GPNV

Antonio GIORDANO
Julien RICHOZ
Kevin JORDIL

SI-t1a
2018

Kevin Jordil
Julien Richoz
Antonio Giordani

Sommaire

Introduction

Dans le cadre de notre formation, nous sommes amenés à travailler sur un projet web. Durant un semestre nous avons travaillé sur le Framework Laravel.
Le but de ce module est de simuler un environnement de travail comme celui qu'on peut retrouver dans une entreprise. Nous avons un client et un chef de projet qui dirigent l'avancement du projet.
Dans l'exemple d'une entreprise que nous intégrons, nous devons reprendre en cours de route un projet et effectuer des améliorations/modifications sur ce dernier.

GPNV

GPNV est un outil de gestion de projet développé au sein du CPNV pour les besoins de l'école. Il permet de gérer les différentes étapes d'un projet, les éléments annexes ou encore les fichiers joints.
Le but du projet a été de reprendre GPNV afin de l'améliorer.

SAML

Qu'est-ce que SAML ?

SAML est l'abréviation de "Security Assertion Markup Language" qui propose une authentification unique, ce qui permet qu'un utilisateur puisse naviguer sur plusieurs sites en ayant qu'un seul identifiant et mot de passe.

L'utilité de SAML au sein du CPNV

Chaque année, des étudiants créent des outils et des logiciels pour le CPNV, mais chacun gère l'inscription à sa manière. Pour éviter ceci, SAML permet de toujours d'avoir le même login et mot de passe, à savoir celui du CPNV. Chaque application sera donc unifiée pour l'authentification.
Le serveur Saml de l'intranet certifie seulement que l'utilisateur a bien réussi à se logguer en nous fournissant certaines informations en retour. Notamment le prénom, nom, adresse e-mail, mais aussi les groupes de l'AD auquel il fait partie.

Intégration

Un paquet a été préparé pour être intégré directement dans un projet Laravel version 5.5 déjà existante. Il ajoute les routes en fonction des paramètres saisis dans le fichier de configuration correspondant. Des listeners permettent de définir ce qu'il se passe lorsqu'un utilisateur se logue ou se délogue.

Problèmes rencontrés

Après plusieurs recherches il a été décidé d'utiliser l'extension aacotroneo/laravel-saml2 présente sur Github (*avant que Github soit acheté par Microsoft*).
Le principal problème fut que cette extension utilise l'extension onelogin/php-saml et que cette extension utilise l'extension PHP mcrypt qui a été dépréciée dès la version 7.1 de PHP. Il a fallu comprendre pour quelles raisons l'extension SAML ne fonctionnait pas. Une fois le bug trouvé, il fallut attendre que les deux propriétaires des applications publient des branches compatibles avec les versions PHP plus récentes que 7.0.
En attendant, nous avons cloné le projet sur le Github de CPNV-ES et nous l'avons adapté pour récupérer les bons éléments. Une fois terminé, les propriétaires des packages avaient mis à jour leurs extensions.

Dans un deuxième temps, il fallait créer une adaptation SAML simple et complète qui soit facile à implémenter dans un projet Laravel existant. Nous avons donc dû comprendre comment l'authentification de SAML pouvait s'intégrer au mieux à Laravel tout en réutilisant ce qui existait déjà dans le Framework.

Installation de SAML

Prérequis

- Laravel 5.5
- Système d'authentification

Installation du package (SP)

Dans le composer.json de votre projet, ajouter ou adapter ceci :

```
"minimum-stability": "dev",
"prefer-stable": true
```

Une fois ajouté, enregistrer le fichier et connectez-vous sur le serveur via une invite de commande.
Exécuter ces commandes

```
composer require aacotroneo/laravel-saml2:dev-remove_mcrypt
```

```
composer update
```

Dans le dossier config de votre projet, ajouter saml2_settings.php
Dans le dossier Controllers de votre projet, ajouter SAMLController.php
Dans le dossier Middleware, ajouter le fichier SamlAuth.php
Dans le dossier app/Providers/EventServiceProvider.php, dans $listen, ajouter :

```
'Aacotroneo\Saml2\Events\Saml2LoginEvent' => [
    'App\Listeners\LogiginListener',
],
```

```
'Aacotroneo\Saml2\Events\Saml2LogoutEvent' => [
    'App\Listeners\LogoutListener',
],
```

Dans http/Kernel.php, dans routesMiddleware, ajouter
`'samlauth' => \App\Http\Middleware\SamlAuth::class,`
Dans app/Http/Middleware/VerifyCrsfToken.php, dans $except, ajouter `"saml2/acs",`

Configuration

Dans le fichier de configuration saml2_settings.php, les paramètres suivants peuvent être adaptés selon votre application :

- routesPrefix
- routesMiddleware
- logoutRoute
- loginRoute
- errorRoute

Configuration du serveur (IDP) (P. Hurni)

Une configuration côté serveur doit être réalisée. Vous devez fournir cette configuration à Monsieur Pascal Hurni en l'adaptant en fonction de votre application.

```
$metadata['http://laravel_url/routesPrefix/metadata'] = array(
    'AssertionConsumerService' => 'http://laravel_url/routesPrefix/acs',
    'SingleLogoutService' => 'http://laravel_url/routesPrefix/sls',
    'NameIDFormat' => 'urn:oasis:names:tc:SAML:2.0:nameid-
format:persistent',
    'simplesaml.nameidattribute' => 'uid'
);
```

Exemple de route

```
Route::group(['middleware' => ['samlauth']], function () {
    //protected routes go here
    Route::get('loggedin', function(){
        return "loggedin";
    });
    Route::get('authtest', function(){
        return "".\Auth::check();
    });

});
```

Dans cet exemple, certaines routes sont accessibles seulement si nous sommes logués avec SAML.

Capistrano

Capistrano est un utilitaire pour automatiser le déploiement d'applications web. Il automatise le processus de création d'une nouvelle version d'une application disponible sur un ou plusieurs serveurs Web, y compris le soutien des tâches telles que la modification des bases de données. Même si Capistrano est écrit en Ruby, il est facilement utilisable pour déployer des applications provenant de Rails aussi bien que d'autres langages/Framework tels que Laravel. Capistrano est implémenté principalement pour de l'utilisation en ligne de commande bash.

Pour le projet GPNV, la migration se faisait auparavant avec FileZilla en FTP. Nous avons donc utilisé Capistrano pour migrer l'application chez Swisscenter via SSH, ce qui permet un gain de temps majeur en automatisant le déploiement via un protocole de communication sécurisé.

Objectif

Déployer un projet Laravel sur l'hébergeur Swisscenter sous le nom de domaine www.gpnv.mycpnv.ch.

Déploiement de GPNV

Les fichiers de Capistrano sont déjà configurés pour GPNV. Une annexe expliquant la configuration pour la migration d'un projet Laravel avec Capistrano est disponible sous https://cpnv-es.github.io/capify_laravel_for_swisscenter.html.

Installation outil Capistrano

A faire une seule fois sur la machine de développement:

1. Installer ruby (avec chocolatey)
   `choco install ruby`
2. Installer Capistrano
   `gem install capistrano capistrano-laravel`.
3. Ajouter les clés publiques/privées dans le dossier config du projet.

Déploiement

1. Déployer depuis la machine dév, dans le répertoire de l'application:
   `cap production deploy`

Description fichiers Capistrano :

```
.
├── Capfile                 # Used to manage Capistrano and its dependencies
├── config
│   ├── deploy
│   │   ├── production.rb   # Configuration for production environment
│   │   └── staging.rb      # Configuration for staging environment
│   └── deploy.rb           # Common configuration for all environments
└── lib
    └── capistrano
        └── tasks           # Customized Capistrano tasks for your project
```

Récupération de version

Nous avons créé une tâche via Capistrano qui permet de récupérer et afficher automatiquement la dernière version du projet (via git). La tâche se retrouve dans le fichier `production.rb` et se nomme `get_version`

Problèmes rencontrés Capistrano

Les principaux problèmes rencontrés furent lors de la commande `cap production deploy`, soit le déploiement. Il y a eu de nombreux petits problèmes qui ont causé une grande perte de temps, surtout lors de la tâche composer:run, à laquelle nous attendions de longues minutes sans que rien ne se passe. L'aide de Mr. Hurni nous a permis de résoudre de nombreux problèmes, parmi lesquels :

- Composer install : impossibilité d'installer les dépendances => Ajout d'une indication dans le fichier `production.rb` permettant à php d'ouvrir des url.
- Problème avec l'utilisation de la commande readlink sur le serveur. Ajout d'une instruction dans le fichier `production.rb` pour y remédier.
- Version PHP du serveur : pas de possibilité de la définir via la configuration de Swisscenter => Sur le serveur, indication dans un fichier quelle version php utiliser.

La majorité des problèmes étaient liés au fait du manque d'options configurables chez Swisscenter. Pour y pallier, nous avons donné les instructions dans les fichiers de configuration du projet de Capistrano.

Problème restant

Lors du déploiement, les fichiers/images existants des projets sont réécrits et ne peuvent plus être affichés/supprimés. Il faut lors du déploiement de Capistrano, sauver les fichiers actuels.

Migration à Laravel 5.5

La version précédente de Laravel utilisée était la 5.3. Nous avons décidé de commencer par mettre à jour le projet vers la version 5.5 de Laravel. Nous verrons les modifications appliquées au projet afin de réaliser cette mise à jour.

Composer.json

Les dépendances du projet ont été modifiées comme suivant (commit : https://github.com/CPNV-ES/GPNV/commit/4f404b3641b693f13a7a77c01fbb2216ed966104)
Si l'ancienne version n'est pas présente, c'est un ajout.
Si la nouvelle version n'est pas présente, c'est une suppression.

Require :

Dépendance	Ancienne version	Nouvelle version
php	>= 5.5.9	>= 7.0.0
laravel/framework	5.2.*	5.5.*
illuminate/support	>= 5.0	5.5.*
illuminate/view	>= 5.0	5.5.*
laravelcollective/html	5.2.*	5.5
fidelopper/proxy		~3.3

Require-dev :

Dépendance	Ancienne version	Nouvelle version
mockery/mockery	0.9.*	~1.0
phpunit/phpunit	~4.0	~6.0
xethron/migrations-generator	dev-l5	
way/generators	dev-feature/laravel-five-stable	
user11001/eloquent-model-generator	~2.0	

Scripts :

post-root-package-install	"php -r \"copy('.env.example', '.env');\""
	"@php -r \"file_exists('.env') || copy('.env.example', '.env');\""
post-autoload-dump	

	"Illuminate\\Foundation\\ComposerScripts::postAutoloadDump", "@php artisan package:discover"

Repositories :
{
"type": "git",
"url": "git@github.com:jamisonvalenta/Laravel-4-Generators.git"
}

Routes

Modification du fichier ‘’app/Providers/RouteServiceProvider.php’’

Ancien code	Nouveau code
use Illuminate\Routing\Router;	use Illuminate\Support\Facades\Route;

public function map(Router $router)
{
$router->group(['namespace' => $this->namespace], function ($router) {
require app_path('Http/routes.php');
});
}
	public function map()
	{
	$this->mapWebRoutes();
	//
	}

	protected function mapWebRoutes()
	{
	Route::middleware('web')
	->namespace($this->namespace)
	->group(base_path('routes/web.php'));
	}

Ajout des fichiers suivant :

- routes/api.php
- routes/channels.php
- routes/console.php

Déplacement du fichier ‘’app/Http/routes.php’’ dans ‘’routes/web.php’’

Providers.php

Modification des fichiers comme suivant :

Fichier	Code
app/Providers/EventServiceProvider.php	
Remplacer :	`public function boot(DispatcherContract $events) { parent::boot($events); }`
Par :	`public function boot() { parent::boot(); }`
app/Providers/RouteServiceProvider.php	
Remplacer :	`public function boot(Router $router) { parent::boot($router); }`
Par :	`public function boot() { parent::boot(); }`

Fichier .env.exemple

Ajout des lignes suivantes dans le fichier “.env.exemple” :

- APP_NAME=GPNV
- DB_CONNECTION=mysql
- DB_PORT=3306

Bootstrap

Suppression du fichier ‘’bootstrap/autoload.php’’.
Modification du fichier ‘’public/index.php’’ :
Remplacer : `require __DIR__.'/../bootstrap/autoload.php';`
par : `require __DIR__.'/../bootstrap/autoload.php';`

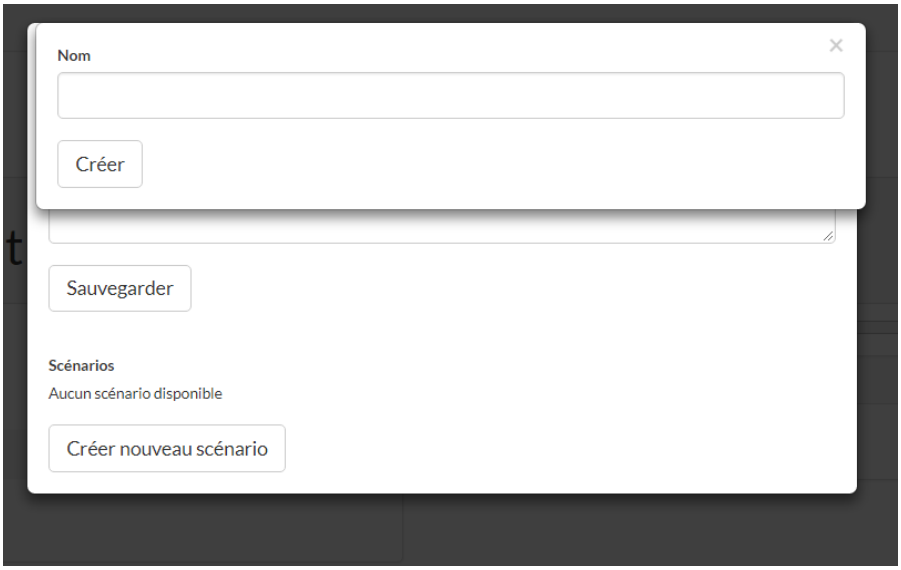
Réusinage du code

Une fois SAML intégré, le déploiement fonctionnel et la migration vers Laravel 5.5 effectuée, restait le développement de GPNV. L'objectif principal était un redesign complet, tant au niveau fonctionnel que visuel.

Interface utilisateur

Après une analyse du site, l'aspect aussi bien ergonomique que visuel était à revoir. En effet, il était pénible et fastidieux d'arriver sur une page désirée. Nous devions affronter moult modals pop-up et diverses péripéties le long du chemin, tel que l'infâme triple modal pop-up.

Double Modal pop-up



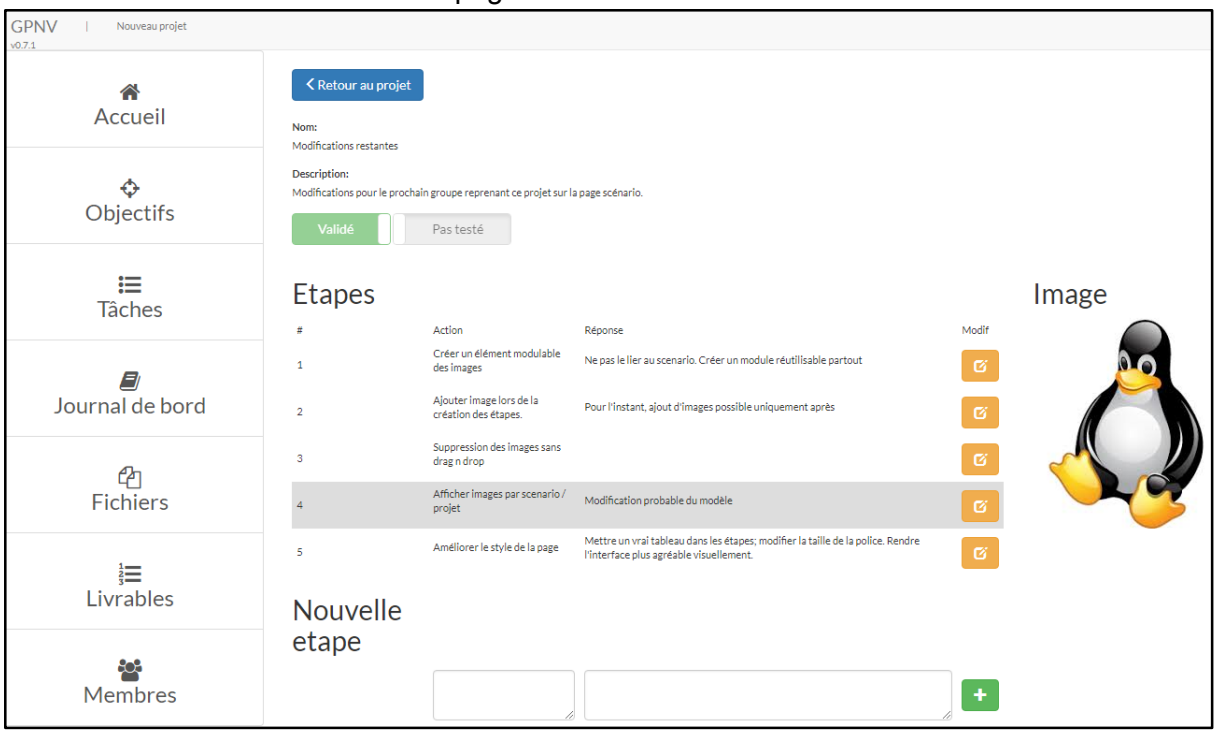
Les désavantages majeurs des modals pop-up sont qu'il n'est pas possible de les ouvrir dans une autre fenêtre du moteur de recherche et qu'il n'est pas possible d'effectuer quelque autre action qu'il soit sur l'application pendant l'affichage du pop-up. De plus, cette fenêtre peut cacher des informations qui peuvent être nécessaires à l'utilisateur lors de la saisie (par exemple : remplissage d'un formulaire de vacances via modal pop-up, mais qui cache la page de description des vacances dont la date peut-être. L'utilisateur doit alors quitter le formulaire pour vérifier la date et la retenir/noter ailleurs).
Les modals pop-up sont donc à éviter le plus possible, bien que dans certains cas il peut être légitime de les utiliser, tel qu'un message de confirmation pour supprimer un élément.

Un deuxième élément important à modifier fut l'ergonomie du site. Il était par exemple compliqué de se rendre sur la page *Scenario* d'un objectif, puisque nous ne savions pas vraiment où cliquer pour y accéder, sans compter les multiples fenêtres modales à braver. Nous avons donc essayé d'appliquer la règle des 3 clics, qui consiste à ce qu'un utilisateur atteigne n'importe quelle information en 3 clics maximum, mais sans toutefois respecter le caractère absolu de cette règle. Cela pourrait avoir comme conséquence des pages extrêmement fournies et remplies de tout lien en fonction de la complexité du site.

Pour offrir une meilleure interface utilisateur, nous avons donc découpé le site en plusieurs pages accessible via un menu. Auparavant, tous les éléments étaient réunis sur une page, ce qui rendait la navigation désagréable une fois tous les éléments ouverts.

Ancien GPNV

Désormais, un menu a été intégré sur la gauche, permettant à l'utilisateur de naviguer aisément entre les différentes pages et d'obtenir une meilleure vision d'ensemble.



Page Scénario

Contrôleurs

Lors de la récupération du projet, la plupart des fonctions étaient présentes dans un seul controller. La plupart des contrôleurs n'étaient pas utilisés comme CRUD.
Notre mission fut de séparer le contenu du contrôleur dans plusieurs d'entre eux. La mission n'est pas terminée, il reste des pages à extraire du contrôleur principal.

Le souci majeur dans cette configuration est que lorsque nous changeons quelque chose, il faut tout tester derrière car tout est lié d'une manière ou d'une autre. C'est donc peu modulable.

Refonte Page Scénario

La refonte de la page scénario réunit les deux aspects vus précédemment, c'est-à-dire une amélioration de l'interface utilisateur ainsi qu'une refonte du modèle.
Le problème majeur de l'interface scénario était la quantité de formulaires actifs en même temps, sans vraiment savoir ce que l'on modifiait/sauvegardait lors de l'enregistrement. Nous avons ainsi découpé cette page - tant au niveau visuel que conceptuel - afin d'offrir une interface plus agréable à l'utilisateur. Toutefois, quelques améliorations restent à effectuer que nous verrons par la suite.

Problèmes rencontrés

Les principaux problèmes rencontrés sont les mêmes que ceux énoncés dans la partie Problèmes généraux rencontrés.
La plus grande difficulté se trouvait cependant dans la reprise/transformation du code javascript, notamment pour les requêtes Ajax. Nous avons alors créé un contrôleur pour les étapes du scénario afin de faciliter le traitement en Ajax et de rendre la structure plus claire.

Refonte objectifs - livrables (checklist)

La ressource checklist a été partiellement supprimée, à la place nous avons créé une ressource pour les objectifs et une pour les livrables.

Les routes ont été modifiées pour correspondre aux modifications ci-dessous.

Create

L'ancienne blade checklist.create permettait de créer une nouvelle checklist. Cette création a été déplacée dans la création de livrables (deliverable.create) et d'objectifs (objective.create), ces deux créations utilisent la fonction store du controller ''checklist''. (ChecklistController@store)

Show

L'ancienne blade checklist.show permettait d'afficher les éléments d'une checklist (tout les livrables par exemple).
Cette blade a été adaptée et se retrouve maintenant dans les livrables et objectifs (deliverable.show / objective.show), ils utilisent la fonction show de leur controller respectif. (...Controller@show)

ShowItem

L'ancienne blade checklist.showItem était liée uniquement aux objectifs (afin d'afficher les scénarios liés à un objectif). La blade se retrouve maintenant dans objectif (objective.showItem) qui utilise la fonction ShowItem de son contrôleur (ObjectiveController@showItem)

Javascript

Presque l'intégralité du fichier ''checklist.js'' a été enlevé. Le code qui s'y trouvait a été transféré dans ''objectives.js'' et ''deliverables.js''

CSS

Des fichiers CSS ont été créer pour les ressources objectifs et livrables afin d'enlever le style présent dans le code HTML.

Améliorations restantes

Deux améliorations majeures et une mineure restent :
Améliorations majeures :

- Créer la partie image en un module externe que l'on peut implémenter indépendamment à chaque page, en choisissant l'affichage d'images par scénario ou par projet.
- Améliorer l'ergonomie de l'étape des scénarios, notamment concernant les images correspondantes. Également pouvoir ajouter une image à une étape directement lors de l'étape de création d'une nouvelle étape.

Amélioration mineure :

- Améliorer le style de la page (css) : plus grande police des éléments textuels.
- Améliorer le tableau des steps (html).

Problèmes généraux rencontrés

Le problème majeur rencontré fut la reprise d'un projet en cours. Nous avons passé beaucoup de temps à le comprendre, analyser son fonctionnement, jusqu'à y apporter des modifications. Le projet était codé assez "groupé", et lors d'un changement dans un fichier cela impactait rapidement l'ensemble du projet. Il fallut donc tout découper en modules plus indépendants afin de faciliter toutes modifications ultérieures.

Conclusion

La reprise d'un projet nous a permis de nous sensibiliser sur la qualité de l'écriture de code. Travailler sur un projet existant nécessite d'abord de l'analyser afin de l'améliorer. Ainsi, un code clair, commenté, structuré et respectant le plus possible un quelconque design pattern (MVC), permet une reprise de projet d'autant plus rapide et efficace.

Pour GPNV, la première partie consistait à mettre à jour le projet vers des technologies plus récentes/à jour, à savoir la migration vers Laravel 5.5, un déploiement automatisé via Capistrano vers Swisscenter et l'intégration d'un système d'authentification du CPNV via SAML. Les objectifs fixés de cette partie ont été atteints.
La deuxième partie consistait à améliorer l'interface utilisateur et modifier le code afin de respecter une architecture plus proche du MVC. Il reste toutefois quelques tâches à améliorer/finaliser, mais l'interface utilisateur est plus intuitive qu'auparavant et le code plus compréhensible.

L'implémentation de ces fonctionnalités permet ainsi aux prochains développeurs reprenant le projet un gain de temps majeur dans la maintenance de l'application, à savoir de l'analyse du code jusqu'à son déploiement en passant par le développement.