all tags, even ones that are not "standard," and pass them on in the *tags* array of the class/method descriptor. So all we have to do is create a template that uses the desired custom tags.

We could, for example, generate a glossary of the terms used within a project (or for the Ubiquitous language if you are into DDD) by using a custom DocBlock tag and a phpDocumentor template. We can document our code with custom `@glossary` tags as seen in the following snippet:

```
/**
 * @glossary A document that is continually updated
 */
class LivingDocument
{}
```

We could then create a new phpDocumentor template that loops through all classes in the project and looks for the glossary tags and lists the description given in the tag together with the class name. An example of this approach in Twig can be seen in Listing 6.

LISTING 6

```
28. # Glossary
29.
30. {% for class in project.indexes.classes|sort_asc %}
31.
32. {% if class.tags.glossary is not empty %}
33. ### {{ class.name }}
34.
35. {% for glossary in class.tags.glossary %}
36. * {{ glossary.description|raw }}
37. {% endfor %}
38.
39. {% endif %}
40.
41. {% endfor %}
```

This will give us a nice Markdown document that lists all the class names that have been tagged with the `@glossary` tag as keywords and the definition below each one.

You might wonder why you would need to generate such a document when the information is already there in the code. And the answers is, you might not need to. With documentation, you should always challenge the assumptions and requirements by really thinking what is the target audience of any given type of documentation and if it's really needed. Is it internal documentation meant for the developers of the project, or external documentation for other people that needs to be published somewhere? Because any created document should also be kept up-to-date, otherwise the documentation will end up being a big pile of outdated and irrelevant documents that will be frustrating for anyone to read and update.

A glossary like this is not necessarily intended for programmers, but for business people, project managers, or other stakeholders who do not read the code but still need to understand the concepts. Still, it could also be useful for on-boarding new developers to quickly get them up to speed with the project's terminology. Also, just the act of writing all of this information down in the DocBlocks might be useful to make you actually think about the core concepts of the project to come up with good names for the classes and clearly and concisely define their meaning.

This glossary is just a quick proof of concept for demonstrating the capabilities of using custom tags with a simple phpDocumentor template. But if you find it interesting and think that something like this would be useful in a project you are working on, you can develop this concept further so that it fits your project and workflow. Writing phpDocumentor templates is very easy and, at least in my opinion, it's more fun than writing those documents manually :)

Carl Vuorinen is a web developer and team lead at W3 Group Finland. He has over ten years of experience in web development and has been working with commercial PHP projects since 2005. He likes to take on challenging projects and loves to solve problems effectively and deliver quality code that produces real value to the customer/end user.

php[tek] 2016

The
PHP Community's
Homecoming

Saint Louis

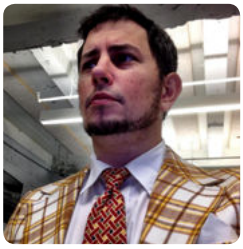
May 23rd - 27th

tek.phparch.com

Keynote Speakers

Cal Evans

Samantha
Quiñones



Jeremy Mikola

AUTOMATTIC

FOXY.IO

Microsoft

nexmo

JET BRAINS

CLASSY LLAMA

A FULL-SERVICE ECOMMERCE AGENCY

Stormpath

robofirm

NEXCESS beyond hosting.

mongoDB

Drupal ASSOCIATION

Community Leaders

Cal Evans

The title of this column is "Community Corner." Since I alone am not the PHP community, I will occasionally outsource the creation of this content to other community members. Don't think of it as me being too lazy to write it; think of it as me synergistically using underutilized resources in the community to further the goals of the project.

Either way, this month's article is written by my good friend Mr. Brandon Savage. Brandon is an active member of the PHP community, an instructor in object-oriented programming in PHP, and the author of the upcoming book *Modern Object-Oriented PHP*.

Cheers! =C=

When you think of a "community leader," what do you typically see in your mind?

If you're anything like me, chances are good that you tend to think about the folks who speak at conferences, who tweet and blog, and who are fairly well known in the community. After all, these are the visible community leaders we have. This relatively small group of people seems to come up with all the great ideas and be involved in all aspects of the community and the development of the language.

But this is only one way of leading a community.

There are others in the PHP community who are quiet and deliberate in their work. They often go unrecognized or unnoticed, but their contributions are crucial to the development of PHP.

In fact, you probably use their code every single day.

Take, for example, the PDO library[1]. Many, if not most, of us use the PDO library regularly with our database interactions. Do you know who wrote it? You may never have heard of Wez Furlong, Ilia Alshanetsky, Marcus Börger, or George Schlossnagle. But they are responsible for this mission-critical piece of software we all use each and every day.

Marcus Börger also helped write the Phar library[2], along with Steph Fox and Greg Beaver. The idea originated with Davey Shafik, who is much better known, but it took this small group of people to bring Phar to fruition and reality. Phar has become crucial to the distribution of PHP components such as PHPUnit and Composer, even though we don't know those who are responsible.

There are many others who work away in silence, making PHP better through their contributions to the documentation, fixing bugs in the core, or just using PHP each and every day.

The really cool thing about being a member of the PHP community is that there's no need to receive an invitation. When you consider the dramatic impact people who aren't seeking the limelight have had on the community at large, it becomes easy to see how you, too, can be a community leader.

Being a community leader doesn't mean you have to stand up and speak at a conference, or write a book, or even be well known. Some leaders are well known, but most leaders in the community go about their business and do good things. Things that all of us rely on.

When I look back at the people who have had the greatest impact on me in my career and time in the PHP community, it's rarely been the most vocal or the most visible people. Instead, it's the countless people who work hard making the PHP language great who have mentored, developed, and encouraged me to become the PHP developer I am today.

One of the great things about open source in general is that nobody needs to be given permission. You don't need authorization to share, or to be a leader. The truth is that open source only works because some people decide to give of their time and skill for the benefit of all. In some ways it's amazing that open source functions at all; yet even as remarkable as the PHP community can be, it still needs new blood—fresh leaders to step up, be ready to take the reigns, and suggest the next big idea that will lead us into the future.

So how can you get involved? The answer is surprisingly simple: solve a problem. Whether that's fixing a bug in the documentation, or offering a patch to core, you can propose your ideas, be involved in the process, and make PHP and the community better.

That's all it takes to be a community leader.

-Brandon Savage

---

1 *PDO:* http://php.net/pdo

2 *Phar:* http://php.net/phar

Past Events

March

Drupalcamp London 2016
March 4–6, London, U.K.
http://drupalcamp.london

Midwest PHP 2016
March 4–5, Minneapolis, MN
http://2016.midwestphp.org

MidCamp 2016
March 17–20, Chicago, IL
http://2016.midcamp.org

PHP Darkmira Tour
March 18–20, Brasilia, Brazil
https://br.darkmiratour.com

Upcoming Events

April

Lone Star PHP
April 7–9, Dallas, TX
http://lonestarphp.com

SymfonyLive Paris 2016
April 7–8, Cologne, Germany
http://paris2016.live.symfony.com

Joomladagen 2016
Aprils 15–17, Zeist, Netherlands
http://joomladagen.nl

SymfonyLive Cologne 2016
April 27–29, Cologne, Germany
http://cologne2016.live.symfony.com

May

New Orleans DrupalCon
May 9–13, New Orleans, LA
https://events.drupal.org/neworleans2016

Meet Magento—Netherlands
May 12–13, Utrecht, Netherlands
https://www.meet-magento.nl

phpDay 2016
May 12–14, Verona, Italy
http://2016.phpday.it

PHPKonf
May 21–22, Istanbul, Turkey
http://phpkonf.org

php[tek]
May 23–27, St. Louis, MO
http://tek.phparch.com

CakeFest 2016
May 26-29, Amsterdam, Netherlands
http://cakefest.org

International PHP Conference 2016
May 29–June 2, Berlin, Germany
https://phpconference.com

PHPSerbia Conference 2016
May 28–29, Belgrade, Serbia
http://conf2016.phpsrbija.rs

June

PHP South Coast 2016
June 10–11, Portshmouth, UK
https://cfp.phpsouthcoast.co.uk

Dutch PHP Conference
June 23–25, Amsterdam, The Netherlands
http://www.phpconference.nl

July

php[cruise]
July 17–24, Baltimore, MD (leaving from)
https://cruise.phparch.com

LaraCon US
July 27–29, Louisville, KY
http://laracon.us

August

Northeast PHP
August 4–5, Charlottetown, Prince Edward Island, Canada
http://2016.northeastphp.org

October

Bulgaria PHP 2016
October 7–9, Sofia, Bulgaria
http://bgphp.org

LoopConf
October 5–7, Ft. Lauderdale, FL
https://loopconf.com

Brno PHP Conference 2016
October 15, Brno, Czech Republic
https://www.brnophp.cz/conference-2016

These days, when not working with PHP, Cal can be found working on a variety of projects like Nomad PHP. He speaks at conferences around the world on topics ranging from technical talks to motivational talks for developers @calevans.

Things We Sponsor

Developers Hangout
Listen to developers discuss topics about coding and all that comes with it.
www.developershangout.io

NEPHP
NorthEast PHP & UX Conference. August 4–5, 2016
2016.northeastphp.org

NomadPHP
Start a habit of Continuous Learning. Check out the talks lined up for this month.
nomadphp.com

Lone Star PHP
Dallas, Texas. April 7–9, 2016
lonestarphp.com

DC PHP
The PHP user group for the DC Metropolitan area
meetup.com/DC-PHP

FredWebTech
The Frederick Web Technology Group
meetup.com/FredWebTech

PhpStorm Intentions for Improved Code Quality

Matthew Setter

Are you as productive as you could be? Are you as efficient with your IDE as you'd like to be? Are you as effective at implementing features as you need to be? Odds are, the answers are no, no, and no. This month, I show you how to leverage PhpStorm's Intentions to write better, more efficient, and more effective code.

This month I wanted to go a little left field, and highlight the three concepts I just introduced—three concepts which we, as developers, always need to be aware of and improve at. Three concepts which are essential, yet often hard to quantify..

Whether one is a new developer or a more seasoned veteran, productivity, efficiency, and effectiveness, or, said more simply, being a software craftsman, are qualities which provide innumerable benefits.

These include:

- Writing code of a higher quality
- Writing less code, yet producing a better result
- Implementing features which are easier to understand
- Implementing features which require less maintenance
- Meeting deadlines with less time and effort
- Reducing unnecessary levels of stress

Some of these benefits are a little hard to quantify. But seeing the significance of these benefits is important; that is, if you want a career which gives both yourself and the people you work for, as well as the people you work with, the greatest return.

They are benefits I increasingly believe to be important, even fundamental, especially after reading such software development classics, as *The Software Craftsman* by Sandro Mancuso.

Consider the following quotes from the book:

> *Simplicity—the art of maximising the amount of work not done—is essential. It is never too late to make things better—it just becomes far more painful when you delay it. Well-crafted software means that regardless how old the application is, developers can understand it easily. When we talk about steadily adding value, we are not just talking about adding new features and fixing bugs. This is also about constantly improving the structure of the code, keeping it clean, extendable, testable, and easy to maintain.*

Depending on where you are in your software development journey, some of these concepts might seem downright foreign to you—even absurd. Perhaps you feel that software development is all about cranking out as much code as you can, that the person who has the greatest LOC (lines of code) count is the best developer in the organization—the one who's most deserving of praise and recognition.

Assumptions like these are not only abjectly wrong, they're laughable and even dangerous. They're dangerous because they actively perpetuate the myth that code is all about quantity over quality. They perpetuate the myth that being a "rock star" is desirable over being an artisan, a craftsperson, or a journeyman.

Assumptions like these, in my opinion at least, are at the root cause, if not *the* root cause of a lot of the problems relating to lack of trust and poor productivity in software development. These problems have plagued our industry for decades. It's for these reasons that I want to do my part to help eradicate such assumptions and improve the overall recognition and status of our industry. In doing so, I want to help you do more with less, and be more productive, effective, and efficient.

Productivity, Efficiency, and Effectiveness

But before I get to exactly how I'm going to help you, the subject of this month's column, what *is* productivity, what *is* effectiveness, and what *is* efficiency? According to dictionary.com[1], Productivity is:

> *"The quality, state, or fact of being able to generate, create, enhance, or bring forth goods and services*

Efficiency[2] *is:*

> 1. *the state or quality of being efficient, or able to accomplish something with the least waste of time and effort; 2. an ability to accomplish a job with a minimum expenditure of time and effort:"*

And effectiveness[3] is:

> *"adequate to accomplish a purpose; producing the intended or expected result."*

But what are they for us as developers? To me, at least, productivity, efficiency, and effectiveness are the ability to develop meaningful applications and write useful code, while expending the least amount of effort required to do so. What's more, it's essential that we're able to do so increasingly well over time.

But the question is—how? How do we constantly improve over time? There's a lot of theories and philosophies as to how. One of my favorites is the 10,000 hours theory, made famous in the book *Outliers*, by Malcolm Gladwell. The theory can be summarized as follows:

> *the key to achieving world-class expertise in any skill, is, to a large extent, a matter of practicing the correct way, for a total of around 10,000 hours.*

1 *Productivity:* http://www.dictionary.com/browse/productivity

2 *Efficiency:* http://www.dictionary.com/browse/efficiency

3 *Effectiveness:* http://www.dictionary.com/browse/effectiveness

The theory goes on to say that it takes this amount of time for the brain to truly bed down the skill, such that it becomes an implicit way of thinking and working, something as well understood as breathing.

It's one of my favorites because it's one which can be applied each and every moment which we're writing (or refactoring) code. It can be—if we're using the correct tools. And one of those tools is the source of this month's column; one which is found in one of the best IDEs available for PHP: PhpStorm. It's called *intentions*.

Intentions

But what are intentions? Paraphrasing PhpStorm, intentions detect issues with your code which can result in errors and hard-to-detect bugs, such as undeclared, or magic, variables, using classes with a fully qualified namespace, undocumented functions, and assertions which aren't specific enough.

They complement PhpStorm's other code analysis and inspection tools, either by suggesting ways in which code can be improved, or providing a range of implementable fixes. While easy to action, as we'll soon see, they're also unintrusive in how they're presented. Let's start exploring them in more detail.

Intentions in Action

Whether that's a small script or code which forms part of a much larger application, intentions help us improve the quality of our code by continuously reflecting and improving upon its meaning and significance. It intermittently reviews what we've written, and if it finds something questionable, indicates that it's found a potential problem along with providing a set of possible ways in which it can be improved.

Take the code snippet I wrote in PHPStorm shown in Figure 1. There you can see that on line 30 I have used a magic variable, and on line 39, I've referenced a class unnecessarily using a fully qualified namespace. While neither of these are strictly "wrong," the first isn't really necessary and could harm readability. The second could lead to hard-to-debug issues, depending on how the code evolves over time.

FIGURE 1

```
\App\Action\BusinessPageAction  __invoke
14  class BusinessPageAction
15  {
16      /** @var Template\TemplateRendererInterface ...*/
19      private $template;
20
21      /** HomePageAction constructor. ...*/
26      public function __construct(
27          Router\RouterInterface $router,
28          Template\TemplateRendererInterface $template = null
29      ) {
30          $this->router = $router;
31          $this->template = $template;
32      }
33
34      public function __invoke(
35          ServerRequestInterface $request,
36          ResponseInterface $response,
37          callable $next = null
38      ) {
39          return new \Zend\Diactoros\Response\HtmlResponse(
40              $this->template->render(
41                  sprintf(
42                      'app::%s-page',
43                      strtolower(basename($request->getUri()->getPath()))
44                  ),
```

So what can we do about it? Well PhpStorm has highlighted the presence of the fully qualified namespace usage, so let's start there. By hovering over it, as I have in Figure 2, you can see it shows both the reason why it's a problem, and what can be done about it. If you click anywhere on that line and wait a second or two, PhpStorm displays a light bulb icon in the far left-hand side. Clicking on it, you see a pop-down menu appear, which presents a series of solutions, these being:

- Generate PhpDoc for function
- Import class

FIGURE 2

```
callable $next = null
) {
    return new \Zend\Diactoros\Response\HtmlResponse(
Inspection 'Unnecessary fully qualified name' options
Adjust code style settings
Generate PHPDoc for function
Import class
                                                    ->getUri()->getPath()))
            ),
            []
        )
    );
}
```

Let's say that I see this as a problem, and decide to import the class. By clicking that menu option, PhpStorm imports the class and removes the fully qualified namespace. Now let's have a look at line 30. Again, by clicking on it, a light bulb appears. And by clicking on it, we're presented, this time, with four potential solutions.

These are:

- Add `@property`
- Declare field
- Surround with an `if(!empty)` construct
- Surround with an `if(isset)` construct

To me, the one which makes most sense is to declare the field. So that's the one I'll choose. After that, a small dialog appears, asking what visibility should be given to the new variable, to which I'll respond with private. Following that, a new private member variable is created, removing the source of the concern.

Let's reflect on what we've just done. Through minimal effort on our part, we've been able to improve the quality of our code and, in the process, learned more about what constitutes higher quality code. We've had a second set of eyes look over our code and give us meaningful suggestions as to how we can improve it. It's rather akin to basic pair programming.

While we're continuously improving our knowledge, we have a helping hand which continues to point out things which we may not have considered yet, not seen, or perhaps not even thought of.

What About Code Documentation?

This might not be something which you see as important. But high-quality code has, as a core component, good documentation. What's more, if you're using PhpStorm or another IDE, then well-documented code—here I'm specifically referring to DocBlocks—will help you be more productive. Given that, have a

FIGURE 3

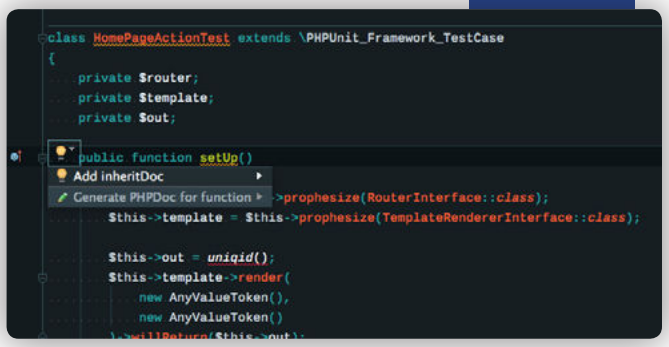
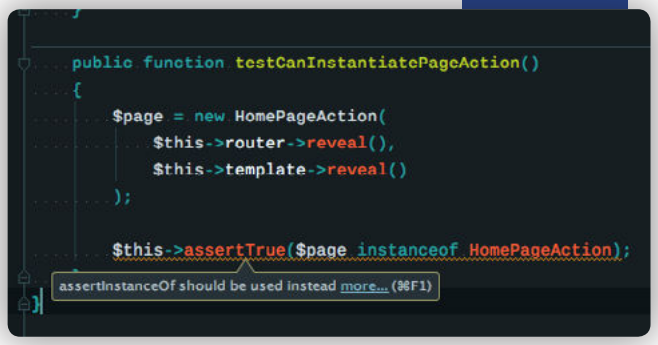


FIGURE 4



look at Figure 3. Here I'm looking at an intention which appeared while I was creating a basic test, using Codeception, for a new Zend Expressive app I'm creating.

You can see that the `setUp` method's been implemented. Given that the test class extends `PHPUnit_Framework_TestCase`, then this method overrides the method of the same name implemented there. The definition in the parent has the following DocBlock comment:

```
/**
* Sets up the fixture, for example, open a network connec-
tion.
* This method is called before a test is executed.
*/
```

Since my method's largely the same, for all intents and purposes, it makes no sense to write a different DocBlock comment for the method. But without an `{@inheritdoc}` annotation, the IDE will show no information for the method. So PhpStorm has flagged this as a potential problem, and offers to do this for us. Clicking on **Add inheritDoc** adds the annotation.

Using Proper Assertions

Now let's look at another one, one perhaps more technical. In Figure 4, I'm performing an assertion that the object `$page` is an instance of `HomePageAction` by using the `assertTrue` assertion. Nothing wrong with that. The intent is achieved. But I could be more specific, by using the `assertInstanceOf` assertion, and, in doing so, make the code more readable. PhpStorm thinks so too, and offers an intention to do just that. As it makes sense, clicking on the intention makes the code change.

Now let's look at a series of other intentions. In Figure 5, you'll see that this time, clicking on the light bulb has presented quite a significant list of intentions which could be implemented. I could replace the double quotes with single quotes, because this intention suggests that this was to avoid the PHP lexer from attempting to perform any variable interpolation.

I could swap the positions of the two arguments; I could even correct the spelling of the word, if it were misspelled, which it's not. Or I could add the word to the current dictionary. This example shows that intentions aren't strictly bound to code-related concerns. They go beyond that, helping you improve the quality of your code holistically.

Configuring Intentions

We've looked at a range of intentions which come prepackaged with PhpStorm. What if you're not interested in them? What if you want to configure the inspections specific to your codebase and developers' experience levels? Consider this example: Say you've taken over a legacy codebase, and it's one big "ball of mud." If the default intentions configuration was in place, you'd likely be getting suggestions all over the place. Arguably, this is something which would be more annoying than helpful.

At this stage of development, I'd suggest it's better to tone them down, temporarily. Then progressively, as the code starts to come into better shape, re-enable the disabled intentions. If that sounds like something you're interested in, here's how to do it. First, when you've clicked the light bulb, instead of clicking on an intention to implement it, right-click to show a successive popup.

The first option will be *Edit Inspection Profile Setting*. Click that to display the intentions configuration dialog. That will take you to the configuration of the intention you were just looking at. It provides a description of what it does and is for, along with the severity, and scope.

FIGURE 5

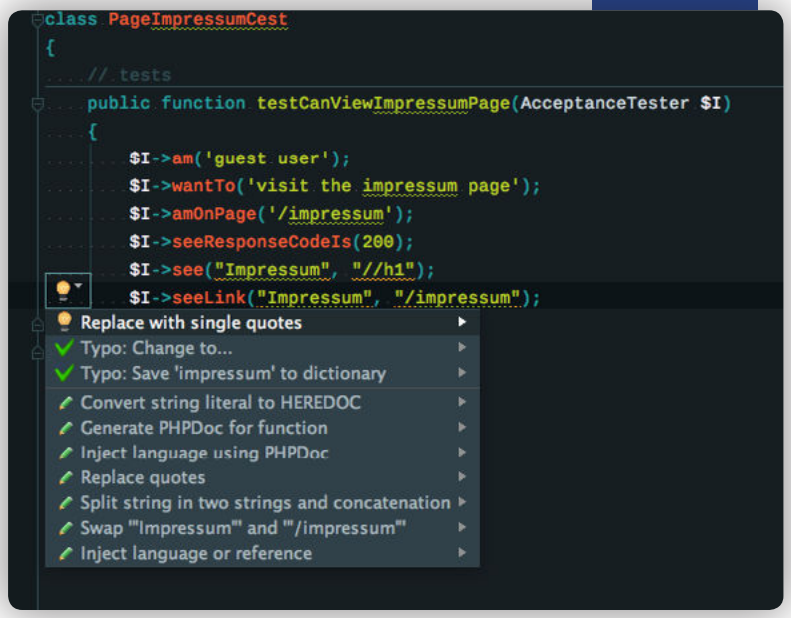
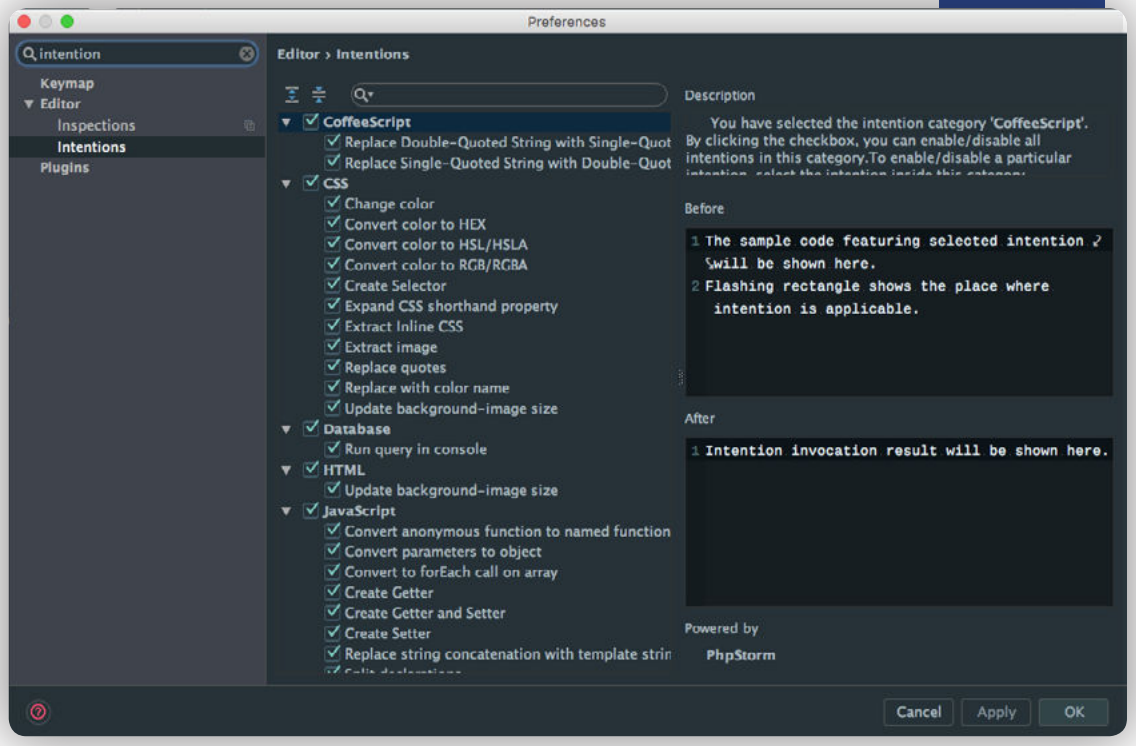


FIGURE 6



The other, and more complete, way is to open the *Settings/Preferences* dialog, by clicking ⌘, on Mac and Linux or choosing **File -> Settings** on Windows, and search for "intentions" as you can see in Figure 6. Out of the box, PhpStorm has intentions for a range of the languages and file types which you'd normally expect to work with as a PHP developer. These include: *CoffeeScript*, *JavaScript*, *TypeScript*, *XML*, *XSLT*, *CSS*, *Regular Expressions*, *SQL*, and *HTML*.

What's more, depending on the plugins you install, you can expand on that list. For example, you can expand on the MultiMarkdown plugin, which provides a host of support for working with Markdown files. if you click on any intention, you'll see a description of what it does, and a before and after example if the intention were to be used. To disable an intention, uncheck the checkbox next to it. To disable all within a category, uncheck the checkbox next to the category's name. Don't feel the need to accept the default set. Make changes that best suit you, your team, and your organization.

And That's a Wrap

I could work through each of the intentions in turn. But I don't see that that would provide any real value to you. The ones which I've gone through, along with the information about configuring them, should be enough to both whet your appetite and provide you with all the information you need to configure your PhpStorm installation to suit your specific needs.

I hope that you appreciate just how effective this feature can be for improving the overall quality of the code you and your team produce, as well as how it aids in the journey as a software craftsperson. I'd love to hear what you think, and what your experiences have been. If you're keen to share, then add your thoughts on the php[architect] Facebook page. Or tweet me @settermjd. Until next month, happy coding.

Matthew Setter is a software developer specializing in PHP, Zend Framework, and JavaScript. He's also the host of http://FreeTheGeek.fm, the podcast about the business of freelancing as a software developer and technical writer, and editor of Master Zend Framework, dedicated to helping you become a Zend Framework master? Find out more http://www.masterzendframework.com.

Requirements

- PhpStorm version 10 or above

Learning a New Framework

David Stockton

Many of us know our preferred framework or platform of choice very well. When faced with an upgrade, a platform evolution, or a new task that requires looking into and learning a new framework, the prospect can be a bit daunting. Whether you have to learn a new framework or platform to keep current on your day-to-day job or just because you love learning, I have a few tips I hope will help make your journey a little easier.

How Did it Come to This?

There are two main reasons to learn a new framework or platform. The first is because you're curious and you want to see what it offers. In this case, you've either already got a framework you like and are familiar with, or it's your first framework, or perhaps you haven't found a framework you really like. You're investigating because you personally feel in some way that it would be good to look into this new framework. Nobody may be forcing you to learn it, but you want to.

The second is that you're effectively being forced to learn it. Perhaps the framework you know and love is no longer being supported. Perhaps your platform of choice is adopting a new way of working, bringing in components from a framework and essentially requiring that to you learn some new and foreign things if you want to stick with your platform. If you're a Drupal developer, the latest version is bringing in quite a bit from the Symfony framework, allowing for greater interoperability with frameworks and components from outside the Drupal ecosystem.

If you're a longtime Zend Framework developer, you may remember what a huge shift it was to move from Zend Framework 1 to Zend Framework 2. That's the transition I went through. Essentially everything about the framework changed except for how it was spelled. Because of this enormous shift, a lot of applications remain on the first version. Just around the corner is Zend Framework 3, which is more of a philosophical shift than the "change everything" approach that marked the move from version 1 to 2.

My most recent foray into a new framework came with trying to learn Zend Expressive, a microframework that uses a middleware approach and Zend Framework 3 components. While I am quite familiar with Zend 1 and 2, middleware was something new, with only a brief bit in a conference tutorial looking into the Slim framework.

The Role of Frameworks

In the last decade of PHP, frameworks have moved from being nearly unheard of to ubiquitous. Dozens, if not hundreds, of new frameworks have popped up, some flourishing, others quickly fizzling out. We've seen large frameworks such as Zend Framework 1 and 2, and Symfony. Laravel, while being relatively new, has enjoyed a meteoric rise in popularity. CodeIgniter and CakePHP were aimed at developers wanting some of the simplicity of Ruby on Rails, but they brought about many of the restrictions as well. Phalcon's goal is speed, with the framework itself being implemented as a C extension. Yii is built to work with third-party code, allowing easier integration with other frameworks. Aura provides fully decoupled components without external dependencies. In short, there's probably a framework that matches your coding style and philosophy, if not overall, then at least for a particular project.

Another movement I've seen over the last couple of years is the move toward microframeworks. These are smaller frameworks that aim to provide a lot of the common functionality, such as routing and configuration, but not everything. Often these are based on components from a major framework. There's Lumen, based on Laravel; Expressive, based on Zend; Silex, based on Symfony; and Slim, which uses a few of the PHP Framework Interoperability Group's PSRs (and others, of course). Whatever your framework choice and for whatever reason, I hope this will help you get going quickly.

Jump in With Both Feet

As I mentioned, my most recent foray into a new framework was with Zend Expressive. This was done by choice. I've been using Zend Framework 1 and 2 since before their official 1.0 release and Apigility since it was announced at ZendCon in 2013. As a result, I know the Zend stuff pretty well, so I thought looking at Expressive would be a lot of nothing new. I was wrong. The shift from "MVC" to middleware requires a change in thinking and understanding what the actual problem is, rather than determining how you would have solved it with the framework you're familiar with. When learning any new framework, there will be a bit of a learning curve bump to get over.

With Expressive, I decided I wanted to build a project that would stick around, rather than a throwaway app. Not something huge, but something I needed. Whether the project went well or poorly, I was going to finish it. This leads me to my first bit of advice when learning a new framework.

Make a Small Real Project

There's a lot to be said for following tutorials or watching videos and reading blogs. All of those are valuable and worthwhile, but if you're not actually doing something real with a new framework, chances are you'll quickly reach a hard limit on what you know about it. So my recommendation is to *build something real*. What I mean by that is a project that you're going to launch or use, at least for a while. Many tutorials take you through building a blog, or a book or movie or CD tracker. These are all fine and good, but beyond adding a book or movie, you're never going to touch them again. You likely won't add any features or functionality beyond what the tutorial told you to add.

To really learn a framework, build a project for something you're going to use. If you're going to be using it, chances are you'll need to come up with new features and you'll need to learn how to make them in your new framework.

For me, the project I wanted to build with Zend Expressive was what I'm calling a webhooks box. We have a series of Jenkins Continuous Integration build servers, as well as projects hosted on Bitbucket and Jira, and we use Slack to communicate. Our Jenkins servers are not publicly accessible, but I wanted to make a way that would allow events or webhooks from each of these services to cause events such as builds, notifications to Slack, and updates to statuses in Bitbucket and Jira. I want to be able to cause tests to run, builds to go, approvals, etc., to happen, because of messages in Slack. So this will be an ongoing and continually updated project for me.

Apply What You Know, Be Open to Learning

Zend Expressive and Zend Framework 3 have been heralded as an end to framework silos. In a testament to this, when installing Zend Expressive, the framework gives you a number of choices for various pieces. It supports routing via `Aura.Router`, `FastRoute`, and `Zend Router`. It supports dependency injection through `Aura.Di`, `Pimple`, and `Zend ServiceManager`. For rendering HTML, it supports `Twig`, `Plates`, and `Zend View`. With these choices, I had the advantage of being able to pick some things I am familiar with while choosing others that I am not at all familiar with. In my case, I chose `FastRoute`, `Zend ServiceManager`, and `Plates`. Of these, `Zend ServiceManager` is something I'm quite familiar with, but I've never used the others.

Editors Note: *For an introduction to Zend Expressive, check out our March 2016 issue.*

Through my work with Zend Framework and Apigility, I am familiar with `Zend Router` and a little familiar with `Zend View`, but I decided to pass them up and see how other libraries work for these things. If you're experienced in a framework, then many of the concepts you'll need to learn for a new framework may be familiar, but the implementation details may not be.

Learn the New API

Zend Expressive allows many choices for what dependency injection container, view renderer, and router you want to use, because in each case, each of the choices follow the same sort of pattern. The middleware that is used follows the interfaces specified by PSR-7. The dependency injection containers follow `ContainerInterop`, which effectively means that by learning some of these new libraries, I'll have a good head start on learning the others.

Each framework has these concepts, whether they are provided by the framework or through external components like they are with Expressive. This means there will be a way to do routing, a way to build objects and inject dependencies, ways to render views, perhaps ways to connect to and interact with databases. In some cases, the framework may provide even more, such as interactions with web APIs, payment gateways, generating PDFs and barcodes, working with file systems (both remote and local), database migrations, and more.

Becoming familiar with what the framework provides can help you when building out your application. That being said, I feel the way forward is for frameworks to provide less and components and packages to be included where needed. To see where some of this is heading, I urge you to look at what the PHP-FIG is doing. In addition to creating recommendations for coding styles, autoloading, caching, logging, and HTTP messages, they are working on recommendations for interfaces that will help libraries interact for hypermedia, containers, and documentation. These interfaces will help ensure a much easier transition from libraries that use them to other libraries and implementations with those same interfaces.

Coming back from FIG land to the API, if your framework doesn't use these interop interfaces, you'll need to learn about how they do each of them. For instance, does the framework provide for dependency injection? If so, is it handled automagically, through configuration, through factories, or something else? Are the factories defined as classes, callable closures, arrays of `$key => $value` mappings, or something else?

Is routing handled via configuration, convention, method calls, or something else? For frameworks that provide routing details via configuration, you many need to learn XML, YAML, JSON, or how the particular flavor of PHP arrays needs to be arranged. If routing is done via convention, it means that you'll place certain kinds of files in certain places with particular names.

In my case with learning Zend Expressive, the new API that I needed to learn concerned middleware. It's a pattern where there's an incoming request and an outgoing response, and in between a number of small pieces of software called "middleware" execute. At each step, each of these functions or classes makes a small change to the response or makes a decision and then the control passes to the next bit of middleware. From what I've seen, it allows for a lot of transparency in what's happening since the framework is mostly there to take away a bit of boilerplate. At any point along the way, a piece of middleware can return a response which means the rest of the middleware doesn't need to run.

It also means that many of the paradigms I'm used to when dealing with applications using an MVC-style framework do not apply here. I've had to change how I think a bit to get things working, but it has also led to what I feel is a group of smaller, more easily testable, and more composable bits of code. So far it has been quite educational and enlightening. I'm definitely looking forward to what I'll be learning in the upcoming weeks.

Sometimes frameworks provide 42 different ways to do the same thing. Sometimes they have terrible documentation or none at all. In this case, looking at the source is a good idea. Another way to find out what's happening behind the scenes is to connect a debugger (See the January 2015 issue for more on this) and step through what's happening. This can be very helpful and enlightening in discovering how the framework does its magic and hopefully determine how it expects you to work with it.

Contribute to Open Source

After you've gone through the tutorials and videos, how to work with the framework may still be a bit of a mystery. The designers and developers who created the framework usually had some idea about how it would be used, but unless they've documented it clearly, it may not be readily apparent what they were thinking. That's where contributing to open source comes to play.

If you can, find an open-source project that uses the framework you're trying to learn. There's a good chance the people who created the project know a thing or two about the framework. You can use the project as a bunch of examples about how the framework can do things. If you are able to make changes—think fixing bugs,

adding features, even writing documentation for the project—you may be able to pick the brains of the maintainers and ask questions about how they might do something if you're trying to figure out the framework.

Additionally, with a fully realized project, my suggestion for using the debugger works here as well. It can help you understand how the project is organized and how the project uses the framework and components it depends on. Sometimes the project may provide additional clues and hints on how to provide contributions that lead to further insight about how the framework works. Working with a full project will almost certainly provide more information about a framework than any tutorial or video.

Contribute to the Framework

Once you've established a bit of familiarity with the framework, another valuable step is to start contributing, or at least trying to contribute, to the framework. Take a look at the framework's GitHub page and look at the issues. See if there's something listed that sounds like it's not too far off the beaten path. Try to reproduce it by building a small example with the framework. If you're able to reproduce the issue but don't feel able to fix it, commenting on the issue that you were able to make it happen and how to replicate it can be very useful information for the maintainers. If you can fix it, put together a patch and make a pull request.

When you become more familiar with some aspect of the framework, look at the documentation around that area. If it's lacking, explained poorly, or could use additional examples, consider writing new documentation or augmenting the docs. Contributing code or documentation back to a framework is a great way of saying thank you to the people who created and maintain the framework. Imagine how much easier you might have found learning the framework with your new and improved documentation in place. Consider it a way of "paying it forward." Improving the docs means that other newcomers will likely have an easier time learning the framework, which means more users, more contributors, and ultimately a better framework with better support.

Practice, Practice, Practice

Ultimately, the best way to learn a framework well is practice. This, of course, applies to all areas of life. The best way to get better is practice. This means continuing to use the framework, building new features into your application, changing existing features to use new, different, and better ways of realizing the framework. It means getting involved in the community, joining the IRC channel for the framework, asking questions, answering questions, and getting feedback. You can go on StackOverflow and ask and answer questions. If the framework is new and you feel you've learned it well enough, consider speaking to your local user group to introduce others to the framework and share what you've learned. Sharing your knowledge and ideas in a written or spoken way is a great way to solidify the ideas and concepts you've learned.

This Framework Sucks, Now What?

In some cases you may find that you don't like the framework or it doesn't do what you need it to. That's OK as well. Not every framework will appeal to everyone. If it did, we'd probably need only one framework, not hundreds. If you finally decide that the framework isn't for you, now you have a choice. If it's close to what you'd like but not quite there, you could consider contributing to fill in the gaps. Of course, the maintainer has no obligations to take your patches, but it is an option. You could fork the framework and make your own version. You could start your own. If you decide to do this, I recommend it mainly as a learning exercise. You can, of course, create and release your own framework, but chances are unless it's doing something particularly elegant, novel, or special, it's unlikely you'll gain many users.

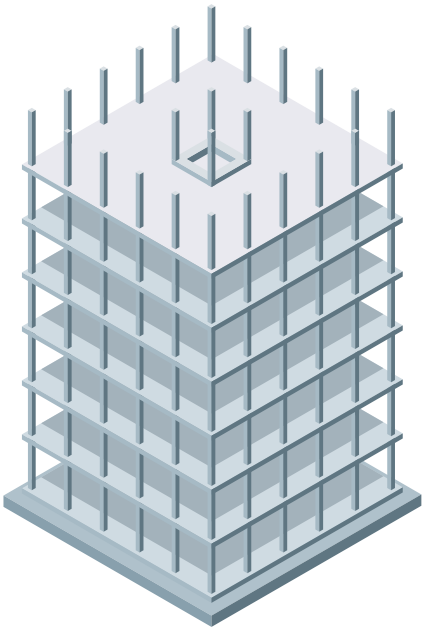
Fortunately, there are tons of other frameworks out there. As I've mentioned numerous times in previous articles, one of the best ways to level up is to not stop learning. Pick another framework and start another project with it. Learn it, contribute to it, and decide if you want to continue to work with it. If you find that you do like it, spread the word. Help others find and use it, and help contribute to it with code, docs, or bug triage. Open source thrives because of the contributions everyone makes.

Now What?

Whether you've got a reason that is being imposed on you to learn a new framework, or you just want to learn something new to keep your skills fresh and learn new ideas, taking the time to learn a new framework can be a great experience. It can lead to new opportunities at your current position, or new contracts. If you contribute by sending pull requests, speaking, or writing, it can be a great way to meet new people, make friends, and have more fun with your work.

The bottom line is, don't be afraid of a new framework. Look at it as an opportunity to learn and improve and absorb new ideas. It's also an opportunity to contribute and give back. Framework authors and contributors are learning and growing as developers as well. This means new frameworks and new versions of existing frameworks will continue. By learning about them and, hopefully, contributing to them, this cycle will continue. See you next month!

David Stockton is a husband, father and Software Engineer and builds software in Colorado, leading a few teams of software developers. He's a conference speaker and an active proponent of TDD, APIs and elegant PHP. He's on twitter as @dstockto, YouTube at http://youtube.com/dstockto, and can be reached by email at levelingup@davidstockton.com.



Securing Legacy Applications–Part 2

Chris Cornutt

Last month, I talked about legacy applications and provided some general "quick hit" tips to help you secure your application. I'm going to continue the same theme this month and offer a few more helpful hints you can use immediately. You'll notice as I go through these points that they all have something in common—they're relatively generic. Legacy applications are a completely different beast when it comes to the mishmash of technologies, development practices, and naming conventions. Unfortunately, this also means that there's no way to really effectively secure it without taking a good look at how it's doing things.

Consider this a word of warning that your mileage may vary when it comes to applying these tips. Don't expect that, if you treat these like a checklist, your application will be magically secure. After all, most of the security issues (probably 9 times out of 10) stem from the integration of functionality and tools, not the tools themselves.

Using Unfiltered Data

If you've done any reading about application security, you'll see one theme mentioned over and over: "don't trust the user." What this basically boils down to for you the developer is that any of the input you get for your application to use should be considered tainted. I briefly touched on this in the section about `$_REQUEST` in last month's article, but I want to expand on it further here and offer suggestions on how you can avoid using potentially malicious data. There are two crucial things to think about when working with input: validation and filtering.

> *As a side note, there's a difference between "filtering" and "escaping," although many will use those terms interchangeably. Filtering is the act of taking the data passed in and removing malicious/bad data from it before use, whereas escaping is taking the data provided and removing the malicious content prior to output.*

The basic definition of *validation* is ensuring that the data you've been given is what you're expecting. If you're looking for a US phone number, you definitely want the three-three-four format (or just 10 digits without country code). The same with things such as Social Security numbers or credit card numbers—you know what they're supposed to look like, so you can verify that structure. That's all validation is—verifying the structure and type of the incoming data. Some, like the numbers I mentioned before, are easier to judge, but others, like free-form text, are a bit more difficult. In that case, you need to determine what kind of text you might allow (hint: usually denying HTML content is a good place to start.)

Here are some examples of validation:

- Structured data (like phone numbers or credit card numbers)
- URLs
- E-mail addresses
- Required

Required? Yes, that's technically a kind of validation, too. Most people think of it as something separate, but it should be treated the same as any other rule is treated. Whereas there are solutions for validation built into PHP, like `filter_var`[1], I recommend pulling in a good validation library that lets you make more complex validations. There's a great one from the Particle-PHP project, the Particle\Validator library[2], that "makes validation fun," with an easy-to-use fluent interface. Hence, for example, say we wanted to verify that the string the user gave us exists, is a certain length, and is only alpha-numeric characters. Although we could use PHP itself to do these kinds of checks (`isset`, `strlen`, and a regular expression), the library makes it a one-line affair:

```
$validator->required('input')->lengthBetween(2, 10)->alnum();
```

By running that through the validator and providing it with the data to check, you can easily check for a match. Some frameworks will also come with their own validators. Laravel, for example, makes a validator available in its controllers for easy use. You can just pass the request itself into it and define the rules as an array:

```
$this->validate($request, [
    'email_address' => 'required|email'
]);
```

There are plenty of options out there, but for legacy applications, I highly recommend pulling in a library and creating validation that way. You could even introduce it at a model layer (you've refactored to use models, correct?) and then run a `validate()` method to make the logic reusable.

Protecting Forms with CSRF Tokens

Another relatively easy security feature to introduce into your legacy application is the idea of Cross-Site Request Forgery (CSRF)[3] tokens. These are randomly generated tokens that in effect prove that the data being submitted did come from your site. Here's how these tokens work:

1. When a user comes to your site and pulls up a web page with a form, the PHP generates a random hash.

1 *Filter Extension:* http://php.net/book.filter

2 *Particle Validator:* http://validator.particle-php.com

3 *Cross-Site Request Forgery:* http://phpa.me/wikipedia-csrf

2. This hash is then stored in the current session and output as a hidden field on the form.
3. When the form is submitted, the two values are then compared.
4. If there's a match, you're ready to go. If not, then it's a potential security issue, and the data should be ignored.

Usually the form field will look something like:

```
<input type="hidden" name="_csrf_token"
value="3441df0babc2a2dda551d7cd39fb235bc4e09cd1e4556bf261 \
bb49188f548348"/>
```

And the code to generate the hash is relatively easy, too:

```
$hash = hash('sha256', openssl_random_pseudo_bytes(256));
```

What problem are we trying to solve here? Because of the way the web works, any site can make a request to any other site using one of the HTTP methods (like `POST`, `GET`, `PUT`, etc.). This means that a completely different site could be sending a `POST` request to your server with whatever data that site would like. By introducing these tokens, you can outright dismiss these outside submissions and drop the data it has sent.

Here are a few tricks to using these tokens to keep in mind, too:

1. You'll need to be sure to use a solution that tracks more than one token at once. If you're only tracking one token, in this day and age of multiple tabs, the user could open another page, and the token on that page would be the one in the user session. One alternative to this is to use the two-field solution that the CSRF middleware for Slim v3 uses: one field for the token name, the other for the token hash—both randomized.
2. When making the hash itself, be sure to pull from fully random data (like the `random_bytes` or `openssl_random_pseudo_bytes` functions) and hash it with at least an SHA-256 hash. Tokens don't need to be cryptographically strong; they just need to be random and unpredictable.
3. Do not—I repeat—*do not* reuse tokens. There are some implementations that will have you hard code tokens and reuse those. The key to the CSRF hashing is that it is random. In fact, it's a good idea to randomize it for every page load and every form.

These two suggestions are a bit more complex than are the ones in the previous article, but they're still things that are completely within reach when refactoring for security in your legacy application. Although the CSRF tokens are a bit more of a "quick hit," the validation will likely take longer to get right, especially if your software is currently accepting a wide range of potential data and not just simple form posts. Keep in mind, though, that one of our primary goals is to simplify. These suggestions are relatively simple but could get overwhelming pretty quickly if you're not careful. As the famous author Ralph Waldo Emerson once said, "Nothing is more simple than greatness; indeed, to be simple is to be great."

For the last 10+ years, Chris has been involved in the PHP community. These days he's the Senior Editor of PHPDeveloper.org and lead author for Websec.io, a site dedicated to teaching developers about security and the Securing PHP ebook series. He's also an organizer of the DallasPHP User Group and the Lone Star PHP Conference and works as an Application Security Engineer for Salesforce.

March Happenings

PHP Releases

- PHP 7.0.5: http://php.net/archive/2016.php#id2016-03-31-1
- PHP 5.6.20: http://php.net/archive/2016.php#id2016-03-31-4
- PHP 5.5.34: http://php.net/archive/2016.php#id2016-03-31-2

News

Paul Jones: Producer: Validate and Release PHP Library Packages

In this post to his site Paul Jones introduces a tool that aims to help you and your Composer-centric workflow, making it easier to validate and release packages for your projects: Producer. He goes on to talk about why you might want to use Producer in your workflow and its functionality for validating and releasing packages. He also answers some of the common questions he's gotten about the tool, mostly around the steps it takes during the validation/release process. http://phpdeveloper.org/news/23799

Symfony Finland: Angular 2 Universal Rendering Coming to Symfony and Twig

As is mentioned in this new post on the Symfony Finland site, the functionality allowing the Twig templating engine to perform Angular 2 Universal Rendering. He points out that there's also a chance that the result could be coupled to Drupal (at least at first) but hopes are that it will end up as a generic component usable in any system using Twig. The post also includes links where you can find out more information about the topic of "Isomorphic Rendering" too. http://phpdeveloper.org/news/23819

AppDynamics PHP Blog: Predicting the Future of PHP Security–Part 3

On the AppDynamics blog there's a post from Omed Habib where he looks at the current state of security in the PHP language and makes predictions about the future of it in PHP and where the language might be heading. In the article he talks about PHP's popularity and how it has somewhat worked against it and its reputation when it comes to secure development. He covers PHP 7 and some of the security-related updates that came with it. http://phpdeveloper.org/news/23803

Joe Ferguson: User Group Advice

If you've been thinking about starting a local technology user group in your area but have been looking for some guidance, you should definitely check out this recent post from Joe Ferguson with some good "dos" and "don'ts" around groups and organization. He starts with the "dos" (like keeping it small to start and making meetings consistent) and "don'ts" (not to worry about sponsors and making the talks "conference level" every month). He also includes an interesting section about "protecting" your user group, preventing things like over-zealous recruiters from coming in and giving people a bad impression of the group. http://phpdeveloper.org/news/23780

Symfony Blog: New in Symfony 3.0

On the Symfony blog there's a new post briefly looking at Symfony 3 and what's different about it as compared to previous releases (and what's not). Each of the changes on their list include links to get more information about the component and the post wraps up with a quick "how-to" on upgrading to Symfony 3 from other releases. http://phpdeveloper.org/news/23771

Edd Mann: Mapping, Filtering and Reducing in PHP

Edd Mann has a post to his site talking about the use of things like "map" and "reduce" in his own development and how to use it in PHP to help reduce data sets and filter them based on certain criteria. He starts with a simple array of data: a set of users with their respective IDs and names. He shows a typical approach, using a method that loops through the data to find only the "name" values. He then shows an alternative that makes use of PHP's own array_filter and array_reduce functions to perform the same operation with just a bit more internal handling. http://phpdeveloper.org/news/23729

Codeception blog: Codeception 2.2 Beta

Codeception released a beta of version 2.2, which brings a number of changes to this testing tool. The biggest addition is the support for Gherkin and `.feature` files. This lets you use Codeception to do Behavior Driven Development just like with Behat. http://codeception.com/03-26-2016/codeception-2.2-beta.html

Badoo Tech Blog: How Badoo Saved One Million Dollars Switching to PHP7

The team at Badoo shares the results of upgrading hundreds of their application servers to PHP 7. While upgrading, they were also able to find bugs in the opcode cache system. They discuss alternatives, including HHVM, that they considered before deciding to wait for PHP 7. The article also includes graphs for memory consumption, CPU usage, and load which illustrate the dramatic improvements they saw as a result of upgrading. http://phpa.me/badoo-php7

Julien Pauli: PHP's OPCache Extension Review

Julien Pauli, one of PHP's release maintainers, looks at how opcode caches improve PHP script performance by caching the opcodes executed for a script. In particular, he examines how OPCache, the opcode cache bundled with PHP, handles opcode caching and activation, manages shared memory, reduces memory usage with strings, and locking mechanism. He ends with some advice on how to configure OPCache to get the best performance. http://jpauli.github.io/2015/03/05/opcache.html

Performance vs Scalability

Eli White

This month I want to touch on a more technical subject, though really it's a terminology subject. (So I guess this is now a linguistic column?) But I want to discuss the differences between the **Performance** of an application, versus the **Scalability** of said application. Because I keep finding far too often that these terms are being used interchangeably. Of course while they are related, they are definitely not the same thing.

Performance

To first touch on performance, I think that this definition is fairly well known. How performant your application is, means how quickly the results are returned to the end user. Alternatively, you may be looking at the machine itself, and how performant an application is is measured by how little CPU time it consumes.

In both cases, the measurement is really the same. It's based on how quickly & efficiently the data is processed from when execution of the program starts to when it ends. Of course, good performance can be achieved by having extremely efficient code, or via having an extremely powerful computer. In fact, deciding between spending more developer time to wring better performance out of a program versus just throwing more hardware at it is one of the oldest trade offs.

Scalability

On the subject of scalability, to say that an application is scalable is to say that it can handle a varying load of traffic. People talk about both scaling up to meet higher demand, and scaling down when the traffic subsides. The core goal of scalability is to have consistent performance no matter the level of traffic. Whether you have a dozen or a million users, page load and execution times should be unaffected.

This is typically handled by having resources that can be dynamically allocated. That is, spinning up additional servers, memory, disks, or networks on demand. This can be either an automated process or a manual one. Either way, the code itself is considered scalable if it supports this without need to update the code. If the code has been written so that it can only run on a single server, then it cannot be said to be scalable.

> *It's hardware that makes a machine fast. It's software that makes a fast machine slow.*
>
> *— Craig Bruce*

Combining the Concepts

The usual issue is that performance and scalability are linked and provide a yin-yang pull between them. Even if an application is written to be scalable, what if it doesn't perform well enough on some baseline of hardware? What if you need to essentially add a new server for every new user that signs onto the system? Then the application is not truly scalable.

Similarly, something can be written to be amazingly performant, but rely on tricks to get there; that simply makes scaling that performance out difficult to impossible.

In the end, you have to balance these. You want something that is performant enough on medium-size servers, and written in a scalable way that allows your application to expand to additional users and shrink back when it isn't in use. The terms will be ever intermingled. Unable to be separated when it comes to writing the code. But let's try to get the terminology correct when discussing it.

Eli White is the Managing Editor & Conference Chair for php[architect] and a Founding Partner of musketeers.me, LLC (php[architect]'s parent company). He's learning personally how to scale in real life, because as he ages, performance is going right down the tubes.

Borrowed this magazine?

Get php[**architect**] delivered to your doorstep or digitally every month!

Each issue of php[**architect**] magazine focuses on an important topic that PHP developers face every day.

We cover topics such as frameworks, security, ecommerce, databases, scalability, migration, API integration, devops, cloud services, business development, content management systems, and the PHP community.

php[**architect**]

Digital and Print+Digital Subscriptions Starting at $49/Year

http://phpa.me/mag_subscribe