February 28, 2020

From Legacy to Symfony

Sébastien Ballangé

ConFoo.CA
DEVELOPER CONFERENCE

paystone

A big rewrite without the“Big Rewrite”

But why?

- Replacing a dead framework
- Structuring a “we don’t need a framework” mess
- Hopefully, speedup development afterwards

Step 0

- Cover the existing app with automated tests, even basic ones
  - Unit tests (PhpUnit, Atoum, ...)
  - Integration/Acceptance tests (Selenium, Behat, Codeception, …)
- Measure/monitor usage of the current app
  - Find out if some parts are not used anymore and remove them

Actually...

Very small project?

forget it and rewrite the whole thing

“from scratch”

Bootstrap Symfony

- `cd my-existing-project/ && git checkout -b bootstrap-symfony && git push && cd -`
- `symfony new --full migration-project`
- `mv migration-project/* my-existing-project/`
- `cd my-existing-project/ && git add . && git commit`

Wrap existing code

- Use Symfony's public/index.php
- Use Symfony's router + legacy fallback controller bootstrapping old code

```php
<?php
namespace App\Controller;

class LegacyController extends AbstractController
{
    /**
     * @Route("/{action}", name="legacy_fallback", requirements={"action": ".+"})
     */
    public function fallback(ContainerInterface $container): Response
    {
        require_once __DIR__ . '/../../legacy/init.php';
        try {
            // Bootstrap the legacy app
            chdir(__DIR__ . '/../../public');
            $legacyApp = \MyCompany\MyApp\Core::bootstrap($container);
            ob_start();
            $legacyApp->dispatch();

            // wrap the content in a Symfony response
            return new Response(ob_get_clean());
        } catch (Zend_Controller_Dispatcher_Exception|Zend_Controller_Action_Exception $exception) {
            throw new NotFoundHttpException($exception->getMessage(), $exception);
        }
    }
}
```

```php
<?php
namespace App\Controller;

class LegacyController extends AbstractController
{
    /**
     * @Route("/{_locale}/{page}.php", name="legacy_fallback", requirements={"_locale":"en|fr|pt"})
     */
    public function fallback(Request $request, string $page): Response
    {
        $legacyPage = "{$request->getLocale()}/{$page}.php";

        chdir(__DIR__ . '/../../legacy');
        if (!$this->isValidPage($legacyPage)) {
            throw new NotFoundHttpException();
        }

        ob_start();
        include $legacyPage;
        $legacyResponse = ob_get_clean();

        // wrap the content in a Symfony response
        return new Response($legacyResponse);
    }
}
```

Dependencies Injection

- Make the container available in the legacy app
  - Only public services will be accessible
- Configure services from Symfony and use them from the legacy code when needed

```php
<?php
namespace MyCompany\MyApp;
use Psr\Container\ContainerInterface;

class Registry
{
   /** @var ContainerInterface */
   private static $container;

   public static function setContainer(ContainerInterface $container): void
   {
       self::$container = $container;
   }

   /** @deprecated Do not use outside of the legacy code */
   public static function getServiceForLegacy(string $serviceId)
   {
       @trigger_error('Avoid using getServiceForLegacy(), have your dependencies
injected instead of relying on the container', E_USER_DEPRECATED);

       return self::$container->get($serviceId);
   }
}
```

```php
// and then somewhere in the legacy application's bootstrap
// or in LegacyController::fallback()
Registry::setContainer($container);
```

Create bridges for common logic

- Translator
- Parsing of configuration files
- PDO connections or ORM
- ...

```php
<?php

namespace App\Legacy\Bridge;

class TranslatorBridge extends \Zend_Translate
{
   /** @var TranslatorInterface */
   private $translator;

   public function __construct()
   {
       // Prevent parent constructor from being called since Zend_Translate reads the file when instantiated
   }

   public function translate(string $messageId, array $parameters = [], $domain = null, $locale = null)
   {
       return $this->getTranslator()->trans($messageId, $parameters, $domain, $locale);
   }

   public function getTranslator(): \Symfony\Contracts\Translation\TranslatorInterface
   {
       if (!$this->translator) {
           $this->translator = \MyCompany\MyApp\Registry::getServiceForLegacy(TranslatorInterface::class);
       }

       return $this->translator;
   }
}
```

ORM / Doctrine

- Keep the existing one if it works for you

Or

- Switch to Doctrine or equivalent

Session + Authentication

- Configure Symfony to manage the Session
  - Port custom code if the existing app accesses `$_SESSION` using a custom storage mechanism
- Implement login/logout in Symfony controllers
- Convert existing access controls to Voters and Firewalls
  - Make sure both old and new pages can be handled

Expose Twig in legacy

- Existing pages might benefit from loading a Twig partial
  - UI elements (navigation, footer, …)
  - Loading JS/CSS assets (with or without Webpack Encore)

```php
<?php

namespace MyCompany\MyApp\View;
use Twig\Environment;

/** View helper to render a Twig template in legacy pages */
class TwigHelper extends \Zend_View_Helper_Abstract
{
   /** @var Environment $twig */
   private $twig;

   /** Render a Twig template */
   public function twig(string $template, array $context = []): string
   {
       return $this->getTwig()->render($template, $context);
   }

   public function getTwig(): Environment
   {
       if (!$this->twig) {
           $this->twig = \MyCompany\MyApp\Registry::getServiceForLegacy('twig');
       }

       return $this->twig;
   }
}
```

Expose Twig in legacy

```
// in application/layout/layout.phtml
<?= $this->twig('common/stylesheets.html.twig'); ?>
```

New Features

- Always develop brand new features in the new codebase
  - Decide of architectural changes now to avoid having to change everything again later
- When modifying existing features:
  - Either migrate before/during the development
  - Or code it in a “Symfony-friendly” way and migrate later

Steps 5 to 247

- Migrate one module/section at a time
  - Start with the simplest pages and gradually increase the complexity
  - Challenge the existing code
  - Cover the new code with automated tests
- Rinse and repeat

Continuous cleanup

- Remove old libraries when not in use anymore
  - Focus on completely removing specific parts of the legacy framework from time to time
- Remove the bridges and similar classes once they are not needed
- Remove entities/models from the old ORM
- Keep upgrading Symfony and others regularly

Celebrate!

```
$ composer remove shardj/zf1-future
```

Questions?

Sébastien Ballangé

@sballange

https://www.paystone.com/careers

https://bit.ly/confoo-legacy2symfony

https://github.com/confooca/yul2020-slides