

Développer des applications business en PHP avec le framework **SILEX**

Version 1.4
2 Oct. 2016

Table des matières

Table des matières	2
1. Le framework SILEX	7
1.1. Introduction.....	7
1.2. Configuration préalable	8
2. Premier projet avec SILEX.....	9
2.1. Notions préalables	9
2.2. Installation du framework	10
2.3. Importation du projet dans un IDE (Netbeans).....	14
2.4. Initialisation d'un dépôt Git.....	16
2.5. Sécurité des répertoires de l'application	17
2.6. Installation des frameworks Bootstrap (Twitter) et jQuery	22
2.7. Finalisation de l'environnement.....	23
3. Anatomie d'un projet SILEX.....	24
3.1. Notions d'architecture, et de MVC	24
3.2. Les Vues et le templating (avec Twig).....	25
3.3. Les Contrôleurs	33
3.4. Vous reprendrez bien un peu de Twig.....	38
3.5. Des menus au look de Bootstrap	43
3.6. Conclusion provisoire	47
4. Approfondissements autour de SILEX.....	48
4.1. Les formulaires.....	48
4.2. Le CRUD.....	62
4.3. Refactoring du code.....	80
4.4. Traduction.....	87
4.5. Les Modèles	93
4.6. Sessions	104
4.7. Tests unitaires	104
5. Les bonus	105
5.1. Pagination et tri dynamique des colonnes.....	105
5.1.1 Pagination	105
5.1.2 Tri dynamique des colonnes.....	108
5.1.3 Pagination combinée avec des Filtres de sélection.....	112

5.2. Graphiques avec Highcharts	113
5.3. Export PDF	122
5.4. Export XML	130
5.5. Export JSON	133
5.5. Export CSV.....	134
6. ANNEXES.....	137
6.1. Le plugin Datepicker	137
6.2. Différences de version concernant les formulaires	140
6.3. Versions des composants PHP utilisés.....	141
6.4. Paginer avec classe	142

Présentation de l'auteur :

Grégory Jarrige est un développeur indépendant qui a fait ses premières armes de programmeurs dans les années 80 sur un Commodore 64, vers l'âge de 12 ou 13 ans. Devenu programmeur professionnel sur mini-système propriétaire, au début des années 90, il a dès lors développé des applications de gestion dans différents secteurs d'activité. Spécialiste du SQL et des traitements transactionnels, il a commencé à se former au développement web à partir de 2006, et a obtenu la certification Zend d'expert PHP en février 2010. Passionné par le développement logiciel, et en particulier par le langage Javascript, il développe actuellement sur le framework PHP Phalcon, ainsi que sur le framework Javascript AngularJS. Il dispense régulièrement des cours au développement web, aussi bien en entreprise que dans le cadre de formation continue.

Au sujet de ce support :

Ce document est un support de cours que l'auteur a créé pour répondre à une problématique qui lui avait été soumise par certains organismes de formation : former des adultes en formation continue (ou en reconversion) à l'utilisation d'un framework de développement, et cela en un temps très court (de 3 à 4 jours). Pour répondre à ce besoin, l'auteur s'est efforcé de rédiger un support que tout développeur débutant devrait pouvoir suivre de manière autonome, avec le moins d'intervention possible de la part du formateur. L'auteur est convaincu que cette méthode est la plus efficace pour que l'apprenant puisse comprendre et mémoriser ce qu'il fait.

Prérequis :

Pour être en mesure de suivre ce support de cours dans de bonnes conditions, l'apprenant doit posséder les connaissances suivantes :

- Une bonne connaissance du langage PHP, et notamment de la programmation objet en PHP
- Une bonne connaissance de la norme HTML, les mécanismes en jeu dans un formulaire HTML doivent être connus et compris
- Une bonne compréhension des mécanismes en jeu au sein du protocole HTTP. A minima, l'apprenant doit savoir distinguer une requête de type GET d'une requête de type POST, et savoir dans quel cas utiliser l'une ou l'autre
- Quelques connaissances minimales en CSS peuvent aider l'apprenant, mais sans plus, sachant que l'on utilisera le framework CSS Bootstrap (de Twitter) dans le cadre de ce cours
- Quelques connaissances en SQL sont indispensables, sachant que ce cours n'est pas un cours sur SQL et qu'aucune explication ne sera fournie sur la manière de créer une table dans une base comme MySQL par exemple.
- Quelques connaissances en Javascript peuvent aider l'apprenant, mais ce sujet ne sera qu'effleuré dans le cadre de ce support de cours.

- Une connaissance minimale de Git est souhaitable. L'auteur insiste beaucoup sur l'importance de « commiter » après chaque modification importante, l'apprenant pouvant ainsi profiter de l'apprentissage de SILEX pour se familiariser avec Git. La non connaissance de Git n'est cependant pas rédhibitoire, l'apprenant pouvant laisser ce sujet de côté pour se focaliser sur l'apprentissage de SILEX.
- Le choix de l'environnement PHP (Wamp, Xamp, Mamp) est laissé à la libre appréciation de l'apprenant. Le point important étant qu'il se sente à l'aise avec l'environnement qu'il utilise.

Avertissements : Certains chapitres de ce support ne sont pas terminés. Ils contiennent la mention « TODO » pour les retrouver plus facilement.

Droits d'auteur :

Ce document est publié sous la Licence Creative Commons n° 6 : BY SA

Pour de plus amples informations sur les termes de cette licence, prière de vous reporter au site officiel Creative Commons :

<http://creativecommons.fr/licences/>

Le framework SILEX est un projet publié sous la licence MIT.

L'auteur du présent support n'entretient pas de rapport particulier avec les créateurs de SILEX. Les choix techniques et les orientations prises dans le cadre de ce support, quand elles divergent des solutions proposées dans la documentation officielle de SILEX, sont de la seule responsabilité de l'auteur du présent support.

Tous les codes sources qui sont proposés dans ce support, et qui ne font pas partie de SILEX, sont publiés sous licence MIT. Les projets connexes (Bootstrap, JQuery, Highcharts, etc...) utilisés dans le cadre de ce support ont des licences spécifiques. Il appartient à chacun de s'assurer que les licences des différents composants utilisés ici sont compatibles avec l'usage qu'il souhaite en faire.

Le code source relatif au projet développé dans le cadre de ce support est disponible sur Github :

<https://github.com/gregja/firstsilex>

Blog de l'auteur :

<http://gregphplab.com>

Tweet de l'auteur :

https://twitter.com/greg_devlab

Donne un poisson à un homme, il fait un repas

Apprends lui à pêcher, il mangera toute sa vie

Confucius

"It's not at all important to get it right the first time.

It's vitally important to get it right the last time."

Andrew Hunt and David Thomas

1. Le framework SILEX

1.1. Introduction

Ce document présente l'utilisation de SILEX, pour le développement de projets webs :

<http://silex.sensiolabs.org>

SILEX est un framework PHP que l'on « range » dans la catégorie des microframeworks.

Le terme « microframework » est utilisé pour qualifier des frameworks de développement web légers, minimalistes. Ceci en contraste avec les frameworks dits « full-stack », donc plus « lourds », que sont par exemple Symfony (SF), Laravel ou Zend Framework (ZF).

Voici quelques exemples de frameworks que l'on range dans la catégorie des microframeworks :

1. Lumen, qui est basé sur Laravel :
 - <https://lumen.laravel.com>
2. Slim, projet indépendant mais qui utilise quelques composants de SF :
 - <http://www.slimframework.com>
3. Flight, qui est encore peu connu :
 - <http://flightphp.com>
4. Expressive, qui s'appuie sur ZF :
 - <https://zendframework.github.io/zend-expressive/>

Le terme « microframework » est sensé désigner des frameworks « légers », embarquant le strict nécessaire (en termes de code) pour couvrir les besoins de projets de moyenne ampleur (comme on en trouve souvent dans les PME).

Le microframework SILEX est un cas à part, car il embarque la quasi-totalité du framework Symfony (SF), tout en simplifiant l'utilisation de ce dernier. Il peut donc couvrir les besoins de projets de tailles très différentes, et surtout il est idéal pour s'initier aux principes de SF, en vue d'acquérir une solide maîtrise de ce framework par la suite. Il faut souligner que SILEX est développé par la même équipe qui développe SF, à savoir la société SensioLabs.

1.2. Configuration préalable

Pour suivre ce support dans les meilleures conditions, il est indispensable d'avoir installé sur le poste de développement, en complément d'un stack LAMP ou WAMP, les outils suivants :

- Le gestionnaire de versions Git : <https://git-scm.com>
- Le gestionnaire de dépendances Composer : <https://getcomposer.org>

La procédure d'installation de ces outils varie selon le système d'exploitation utilisé sur le poste de développement, on se reportera à leurs sites officiels respectifs pour de plus amples informations sur ce sujet.

2. Premier projet avec SILEX

2.1. Notions préalables

Il existe plusieurs méthodes pour démarrer un projet SILEX :

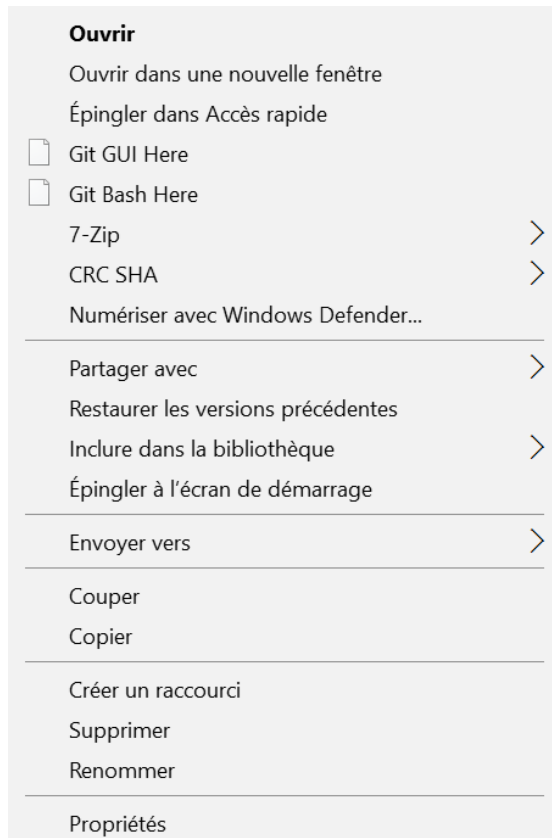
1. On peut par exemple créer un projet vide, et utiliser Composer pour « embarquer » SILEX à l'intérieur du projet. On pourra dans ce cas utiliser le composant « silex/silex » comme expliqué sur la page suivante :
 - <https://packagist.org/packages/silex/silex>
2. On peut télécharger un zip du projet et l'installer dans son répertoire « www » ou « htdocs », comme expliqué sur la page suivante :
 - <http://silex.sensiolabs.org/download>
3. On peut enfin utiliser une architecture « clés en main » proposée par SensioLabs, téléchargeable à l'adresse suivante :
 - <https://github.com/fabpot/Silex-Skeleton>

Nous n'avons pas encore étudié en détail le fonctionnement de Composer, aussi nous écarterons la méthode n° 1 pour l'instant. La méthode n° 3 est très pratique, aussi nous allons l'utiliser en priorité. Si lors de l'installation nous nous apercevons que nous sommes bloqués par des problèmes de proxy, nous utiliserons la méthode n° 2 comme méthode de secours.

NB : Pour les besoins du projet, nous utiliserons fréquemment la ligne de commande, et en particulier la ligne de commande de Git (Git Bash) qui est très pratique. Sous Windows, on rappelle qu'elle est facilement accessible, via un clic droit effectué sur n'importe quel répertoire.

2.2. Installation du framework

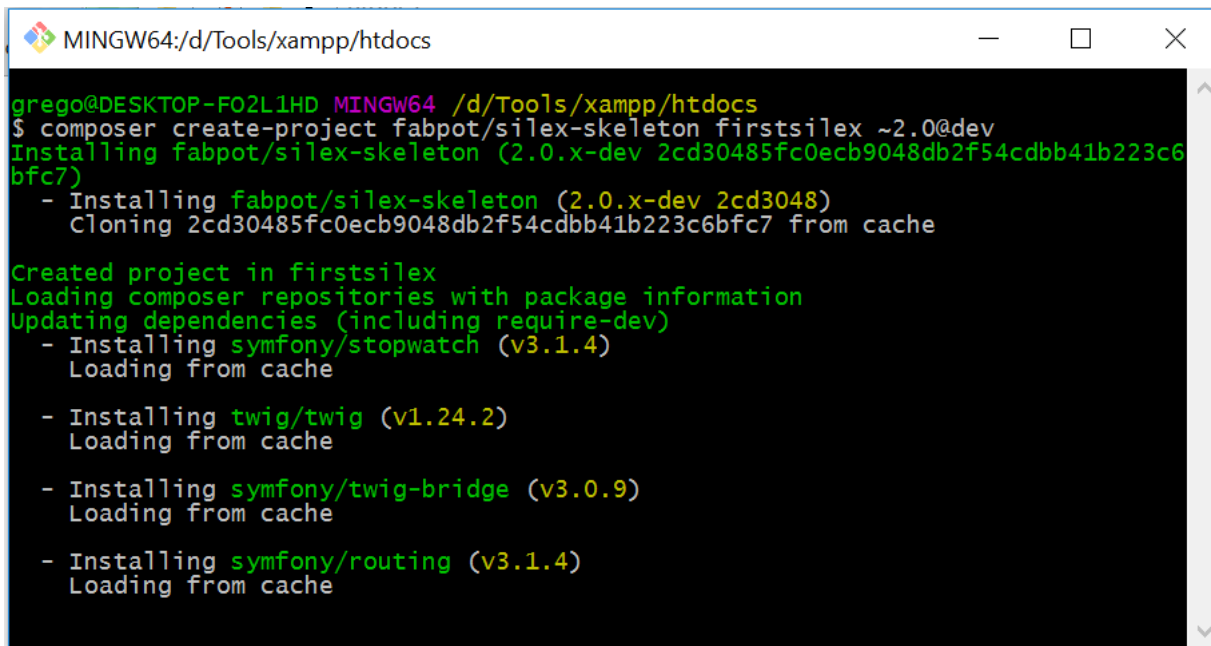
Placez-vous sur le répertoire « web » de votre stack PHP. Il s'agit du répertoire « htdocs » (pour XAMPP) ou « www » (pour WAMP). Faites un clic droit et sélectionnez l'option « Git Bash » :



AVERTISSEMENT : pour la suite de ce cours, nous supposons que le projet que nous souhaitons créer se trouve dans le répertoire « firstsilex ». Vous avez tout à fait le droit de choisir un autre nom pour votre répertoire, mais privilégiez un nom court, de préférence en minuscules, sans tirets, sans espaces, sans accents ni caractères exotiques (le seul caractère « exotique » tolérable étant l'underscore _).

A partir de « Git Bash », et de votre répertoire « www » ou « htdocs », saisissez la commande suivante :

```
composer create-project fabpot/silex-skeleton firstsilex ~2.0@dev
```



```
MINGW64:/d/Tools/xampp/htdocs
grego@DESKTOP-FO2L1HD MINGW64 /d/Tools/xampp/htdocs
$ composer create-project fabpot/silex-skeleton firstsilex ~2.0@dev
Installing fabpot/silex-skeleton (2.0.x-dev 2cd30485fc0ecb9048db2f54cddb41b223c6bfc7)
- Installing fabpot/silex-skeleton (2.0.x-dev 2cd3048)
  Cloning 2cd30485fc0ecb9048db2f54cddb41b223c6bfc7 from cache

Created project in firstsilex
Loading composer repositories with package information
Updating dependencies (including require-dev)
- Installing symfony/stopwatch (v3.1.4)
  Loading from cache
- Installing twig/twig (v1.24.2)
  Loading from cache
- Installing symfony/twig-bridge (v3.0.9)
  Loading from cache
- Installing symfony/routing (v3.1.4)
  Loading from cache
```

... Composer va installer un certain nombre de composants dont la plupart sont des composants provenant de Symfony (SF).

Une fois l'installation des composants terminée, l'écran ci-dessous devrait s'afficher, répondez « Y » à la réponse qui vous est posée :

```

MINGW64:/d/Tools/xampp/htdocs
monolog/monolog suggests installing ruflin/elastica (Allow sending log messages
to an Elastic Search server)
monolog/monolog suggests installing sentry/sentry (Allow sending log messages to
a Sentry server)
symfony/security suggests installing symfony/expression-language (For using the
expression voter)
symfony/security suggests installing symfony/ldap (For using the LDAP user and a
uthentication providers)
symfony/translation suggests installing symfony/yaml ()
symfony/validator suggests installing doctrine/annotations (For using the annota
tion mapping. You will also need doctrine/cache.)
symfony/validator suggests installing doctrine/cache (For using the default cach
ed annotation reader and metadata cache.)
symfony/validator suggests installing egulias/email-validator (Strict (RFC compl
iant) email validation)
symfony/validator suggests installing symfony/expression-language (For using the
Expression validator)
symfony/validator suggests installing symfony/yaml ()
Writing lock file
Generating autoload files
Do you want to remove the existing VCS (.git, .svn..) history? [Y,n]? Y
grego@DESKTOP-F02L1HD MINGW64 /d/Tools/xampp/htdocs
$

```

Pourquoi a-t-on répondu « Yes », à Composer qui nous demandait si l'on souhaitait conserver l'historique du projet ? Parce que cet historique ne nous intéresse pas (il nous intéresserait uniquement si l'on souhaitait participer à l'évolution du projet SILEX). En ce qui nous concerne, nous souhaitons utiliser SILEX pour contruire notre propre projet, c'est donc notre propre historique Git (que nous initialiserons dans un instant) qui nous intéresse.

On peut vérifier tout de suite le bon fonctionnement de notre application via le serveur web intégré à PHP, donc sans lancer WampServer (ou Xamp). Pour cela il suffit d'aller, en mode ligne de commande, dans le répertoire du projet et de taper la ligne de commande suivante :

```
php -S 0.0.0.0:8080 -t web web/index.php
```

```

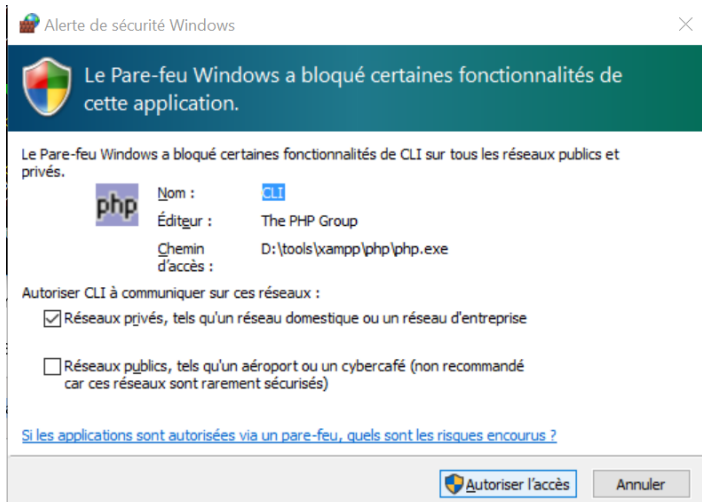
grego@DESKTOP-F02L1HD MINGW64 /d/Tools/xampp/htdocs/firstsilex
$ php -S 0.0.0.0:8080 -t web web/index.php
PHP 5.6.23 Development Server started at Fri Sep 09 07:31:22 2016
Listening on http://0.0.0.0:8080
Document root is D:\Tools\xampp\htdocs\firstsilex\web
Press Ctrl-C to quit.
[Fri Sep 09 07:32:37 2016] 127.0.0.1:62858 Invalid request (Unexpected EOF)
[Fri Sep 09 07:34:11 2016] 127.0.0.1:62866 Invalid request (Unexpected EOF)

```

On notera ici que l'on a utilisé le port 8080, afin de ne pas risquer d'entrer en conflit avec l'Apache de notre WampServer (ou de notre Xampp), qui est peut être déjà démarré, et qui lui « surveille » le port 80 par défaut.

Comme indiqué dans la copie d'écran ci-dessus, vous pouvez à tout moment arrêter le serveur intégré de PHP avec Ctrl-C.

NB : La première fois qu'on lance ce serveur PHP intégré sous Windows, on obtient le message d'avertissement ci-dessous, il faut alors cliquer sur « autoriser l'accès ».



Saisissez maintenant l'URL ci-dessous dans le navigateur :

<http://localhost:8080/>

Vous devriez voir apparaître le message suivant dans votre navigateur :

Welcome to your new Silex Application!

Vous pouvez également arrêter le serveur intégré de PHP (avec Ctrl+C), démarrer Apache, et tester le même projet avec l'URL suivante :

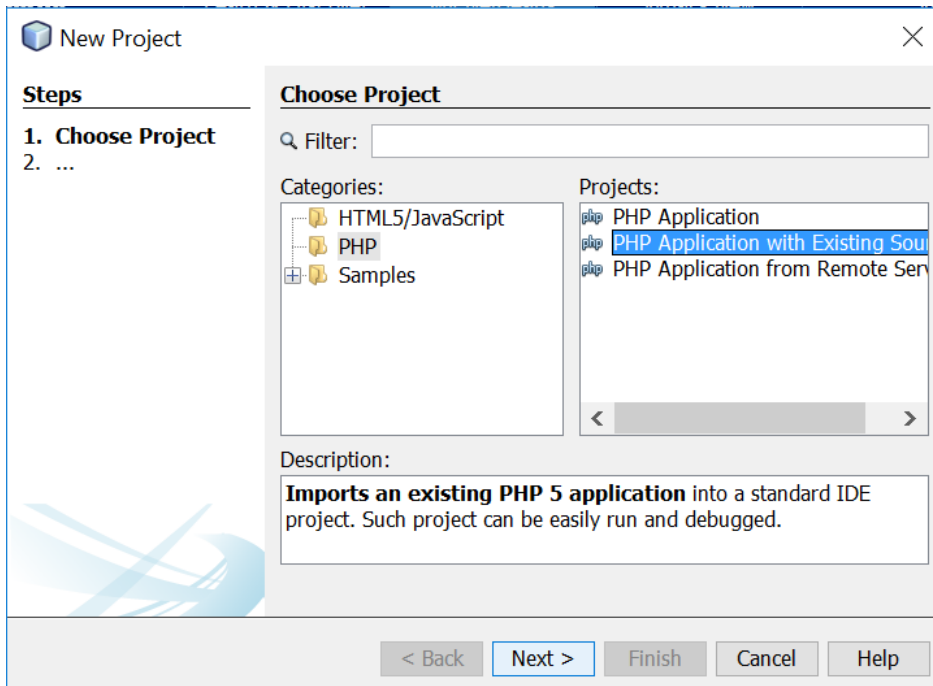
<http://localhost/firstsilex/web/>

Vous devriez voir apparaître le même message :

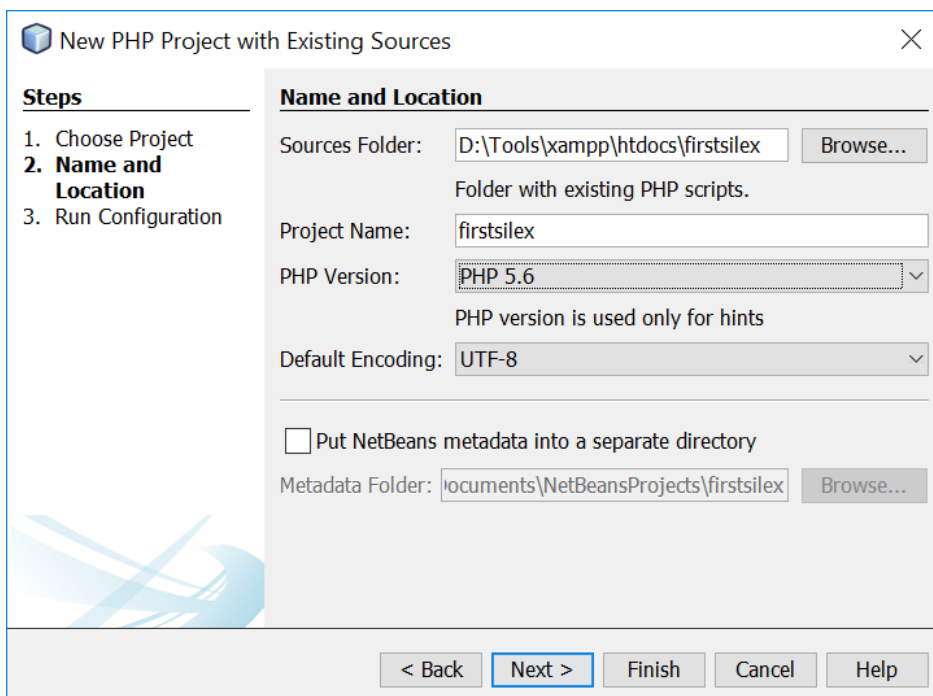
Welcome to your new Silex Application!

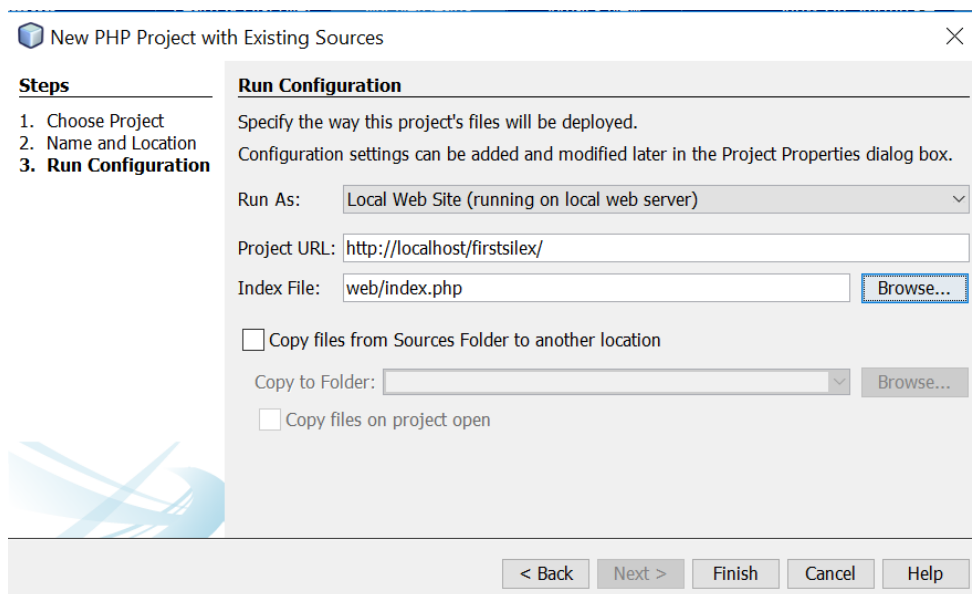
2.3. Importation du projet dans un IDE (Netbeans)

A ce stade, le projet est opérationnel, nous pouvons l'importer dans un IDE, par exemple Netbeans :

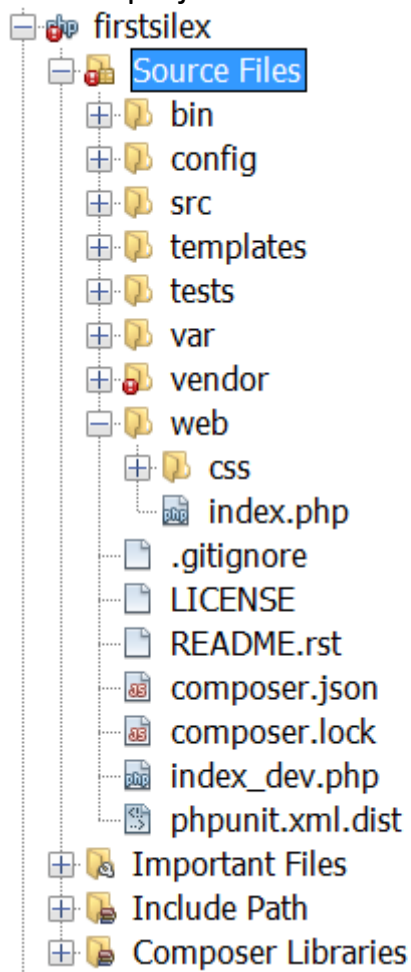


Pensez à sélectionner la version 5.6 de PHP, et vérifiez que l'encodage est bien UTF-8 :





Voici le projet sous Netbeans :



2.4. Initialisation d'un dépôt Git

Avant d'effectuer toute modification, nous allons initialiser notre projet en tant que dépôt Git, de façon à disposer d'un historique complet de toutes les modifications que nous allons effectuer à partir de maintenant.

Pour initialiser un dépôt Git, notamment si vous êtes sous Windows, le plus rapide consiste à faire un clic droit sur le répertoire du projet, puis sélectionner l'option « git bash », pour obtenir la ligne de commande Git. A partir de là, il vous suffit de saisir la commande ci-dessous pour initialiser un dépôt git à l'intérieur du projet PHP :

```
git init
```

Penser à compléter le fichier `.gitignore` pour éviter que Git ne surveille inutilement des fichiers de configuration propres à Netbeans ou à d'autres IDE comme par exemple Eclipse :

```
1 /vendor/  
2 /nbproject/  
3 .settings  
4 .buildpath  
5 .project
```

Pensez à faire un « git add . » et un « git commit » pour enregistrer la configuration du fichier `.gitignore` (comme ça on n'en parle plus).

Rappel sur Git : si Git vient tout juste d'être installé sur votre poste de développement, il n'a peut être pas été complètement configuré. Si ce n'est pas déjà fait, pensez à faire un :
















```
git config --global user.name "twitty and sylvester"  
git config --global user.email "twitty_sylvester@warnerbros.com"
```


2.5. Sécurité des répertoires de l'application

Peut être aurez-vous remarqué qu'avec le projet tournant sous Apache, si l'on utilise l'URL suivante ci-dessous, on obtient l'affichage suivant :

<http://localhost/firstsilex>

Index of /firstsilex

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 Parent Directory		-	
 LICENSE	2016-09-04 22:13	1.1K	
 bin/	2016-09-04 22:13	-	
 composer.json	2016-09-04 22:13	1.3K	
 composer.lock	2016-09-04 22:14	83K	
 config/	2016-09-04 22:13	-	
 index_dev.php	2016-09-04 22:13	772	
 nbproject/	2016-09-05 06:34	-	
 phpunit.xml.dist	2016-09-04 22:13	771	
 src/	2016-09-04 22:13	-	
 templates/	2016-09-04 22:13	-	
 tests/	2016-09-04 22:13	-	
 var/	2016-09-04 22:13	-	
 vendor/	2016-09-04 22:14	-	
 web/	2016-09-04 22:13	-	

Apache/2.4.17 (Win32) OpenSSL/1.0.2d PHP/5.6.23 Server at localhost Port 80

C'est assez ennuyeux de voir un tel contenu apparaître en clair, cela pourrait donner de mauvaises idées à certains hackers. Il faut savoir qu'en règle générale, les serveurs web de production sont configurés pour ne pas afficher le contenu d'un répertoire, comme vient de le faire Apache sur votre environnement de

développement. Nous étudierons plus tard la manière dont on peut sécuriser notre application, pour l'instant nous conservons le paramétrage par défaut d'Apache, après tout nous sommes sur un environnement de développement ☺.

Nous allons en revanche ajouter un fichier de configuration Apache dans le sous-répertoire « web » de notre application. Il s'agit d'un fichier texte, que je vous propose de créer via votre IDE (Netbeans ou autre) en lui donnant le nom suivant : `.htaccess`. Pour cela, utilisez l'option « créer un fichier », puis l'option « empty file » ou « fichier vide », et saisissez le nom `.htaccess` avant de valider. Ensuite, saisissez dans ce fichier `.htaccess` le code suivant :

```
<IfModule mod_rewrite.c>
    Options -MultiViews
    RewriteEngine On
    RewriteBase /firstsilex/web/
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^ index.php [QSA,L]
</IfModule>
```

Ce fichier `.htaccess` a pour effet de renvoyer toutes les requêtes http qui sont transmises à l'application à l'intérieur du répertoire « web », vers le script `index.php` qui se trouve dans ce même répertoire. C'est la raison pour laquelle les requêtes HTTP suivantes fonctionnent :

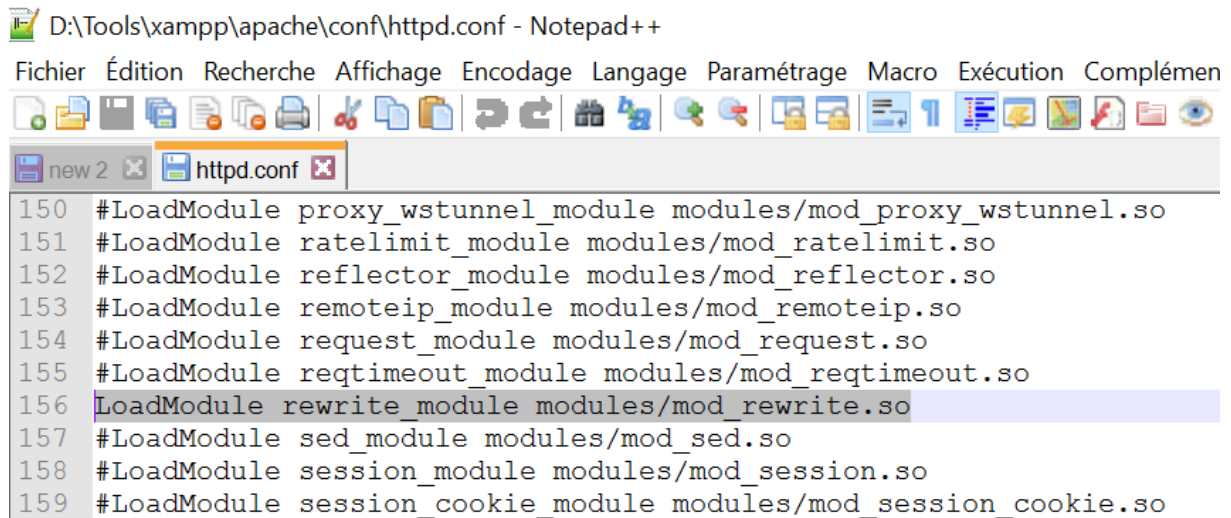
<http://localhost/firstsilex/web>
<http://localhost/firstsilex/web/index.php>
<http://localhost/firstsilex/web/index.php?xx>

... mais que les requêtes suivantes ne fonctionnent pas :

<http://localhost/firstsilex>
<http://localhost/firstsilex/xx>
<http://localhost/firstsilex/web/x>

Attention, le fichier `.htaccess` exploite un mécanisme d'Apache appelé « url rewriting » (réécriture d'URL). Pour que ce mécanisme fonctionne, il faut que le module Apache correspondant à cette fonctionnalité soit activé dans le fichier de configuration « `httpd.conf` » de votre Apache.

La copie d'écran ci-dessous correspond au fichier httpd.conf de XAMPP :



```

D:\Tools\xampp\apache\conf\httpd.conf - Notepad++
Fichier  Édition  Recherche  Affichage  Encodage  Langage  Paramétrage  Macro  Exécution  Complémen
new 2  httpd.conf
150 #LoadModule proxy_wstunnel_module modules/mod_proxy_wstunnel.so
151 #LoadModule ratelimit_module modules/mod_ratelimit.so
152 #LoadModule reflector_module modules/mod_reflector.so
153 #LoadModule remoteip_module modules/mod_remoteip.so
154 #LoadModule request_module modules/mod_request.so
155 #LoadModule reqtimeout_module modules/mod_reqtimeout.so
156 LoadModule rewrite_module modules/mod_rewrite.so
157 #LoadModule sed_module modules/mod_sed.so
158 #LoadModule session_module modules/mod_session.so
159 #LoadModule session_cookie_module modules/mod_session_cookie.so
  
```

NB : on voit dans la copie d'écran ci-dessus que la ligne relative au module « mod_rewrite » n'est pas en commentaire. Si dans votre cas elle l'était, retirez le dièse en début de ligne, sauvegardez le fichier et relancez Apache.

Nous pouvons sécuriser un peu plus l'application en procédant de la façon suivante :

- A la racine du projet nous allons ajouter un fichier index.php
- Toujours à la racine du projet, nous allons ajouter un autre fichier .htaccess

Source du script index.php à ajouter à la racine du projet :

```

<?php
// toute tentative d'accès au script index.php se soldera par
// une redirection vers le script index.php du répertoire web
header('Location: web/index.php');
  
```

Source du fichier .htaccess à ajouter à la racine du projet :

```

RewriteEngine On
# Toute tentative pour afficher autre chose que "index.php" se
# soldera par une redirection sur index.php
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^ index.php [QSA,L]
  
```

Avec ces quelques réglages en place, il devrait être impossible de sortir des rails que nous avons fixés, à savoir que toute requête doit passer par le contrôleur principal de l'application. Vous noterez que j'emploie le conditionnel dans la phrase qui précède, car en matière de sécurité, rien n'est jamais définitivement acquis.

Votre application SILEX (Symfony) est opérationnelle, on va pouvoir commencer à développer avec.


Pensez à faire un « `git add .` » et un « `git commit -m 'xxx'` ».

Attention au cache de Google Chrome :

Il peut arriver dans certains cas, que vous ayez modifié des redirections d'URL dans la configuration du fichier .htaccess, et que ces redirections – pourtant correctes sur le plan syntaxique - ne fonctionnent pas. Comme Google Chrome s'obstine à ne pas prendre en compte la nouvelle redirection, votre premier réflexe – logique – serait de soupçonner Apache de ne pas avoir pris en compte la nouvelle configuration du fichier .htaccess. En réalité, Apache n'est pas en cause, car Google Chrome utilise son propre cache d'URL, et s'il considère que vous lui avez déjà demandé une URL et que cette URL avait donné lieu à une redirection les fois précédentes, Google Chrome va s'appuyer sur son cache et effectuer de lui-même cette redirection en faisant l'économie d'une requête HTTP vers le serveur. Ce comportement de Google Chrome peut se comprendre, mais dans le cas présent il vous empêche de travailler. Pour contourner le problème, vous pouvez vider le cache de Chrome comme indiqué dans la procédure suivante :

<https://support.google.com/accounts/answer/32050?hl=fr>

Si vous n'avez pas accès à internet, voici un extrait du support Google :

1. Ouvrez Chrome.
2. Dans la barre d'outils de votre navigateur, cliquez sur le menu Chrome .
3. Cliquez sur **Plus d'outils > Effacer les données de navigation**.
4. Dans la boîte de dialogue qui s'affiche, cochez les cases **Cookies et autres données de site et de plug-in** et **Images et fichiers en cache**.
5. Utilisez le menu situé en haut de l'écran pour sélectionner la quantité de données à supprimer. Sélectionnez **tous** pour supprimer tous les éléments.
6. Cliquez sur **Effacer les données de navigation**.

NB : Votre professeur a rencontré ce problème qui lui a fait perdre un temps précieux. Si vous rencontrez vous-même ce problème et que vous vous souvenez de ce qui est écrit sur cette page, vous économiserez du temps (et donc de l'argent 😊), et vous vous épargnerez une poussée d'adrénaline 😞. Merci qui ?

2.6. Installation des frameworks Bootstrap (Twitter) et jQuery

Le répertoire « web » ne contient pas grand-chose au démarrage du projet, on va effectuer les actions suivantes :

- ajouter dans le répertoire « web » un sous-répertoire JS (pour Javascript),
- copier dans « web/css » le CSS de Bootstrap et dans « web/js » le code JS de Bootstrap,
- copier dans « web/js » différentes versions de jQuery, histoire de pouvoir utiliser l'une ou l'autre en fonction des besoins.
- Copier aussi le sous-répertoire « fonts » spécifique à Bootstrap.

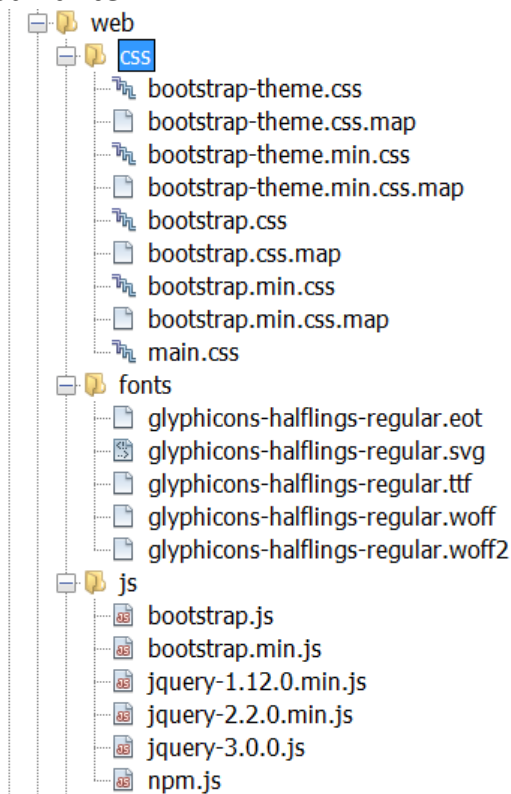
Vous pouvez récupérer Bootstrap et jQuery sur les sites officiels de ces 2 projets, ou les récupérer d'un projet local existant.

Les sites officiels des 2 projets sont :

<http://getbootstrap.com/>

<https://jquery.com/>

Après installation des 2 frameworks, on devrait obtenir à l'arrivée l'arborescence suivante :



Pensez à faire un « git add . » et un « git commit -m 'xxx' ».

2.7. Finalisation de l'environnement

Par défaut, le projet est livré avec deux fichiers `index.php`, celui qui se trouve dans le sous-répertoire « web » et un autre qui se trouve à la racine du projet et qui s'appelle « `index_dev.php` ». Le premier est un fichier « `index.php` » adapté à un environnement de production. Le second, comme son nom l'indique, est adapté à un environnement de développement, nous allons donc l'utiliser, en procédant à une simple permutation.

A la racine du projet, créez un répertoire « archive » dans lequel vous allez déplacer le fichier « `index_dev.php` ». Déplacez également dans ce sous-répertoire « archive » le fichier « `index.php` » qui se trouve actuellement dans « web », puis renommez-le en « `index_prod.php` ». Il sera plus facile de le reconnaître par la suite. Enfin, copiez le fichier « `index_dev.php` » vers le répertoire « web » puis renommez-le en « `index.php` ».

Ca y est, votre environnement de développement est prêt. Nous verrons par la suite, les avantages que nous procure le script « `index_dev.php` » (que nous venons de renommer en `index.php`), par rapport à la version de production.

3. Anatomie d'un projet SILEX

3.1. Notions d'architecture, et de MVC

L'architecture MVC a été conçue par un informaticien norvégien, Trygve Reenskaug, professeur à l'Université d'Oslo. Il a conçu cette architecture entre 1978 et 1979, pour répondre aux besoins inhérents au développement d'interfaces graphiques. Il semble qu'il ait présenté cette architecture pour la première fois en 79, au Xerox PARC (Palo Alto Research Center) de Californie. Il semble aussi qu'une des premières implémentations de MVC ait été réalisée en 80, au même Xerox PARC, avec le langage à objets Smalltalk. Si l'on considère que Steve Jobs, quelques années plus tard, s'inspirera largement des travaux du Xerox PARC (la souris, l'interface graphique, etc...) pour créer le premier Macintosh, on peut considérer que Trygve Reenskaug a apporté énormément au secteur du développement logiciel.

L'architecture MVC a mis un certain temps avant d'être adoptée par les développeurs webs. Ruby On Rails (RoR) est l'un des premiers frameworks de développement web à avoir intégré cette architecture de manière très poussée, et il a contribué à la populariser (vers 2005-2006). La plupart des frameworks de développement PHP ont intégré ces principes, pour certains dans la même période, pour les autres dans les années qui ont suivi. Les frameworks PHP tels que Zend Framework, Symfony, Laravel, Yii, etc.. implémentent tous une architecture de type MVC.

Nous allons découvrir en mode « pas à pas » les principes de l'architecture MVC, architecture sous-jacente à SILEX (et à Symfony).

Nous allons dans un premier temps étudier la notion de vue et de contrôleur, nous nous pencherons plus tard sur la notion de modèle.

3.2. Les Vues et le templating (avec Twig)

Attention : nous allons utiliser le moteur de templating Twig dans la suite de ce cours. Si vous souhaitez travailler sous Netbeans, et bénéficier de la coloration syntaxique sur les templates Twig, installez le plugin « Twig templates ».

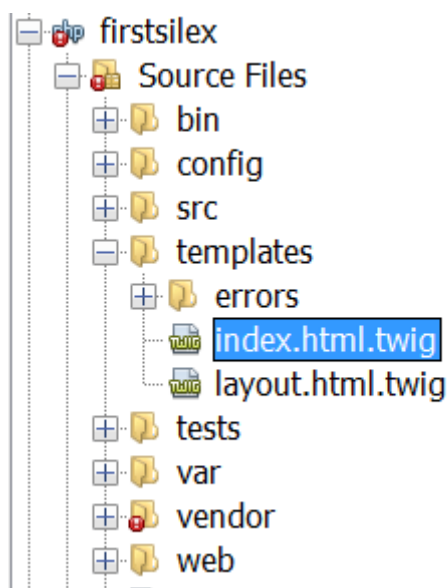
Lors de nos premiers tests, en saisissant par exemple l'URL suivante avec Apache démarré...

<http://localhost/firstsilex/web/>

... nous avons obtenu le message suivant :

Welcome to your new Silex Application!

D'où vient ce message ? Eh bien il est stocké « en dur » dans une vue, qui se trouve dans le répertoire « Templates » :



Cette vue est un fichier index.html, avec le suffixe Twig, observons son code :

```
{% extends "layout.html.twig" %}

{% block content %}
    Welcome to your new Silex Application!
{% endblock %}
```

Les symboles `{% %}` font partie de la syntaxe spécifique au composant Twig. Nous allons y revenir.

Twig est un composant développé par la même équipe qui développe SILEX et Symfony, à savoir la société Sensiolabs. Twig est un projet PHP à part entière, qui peut être utilisé au sein de projets n'utilisant pas SILEX et Symfony. Twig est préinstallé sur SILEX et Symfony, car c'est un moteur de templating très puissant et très pratique, il est donc très souvent utilisé avec ces frameworks.

Le site officiel du projet Twig est le suivant :

<http://twig.sensiolabs.org>

Twig est un concurrent direct de projets PHP plus anciens, tels que Smarty ou TPLN, qui étaient des moteurs de templating très utilisés par les développeurs PHP dans les années 2000. Plus moderne et surtout plus avancé que ses vieux concurrents, Il semble que Twig soit en passe de les détrôner, si ce n'est déjà fait.

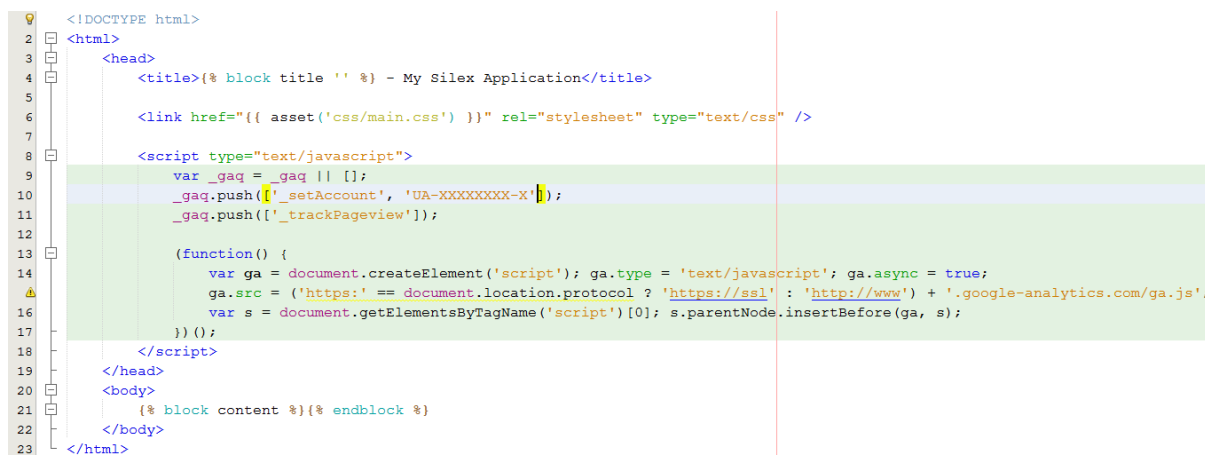
Donc notre vue SILEX utilise le composant Twig pour afficher des données dans les pages, très bien, alors voyons comment ça fonctionne.

Twig gère la notion d'héritage entre pages, on le voit au travers de la ligne suivante :

```
{% extends "layout.html.twig" %}
```

Pour Twig, chaque vue fonctionne comme une classe, et nous pouvons dire à Twig qu'une page hérite du contenu et des caractéristiques d'une autre vue. C'est ce que nous indiquons à Twig ici, en lui disant de charger dans la vue en cours, le contenu de la vue « layout.html.twig ». On notera que le mot clé « extends » a le même sens ici que le mot clé « extends » utilisé en PHP pour l'héritage de classe. Si vous comprenez le principe de l'héritage de classe en PHP, vous comprenez donc ce qui est train de se jouer ici.

Jetons un coup d'œil à la vue « layout.html.twig » :



```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>{% block title ' ' %} - My Silex Application</title>
5
6     <link href="{{ asset('css/main.css') }}" rel="stylesheet" type="text/css" />
7
8     <script type="text/javascript">
9       var _gaq = _gaq || [];
10      _gaq.push(['_setAccount', 'UA-XXXXXXX-X']);
11      _gaq.push(['_trackPageview']);
12
13      (function() {
14        var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;
15        ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.google-analytics.com/ga.js';
16        var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);
17      })();
18    </script>
19  </head>
20  <body>
21    {% block content %}{% endblock %}
22  </body>
23 </html>

```

On voit qu'il s'agit d'une structure de page HTML classique, mais vous noterez la présence d'une ligne importante dans le bloc « body » :

```
{% block content %}{% endblock %}
```

Cette ligne indique à Twig qu'il doit réserver à cet endroit de la page un bloc qui s'appelle « content », bloc qui héritera du contenu de la page « fille ». C'est grâce à la présence de ce bloc que la vue « index.html.twig » sait où placer son contenu, après avoir hérité du contenu de la classe mère « layout.html.twig ».

Maintenant, dans la page de votre navigateur qui affiche le message « welcome... », faites un clic-droit et demandez l'option « affichage du code source ». Vous allez constater que la page contient bien un « mix » entre le code des 2 vues, et que le code HTML de la vue « fille » s'est placé au bon endroit au sein du code HTML de la vue « mère ».

Twig enrichit le HTML avec ses propres balises. Ces balises permettent à Twig de fonctionner et de générer ce qu'on lui demande de générer, puis Twig les élimine du résultat final (en l'occurrence le HTML envoyé au navigateur). Twig a trois types de balises qui sont :

- Une balise d'affichage symbolisée par les doubles accolades :
 - o {{ affichage }}
- Une balise d'instruction, ou de bloc d'instructions, symbolisée par la combinaison d'accolades et de pourcentages
 - o {% instruction %}

- Une balise de commentaire, symbolisée par une accolade et un dièse :
 - o {# commentaire #}

Nous approfondirons l'utilisation des différentes balises de Twig dans les exemples suivants. En ce qui concerne les balises d'instructions, retenez pour l'instant qu'elles vont nous permettre de placer des boucles et des débranchements, comme par exemple :

```
{% if titre is defined %}
    <p>Titre: {{ titre }} </p>
{% endif %}
```

Ou encore :

```
{% for album in listebd %}
    <li>Titre: {{ album.album }}, {{ album.auteur }} </li>
{% endif %}
```

Nous utiliserons ces techniques dans différents exemples par la suite.

Avant de poursuivre notre étude de SILEX, et puisque nous avons embarqué le framework CSS Bootstrap dans notre projet, profitons-en pour nous poser 2 minutes et donner à notre vue « layout.html.twig » quelques une des couleurs de Bootstrap.

Modifiez la vue « index.html.twig » en y ajoutant une balise « div » avec la classe « jumbotron » :

```
1  {% extends "layout.html.twig" %}
2
3  {% block content %}
4      <div class="jumbotron">
5          Welcome to your new Silex Application!
6      </div>
7  {% endblock %}
8
```

Maintenant, modifiez la vue « index.layout.twig » de la façon suivante :

```
<!DOCTYPE html>
<html>
  <head>
    <title>{% block title '' %} - My Silex Application</title>
    <meta name="viewport" content="width=device-width">
    <link href="{{ asset('css/bootstrap.min.css') }}" rel="stylesheet" type="text/css">
    <link href="{{ asset('css/main.css') }}" rel="stylesheet" type="text/css" />
    <script src="{{ asset('js/jquery-2.2.0.min.js') }}"></script>
    <script src="{{ asset('js/bootstrap.min.js') }}"></script>

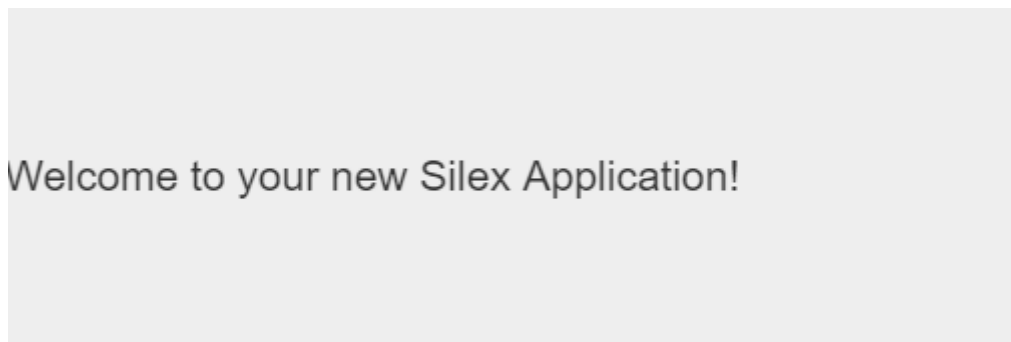
  </head>
  <body>
    {% block content %}{% endblock %}
  </body>
</html>
```

Vous noterez que nous avons supprimé le bloc de code Javascript. Ce bloc est utile si nous souhaitons référencer nos pages sur le moteur de Google, dans le cas contraire il ne nous est d'aucune utilité. Si nous considérons que nous souhaitons ici développer une application destinée à l'usage interne d'une entreprise, alors le référencement de nos pages sur Google n'est pas d'actualité. Nous avons également ajouté quelques lignes pour inclure le CSS et le Javascript dont Bootstrap a besoin pour fonctionner. Le Javascript de Bootstrap est surtout utile pour la gestion des menus déroulants, et il a besoin de la présence de jQuery pour pouvoir fonctionner.

Vous noterez aussi la présence de la ligne ci-dessous, qui permet sur les terminaux équipés d'un écran tactile de zoomer/dézoomer avec deux doigts (le fameux geste du « pincement ») :

```
<meta name="viewport" content="width=device-width">
```

Rafraîchissez la page dans votre navigateur :



On voit que le CSS de Bootstrap a bien été pris en compte.

Vous aurez sans doute noté la présence en bas du navigateur d'une barre d'état :



Cette barre d'état est l'un des outils fournis par Symfony, elle est apparente car nous utilisons le fichier « index.php » dédié à un environnement de développement. Elle n'apparaît pas quand nous utilisons le fichier « index.php » dédié à l'environnement de production (dont on rappelle que nous l'avons déplacé temporairement dans le répertoire « archive »).

Le code « 200 » en début de ligne indique que tout s'est bien passé.

Nous allons provoquer volontairement une erreur pour voir ce qui se passe. Dans la vue « layout.html.twig », modifiez l'une des lignes en omettant une parenthèse fermante :

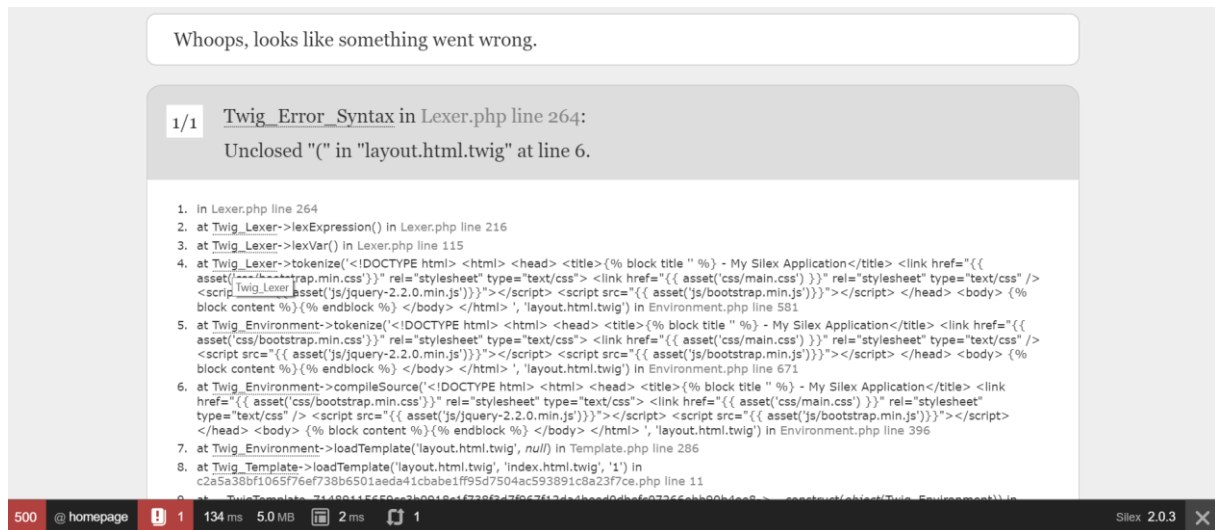
- Avant l'erreur :

```
<link href="{{ asset('css/bootstrap.min.css') }}" rel="stylesheet" type="text/css">
```

- Avec l'erreur :

```
<link href="{{ asset('css/bootstrap.min.css') }}" rel="stylesheet" type="text/css">
```

Rafraîchissez la page du navigateur et observez :



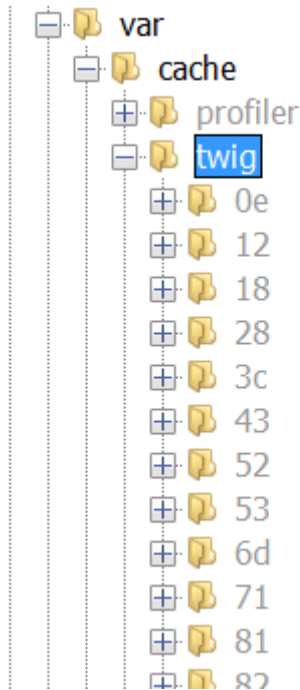
Nous avons déclenché une erreur 500, et Twig nous indique qu'il s'est planté sur notre template à la ligne 6, à cause d'une parenthèse non refermée. Corrigez l'erreur, sauvegardez et rafraîchissez le navigateur, tout va rentrer dans l'ordre.

On voit que SILEX (et donc Symfony), nous apporte une assistance précieuse durant le processus de développement.

Nous allons en rester là avec les vues pour le moment, nous y reviendrons par la suite, allons maintenant étudier ce qui se passe du côté des contrôleurs.

Mais avant de quitter ce chapitre, et vu que nous avons fait plusieurs modifications, il est temps de les enregistrer dans notre historique Git. Faites d'abord un « git status » pour voir les modifications réalisées. Puis faites un « git add --all » pour prendre en compte aussi les suppressions et/ou déplacement de sources, et enfin un « git commit » avec un petit message.

Avertissement : Twig gère un cache interne, qui lui sert à optimiser certaines opérations de génération de code HTML. Ce cache se trouve dans le répertoire « `var/cache/twig/` » :



Il arrive que, après avoir corrigé un code erroné utilisant Twig, la page concernée continue à afficher l'erreur, donnant ainsi l'impression que la correction (ou modification) a été sans effet. En réalité, c'est peut être le gestionnaire de cache de Twig qui nous empêche de constater l'effet de notre modification. En cas de doute, vous pouvez vider l'intégralité du répertoire « `var/cache/twig/` », avant de procéder à un nouveau test.

Il semble qu'il soit possible de désactiver le cache de Twig, mais mes tentatives se sont révélées jusqu'ici inopérantes.

Quelques liens utiles :

<http://www.lafabriquedecode.com/blog/2014/05/symfony-2-en-finir-nettoyage-du-cache-via-cacheclear/>

<http://stackoverflow.com/questions/18883546/symfony2-disable-twig-cache>

<https://www.baptiste-donaux.fr/twig-accelerer-generation-templates/>

<https://www.baptiste-donaux.fr/twig-include-performance/>

3.3. Les Contrôleurs

Nous avons commencé à voir comment fonctionnaient les vues, mais ces vues ne pourraient pas fonctionner sans le pilotage des contrôleurs.

Allons tout de suite lire le code du script suivant : `/src/controllers.php`

```
<?php
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpFoundation\RedirectResponse;
use Symfony\Component\HttpKernel\Exception\NotFoundHttpException;

//Request::setTrustedProxies(array('127.0.0.1'));

$app->get('/', function () use ($app) {
    return $app['twig']->render('index.html.twig', array());
})
->bind('homepage')
;

$app->error(function (\Exception $e, Request $request, $code) use ($app) {
    if ($app['debug']) {
        return;
    }

    // 404.html, or 40x.html, or 4xx.html, or error.html
    $templates = array(
        'errors/'.$code.'.html.twig',
        'errors/'.$substr($code, 0, 2).'x.html.twig',
        'errors/'.$substr($code, 0, 1).'xx.html.twig',
        'errors/default.html.twig',
    );

    return new Response($app['twig']->resolveTemplate($templates)-
>render(array('code' => $code)), $code);
});
```

Explications : nous sommes dans le contrôleur principal de l'application. Ce contrôleur définit pour le moment deux routes :

- Une route qui va être utilisée si l'utilisateur saisit l'URL du projet sans préciser de sous-répertoire, par exemple :
 - <http://localhost/firstsilex2/web/>

Cette route est matérialisée par le code suivant :

```
$app->get('/', function () use ($app) {
    return $app['twig']->render('index.html.twig', array());
})
->bind('homepage')
;
```

- Une route qui est en fait une « voie de garage », pour le cas où vous demanderiez une route qui n'existe pas au sein de l'application (par exemple : .../web/titi

Cette « voie de garage » est matérialisée par le dernier groupe de code, dont je n'ai mis qu'un extrait :

```
$app->error(
    ...
) ;
```

Après la route existante, ajoutez-en une seconde, que vous appellerez « test » :

```
$app->get('/', function () use ($app) {
    return $app['twig']->render('index.html.twig', array());
})
->bind('homepage')
;
$app->get('/test/', function () use ($app) {
    return $app['twig']->render('test.html.twig', array());
})
->bind('testpage')
;
```

La méthode « bind » permet de donner à notre nouvelle route un nom qui permet à SILEX de bien la distinguer des autres routes. Nous l'avons appelée ici « testpage », et elle « pointe » sur le répertoire « /test/ » qui se trouve à

l'intérieur du répertoire public « web » (nous n'avons pas besoin de préciser ce dernier point, c'est implicite, SILEX le gère pour nous).

Allez maintenant dans le répertoire « templates » et créez un fichier « test.html.twig » dans lequel vous allez placer le code suivant :

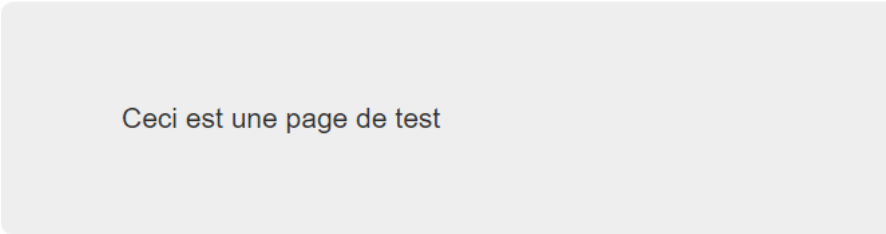
```
{% extends "layout.html.twig" %}

{% block content %}
<div class="jumbotron">
    Ceci est une page de test
</div>
{% endblock %}
```

Dans la barre de votre navigateur, saisissez l'URL suivante :

<http://localhost/firstsilex/web/test/>

Si vous n'avez pas fait d'erreur, vous devriez voir apparaître une page contenant :



Ceci est une page de test

Voilà, vous venez de créer votre première route, au sens MVC du terme.

Maintenant, supposons que nous souhaitions transmettre un paramètre à une page :

<http://localhost/firstsilex/web/testparam/titi>

Dans l'exemple ci-dessus, nous avons créé une route qui s'appelle « testparam » et qui reçoit un seul paramètre qui a pour valeur « titi ». Pour ce faire, nous ajoutons un nouveau contrôleur dans le fichier « controllers.php » :

```
$app->get('/testparam/{id}', function ($id) use ($app) {
    return $app['twig']->render('testparam.html.twig', array());
})
->bind('testparam')
;
```

Et nous ajoutons une nouvelle vue dans le répertoire « templates » que nous appellerons « testparam.html.twig » :

```
{% extends "layout.html.twig" %}

{% block content %}
<div class="jumbotron">
    Ceci est une page de test avec paramètre
</div>
{% endblock %}
```

Essayez l'URL suivante :

<http://localhost/firstsilex/web/testparam/titi>

La page s’affiche bien, mais comment récupérer le paramètre « titi » et surtout comment l’afficher à l’intérieur de notre template ?

En fait, c’est très simple. Modifiez votre contrôleur « testparam » de la façon suivante :

```
$app->get('/testparam/{id}', function ($id) use ($app) {
    return $app['twig']->render('testparam.html.twig',
        array(
            'param1' => $id
        ));
})
->bind('testparam')
;
```

Notre route « testparam » est maintenant en mesure de prendre un paramètre que nous avons appelé \$id. Vous noterez que ce paramètre est transmis à la fonction anonyme qui est utilisée par le contrôleur, et que nous transmettons ensuite le paramètre \$id à un paramètre « param1 », qui lui sera utilisé à l’intérieur de la vue « testparam.html.twig ». Nous aurions pu appeler ce dernier paramètre « id », mais nous avons préféré l’appeler « param1 » pour bien montrer qu’il s’agit d’un paramètre différent.

En résumé, nous avons transmis le contenu de la variable \$id au template Twig via le paramètre « param1 ». \$id est une variable qui ne fait pas partie du « scope » (c’est-à-dire du « périmètre ») du template Twig, elle n’est pas utilisable à l’intérieur de la vue générée par Twig, tandis que « param1 » fait bien partie de ce scope, comme nous allons le constater maintenant.

Modifiez la vue testparam.html.twig :

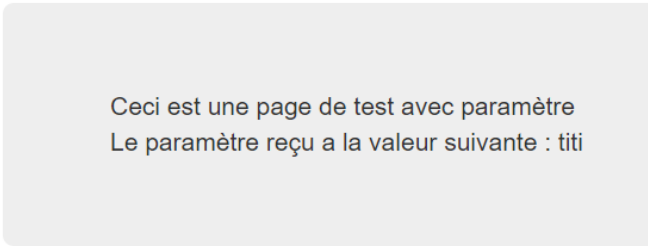
```
{% extends "layout.html.twig" %}

{% block content %}
<div class="jumbotron">
    Ceci est une page de test avec paramètre<br>
    Le paramètre reçu a la valeur suivante : {{param1}}
</div>
{% endblock %}
```

Nous avons ajouté à l'intérieur de la vue, le paramètre « param1 », en utilisant les doubles accolades. Ces doubles accolades sont le moyen utilisé par Twig pour identifier les variables reçues par la vue, et surtout pour vous permettre de les afficher à l'endroit qui vous arrange.

Redemandez l'affichage de la page suivante, vous devriez voir apparaître « titi » à l'intérieur de la page affichée :

<http://localhost/firstsilex/web/testparam/titi>



Ceci est une page de test avec paramètre
Le paramètre reçu a la valeur suivante : titi

3.4. Vous reprendrez bien un peu de Twig

Supposons maintenant que nous souhaitons afficher une liste dans une vue, comme par exemple le contenu d'un tableau associatif dans un tableau HTML. Nous allons réaliser ce premier exemple en utilisant une liste d'albums de bande dessinée.

Ajoutez un nouveau contrôleur que vous appellerez « listebd » :

```
$app->get('/listebd/{id}', function ($id) use ($app) {
    require_once 'tempdata/liste_bd_temp.php';

    return $app['twig']->render('listebd.html.twig',
        array(
            'param1' => $id,
            'listebd' => getListeBD()
        ));
})
->bind('listebd')
;
```

On voit que la liste des BD provient d'une fonction getListeBD(), qui est stockée dans le script « liste_bd_temp.php ». Je vous propose de créer un sous-répertoire « tempdata » à l'intérieur du sous-répertoire « src », pour y placer le source suivant.

Source du script « liste_bd_temp.php » :

```
function getListeBD() {
    $listeBD = array();
    $listeBD[1] = array(
        'id' => 1,
        'album' => 'Garulfo',
        'auteur' => 'Ayroles, Maïorana, Leprévost',
        'editeur' => 'Delcourt',
        'parution' => '2011-05-15'
    );
    $listeBD[2] = array(
        'id' => 2,
        'album' => 'horologiom',
        'auteur' => 'fabrice Lebeault',
        'editeur' => 'Delcourt',
        'parution' => '2005-01-01'
    );
    $listeBD[3] = array(
        'id' => 3,
        'album' => 'Le château des étoiles',
        'auteur' => 'Alex Alice',
        'editeur' => 'Rue De Sevres',
        'parution' => '2014-05-01'
    );
    $listeBD[4] = array(
        'id' => 4,
        'album' => 'Le voyage extraordinaire',
        'auteur' => 'Camboni, Filippi',
        'editeur' => 'Vents d\'Ouest',
        'parution' => '2012-09-01'
    );
    return $listeBD;
}
```

NB : la liste d'albums de bande dessinée ci-dessus existe réellement. La plupart de ces albums sont en réalité des séries, puisqu'ils ont été publiés en plusieurs tomes, mais j'ai occulté volontairement cette notion pour rester sur un jeu de données très simple. Concernant la date de parution, l'année correspond à la parution du premier tome, mais les mois et jours sont fictifs (j'avais besoin d'une

date complète pour certains des exemples qui vont suivre, et je n'avais pas le temps de vérifier les dates exactes de parution des différents albums).

Créez maintenant une nouvelle vue (toujours dans le répertoire « templates ») que vous appellerez « listebd.html.twig » :

```
{% extends "layout.html.twig" %}

{% block content %}
<div class="jumbotron">
    <h2>Albums de BD</h2>
    Le paramètre "param1" a la valeur suivante : {{param1}} (il n'est pas
    utilisé ici)

    <table class="table table-striped">
        <thead>
            <tr>
                <th>#</th>
                <th>Titre</th>
                <th>Auteur(s)</th>
                <th>Editeur</th>
                <th>Parution</th>
            </tr>
        </thead>
        <tbody>
            {% for album in listebd %}
                <tr>
                    <td>{{album.id}}</td>
                    <td>{{album.album}}</td>
                    <td>{{album.auteur}}</td>
                    <td>{{album.editeur}}</td>
                    <td>{{album.parution}}</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
</div>
{% endblock %}
```

Vous noterez ici la présence d'une boucle « for » pilotée par Twig. Son fonctionnement est similaire à celui de la boucle « foreach » de PHP.

L'URL ci-dessous devrait vous permettre d'afficher une page HTML contenant la liste des albums de BD :

<http://localhost/firstsilex/web/listebd/200>

Albums de BD				
Le paramètre "param1" a la valeur suivante : 200 (il n'est pas utilisé ici)				
#	Titre	Auteur(s)	Editeur	Parution
1	Garulfo	Ayroles, Maïorana, Leprévost	Delcourt	2011-05-15
2	horologiom	fabrice Lebeault	Delcourt	2005-01-01
3	Le château des étoiles	Alex Alice	Rue De Sevres	2014-05-01
4	Le voyage extraordinaire	Camboni, Filippi	Vents d'Ouest	2012-09-01

Cet exemple est intéressant car il nous a permis de voir comment combiner dans une même vue des données provenant d'une requête HTTP de type GET (avec l'exemple du paramètre « id » qui ici ne sert à rien), avec des données stockées au sein de l'application (ici une liste d'albums de bande dessinée stockée dans un tableau associatif, mais ce pourrait être un jeu de données renvoyé par une requête SQL, comme nous le verrons par la suite).

Avant de poursuivre, pensez à enregistrer vos modifications dans l'historique de Git (via « git add » et « git commit »).

Il faut souligner que Twig fournit un certain nombre de filtres intéressants pour améliorer le rendu des données affichées. Dans l'exemple ci-dessous, le filtre « capitalize » force la première lettre de la chaîne de caractères en majuscule, tandis que le filtre « title » force la première lettre de chaque mot en majuscule. Le filtre « date » se passe de commentaire :

```
<tr>
  <td>{{album.id}}</td>
  <td>{{album.album|capitalize}}</td>
  <td>{{album.auteur|title}}</td>
  <td>{{album.editeur|title}}</td>
  <td>{{album.parution|date("d/m/Y")}}</td>
</tr>
```

Voici la liste des filtres proposés actuellement par Twig :

- [abs](#)
- [batch](#)
- [capitalize](#)
- [convert_encoding](#)
- [date](#)
- [date_modify](#)
- [default](#)
- [escape](#)
- [first](#)
- [format](#)
- [join](#)
- [json_encode](#)
- [keys](#)
- [last](#)
- [length](#)
- [lower](#)
- [merge](#)
- [nl2br](#)
- [number_format](#)
- [raw](#)
- [replace](#)
- [reverse](#)
- [round](#)
- [slice](#)
- [sort](#)
- [split](#)
- [striptags](#)
- [title](#)
- [trim](#)
- [upper](#)
- [url_encode](#)

Prenez à ce stade le temps de lire la documentation de Twig, et de tester les différents filtres que ce composant propose.

Le lien vers la documentation officielle de Twig, pour les filtres :

<http://twig.sensiolabs.org/doc/filters/index.html>

3.5. Des menus au look de Bootstrap

Il est temps, pour conclure cette première partie du cours, de donner un look un peu plus professionnel à notre application. Pour ce faire, nous allons repomper un bout de code pris sur un menu de Bootstrap, et nous allons l'insérer dans le fichier « layout.html.twig » :

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>{% block title ' ' %} - My Silex Application</title>
    <meta name="viewport" content="width=device-width">
    <link href="{{ asset('css/bootstrap.min.css') }}" rel="stylesheet" type="text/css">
    <link href="{{ asset('css/main.css') }}" rel="stylesheet" type="text/css" />
    <script src="{{ asset('js/jquery-2.2.0.min.js') }}"></script>
    <script src="{{ asset('js/bootstrap.min.js') }}"></script>
  </head>
  <body>
    <!-- Fixed navbar -->
    <nav class="navbar navbar-inverse navbar-fixed-top">
      <div class="container">
        <div class="navbar-header">
          <button type="button" class="navbar-toggle collapsed"
            data-toggle="collapse" data-target="#navbar"
            aria-expanded="false" aria-controls="navbar">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>
          <a class="navbar-brand" href="#">FirstSilex Project</a>
        </div>
        <div id="navbar" class="collapse navbar-collapse">
          <ul class="nav navbar-nav">
            <li class="active"><a href="{{ url('homepage') }}">Accueil</a></li>
            <li class="dropdown">
              <a href="#" class="dropdown-toggle" data-toggle="dropdown"
                role="button" aria-haspopup="true"
                aria-expanded="false">Travaux <span class="caret"></span></a>
              <ul class="dropdown-menu">
                <li><a href="{{ url('testpage') }}">Page de test</a></li>
                <li><a href="{{ url('testparam', {'id':'200'}) }}">Page de test avec
paramètre 200</a></li>
                <li><a href="{{ url('listebd', {'id':'200'}) }}">Albums de BD</a></li>
              </ul>
            </li>
          </ul>
        </div><!--/.nav-collapse -->
      </div>
    </nav>
    <div class="container">
      <br><br>
      <div class="page-header">
        <h2>SILEX Project</h2>
      </div>
      {% block content %}{% endblock %}
    </div>
```

```

<footer class="footer">
  <div class="container">
    <p class="text-muted">Premier projet avec SILEX</p>
  </div>
</footer>
</body>
</html>

```

Au tout début du template, nous avons inséré quelques informations générales mais néanmoins importantes, telles que la langue et l'encodage de la page (ces éléments sont indiqués en gras ci-dessous) :

```

<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">

```

Mais le plus important reste à venir. Ce sont les quelques lignes de menu qui sont reliées à des pages de notre application :

- L'option de menu liée à la page d'accueil :

```
<li class="active"><a href="{{ url('homepage') }}">Accueil</a></li>
```

- Les options de menu qui se trouvent dans le menu « travaux » :

```

<li><a href="{{ url('testpage') }}">Page de test</a></li>
<li><a href="{{ url('testparam', {'id':'200'}) }}">Page de test avec paramètre
200</a></li>
<li><a href="{{ url('listebd', {'id':'200'}) }}">Albums de BD</a></li>

```

Si vous vous souvenez bien, nous avons créé dans notre contrôleur 4 routes qui sont les suivantes :

- homepage
- testpage
- testparam
- listebd

Ce sont ces 4 routes que nous avons déclarées dans le code HTML, au travers de la fonction `url()`, fonction qui est fournie par Twig. Ce sont également ces 4 noms

de route que nous avons déclarés dans le script controllers.php. Twig va s'appuyer sur ces noms de routes pour générer le code HTML correspondant, en l'occurrence des balises HTML de type `<a>`. Par exemple, si vous faites un clic-droit sur l'option de menu correspondant à « homepage », et que vous demandez l'option « inspecter élément », vous trouverez le code suivant :

```
<a href="http://localhost/firstsilex/web/index.php/">Accueil</a>
```

Point important : nous aurions pu utiliser la fonction Twig `path()`, en remplacement de la fonction `url()`, pour la génération de nos menus :

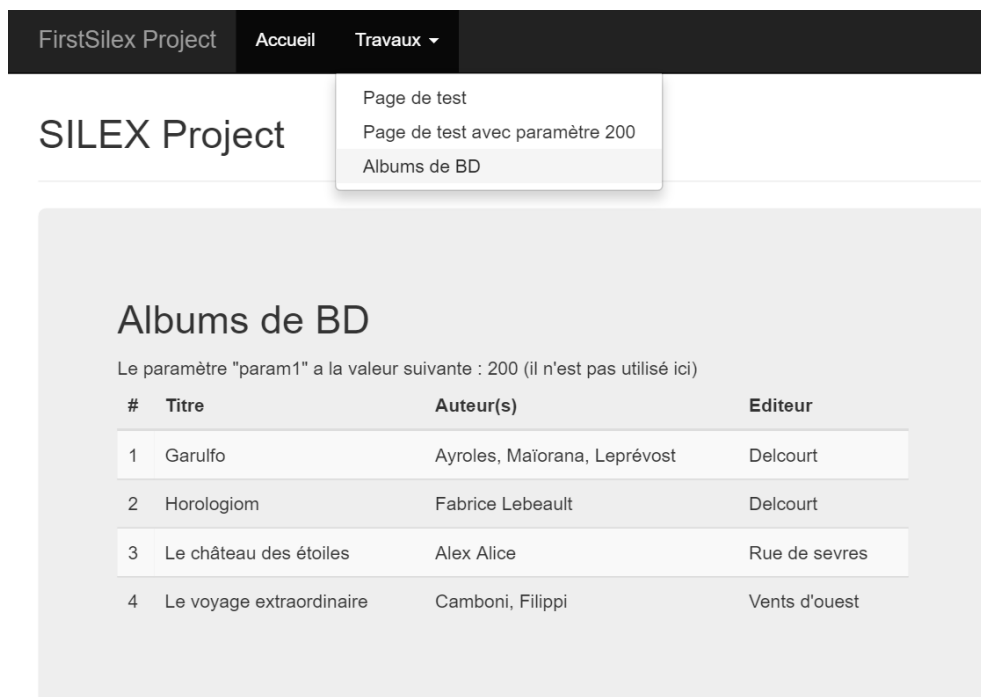
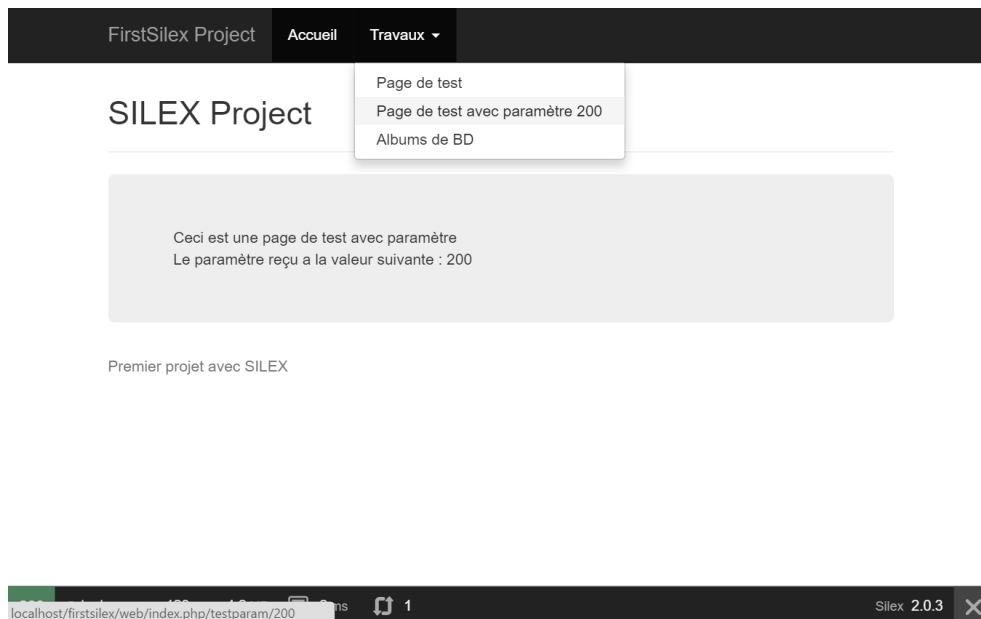
```
<li class="active"><a href="{{ path('homepage') }}">Accueil</a></li>
```

Le code HTML généré aurait alors été le suivant :

```
<a href="/firstsilex/web/index.php/">Accueil</a>
```

La fonction `url()` génère un chemin « absolu » tandis que la fonction `path()` génère un chemin « relatif ». En règle générale on préférera générer des chemins relatifs, sauf cas particulier.

Voici quelques exemples de pages obtenues avec cette première version de l'application :



3.6. Conclusion provisoire

Au cours de cette première découverte de SILEX, nous avons abordé une partie de l'architecture MVC, en l'occurrence les Vues et les Contrôleurs (nous aborderons la partie « Modèles » par la suite).

Le mécanisme dit de « routage » que nous avons commencé à utiliser est un mécanisme puissant, qui offre une grande souplesse de développement. On retrouve ce mécanisme dans tous les frameworks construits selon une architecture de type MVC. On notera qu'il existe d'autres types d'architecture, comme MVVM (Modèle Vue Vue Modèle), ou MVP (Modèle Vue Présentation). Ce sont des architectures très intéressantes à étudier, mais elles sont en dehors du cadre de ce support.

On notera que, dans une approche plus traditionnelle du développement PHP, il était courant que chaque lien de page au sein d'une application soit lié à un script PHP différent. Dans ce contexte, chaque script devait faire appel aux mêmes mécanismes de sécurité, le moindre oubli pouvant être fatal en termes de sécurité.

Dans l'approche proposée par SILEX (et donc par Symfony), le script `index.php` est le point d'entrée unique de l'application. Ce script `index.php` instancie l'objet `Application`, il « charge » dans la mémoire de l'application les différentes routes déclarées dans `controllers.php`, et enfin il lance l'application, au moyen de la méthode `run()`. A partir de là, l'objet `Application` pilote l'ensemble du processus de génération des pages, en détectant la route sélectionnée par l'utilisateur et en chargeant les données et les templates Twig correspondant à cette route.

Cette approche proposée par Twig, fondée sur un composant « chef d'orchestre » (l'option `Application`), existe sous des formes différentes dans la plupart des frameworks PHP du marché.

4. Approfondissements autour de SILEX

4.1. Les formulaires

Pour nous initier au fonctionnement des formulaires dans SILEX, nous allons créer un formulaire de mise à jour des albums de bande dessinée, s'appuyant sur notre base de données fictive (que nous remplacerons plus tard par une base MySQL).

Mais avant de commencer à développer des formulaires, commençons avec un peu de théorie.

C'est Twig qui va piloter la génération des formulaires, mais pour ce faire, il a besoin que l'on lui fournisse plusieurs services complémentaires. Ces services vont être apportés à Twig par l'intermédiaire de fournisseurs de services (en anglais : « service providers »), dont « FormServiceProvider », « ValidatorServiceProvider », « LocaleServiceProvider », et « TranslationServiceProvider ».

Nous allons ajouter ces fournisseurs de service, en modifiant le script `src/app.php` (les lignes en gras ci-dessous indiquent les principaux changements) :

```
<?php

use Silex\Application;
use Silex\Provider\AssetServiceProvider;
use Silex\Provider\TwigServiceProvider;
use Silex\Provider\ServiceControllerServiceProvider;
use Silex\Provider\HttpFragmentServiceProvider;

// formulaire et traduction
use Silex\Provider\ValidatorServiceProvider;
use Silex\Provider\FormServiceProvider;
use Silex\Provider\LocaleServiceProvider;
use Silex\Provider\TranslationServiceProvider;

$app = new Application();

$app->register(new ServiceControllerServiceProvider());
$app->register(new AssetServiceProvider());
```



```

$app->register(new TwigServiceProvider(), array(
    'twig.path' => array(__DIR__.'/../templates'),
    'twig.options' => array('cache' => __DIR__.'/../cache'),
));
$app->register(new HttpFragmentServiceProvider());

// formulaire et traduction
$app->register(new FormServiceProvider());
$app->register(new Silex\Provider\LocaleServiceProvider());
$app->register(new Silex\Provider\TranslationServiceProvider(), array(
    'locale_fallbacks' => array('en'),
));

return $app;

```

Twig propose plusieurs manières de générer des formulaires.

La documentation officielle de Symfony propose des exemples intéressants que je vous invite à étudier :

http://symfony.com/doc/current/form/form_customization.html

Voici un exemple de champ de saisie, accompagné de son label, tel qu'il est possible d'en générer avec Twig (exemple emprunté à un autre tuto) :

```

<p>
    {{ form_label(form.mail, 'Votre mail : ') }}
    {{ form_widget(form.mail, {'attr' : { 'maxlength' : '50',
                                          'placeholder' : 'cara@melmou.fr' }} ) }}
</p>

```

On voit ici qu'il est possible de personnaliser la plupart des attributs d'un élément de formulaire généré par Twig, avec les exemples de « maxlength » et « placeholder ».

Voici un autre exemple emprunté à la documentation officielle de Twig :

```

<div>
    {{ form_label(form.age) }}
    {{ form_errors(form.age) }}
    {{ form_widget(form.age) }}
</div>

```

Voici le code HTML que Twig va générer pour le code ci-dessus :

```
<div>
  <label for="form_age">Age</label>
  <ul>
    <li>This field is required</li>
  </ul>
  <input type="number" id="form_age" name="form[age]" />
</div>
```

La génération de ce code HTML ne se fait pas par magie, le développeur doit définir, à un autre niveau, le type de champ à générer, les éventuelles classes CSS à appliquer, et surtout les contrôles à effectuer par rapport à la saisie de l'utilisateur. C'est du côté des contrôleurs que le plus gros du travail va se faire.

Nous allons découvrir la manière dont cela fonctionne, au travers de la création d'un premier formulaire. Ce formulaire constituera un brouillon par rapport au module de gestion des albums (de bande dessinée) que nous développerons par la suite.

Nous allons commencer par ajouter une nouvelle route dans le script `src/controllers.php`. Comme il s'agit d'un formulaire provisoire, destiné à nous initier au fonctionnement des formulaires avec Twig, nous l'appellerons « `albumupdate-draft` ».

Voici le code source de départ de cette nouvelle route :

```
$app->match('/albumupdate-draft/{id}', function (Request $request,
Silex\Application $app) {

    // TODO : nous ajouterons le code de génération de formulaire ici

})
->bind('albumupdate-draft');
```

Vous vous demandez sans doute à quoi correspond la méthode « `match()` » ci-dessus. Eh bien, il faut se rappeler que les formulaires ont quelques spécificités, du point de vue du protocole HTTP.

Le cas de formulaire le plus simple, c'est le formulaire de recherche, auquel on affecte généralement la méthode « `GET` », ce qui a pour effet de générer une

requête HTTP de type GET. Dans ce cas de figure, une route de type « get » fera très bien l'affaire :

```
$app->get('/recherche/{cible}', function (Request $request,
Silex\Application $app) {

    // TODO : nous ajouterons Le code de génération de formulaire ici

})
->bind('recherche');
```

Dans le cas de formulaire de mise à jour, on utilise de préférence la méthode POST qui offre plusieurs avantages (par rapport à GET), que je ne rappelle pas ici. Mais cela a un effet de bord sur notre système de routage. En effet, la première fois que nous accédons à une page contenant un formulaire, nous y arrivons généralement par l'intermédiaire d'une requête HTTP de type GET.

Si le formulaire contient des erreurs de saisie, nous allons le réafficher en boucle jusqu'à ce que l'utilisateur ait corrigé toutes les erreurs. Ce réaffichage se fera au travers de requêtes HTTP de type POST. Nous avons donc deux routes différentes pour accéder au même formulaire, une route de type GET, et une route de type POST. Pour nous éviter d'avoir à déclarer deux routes, les concepteurs de SILEX nous proposent d'utiliser la méthode « match() », qui permet la gestion d'une même route pour les 2 types de requêtes GET et POST.

Nous allons immédiatement créer un nouveau template, nous l'appellerons : « albumbd-form-draft.html ». Voici son code source :

```
{% extends "layout.html.twig" %}
{% block content %}

<form method="post">
  <fieldset>
    <legend>Album de BD</legend>

    {{ form_errors(form) }}
    {{ form_row(form.album) }}
    {{ form_row(form.auteur) }}
    {{ form_row(form.editeur) }}
    {{ form_row(form.parution) }}
    <br>
    {{ form_row(form.save, { 'label': 'Enregistrer' }) }}
    {{ form_row(form.reset, { 'label': 'Réinitialiser' }) }}
```

```

</fieldset>
</form>
{% endblock %}

```

Explications : les fonctions `form_errors()` et `form_row()` sont des fonctions fournies par Twig. La première a pour effet de générer une liste de messages d'erreur globale, pour l'ensemble du formulaire. La fonction `form_row()` a pour effet de générer, pour chacun des champs « album », « auteur », « editeur » et « parution », une « div » contenant les éléments suivants :

- le label associé au champ de saisie considéré
- une liste de message d'erreurs spécifique au champ considéré
- le champ de saisie lui-même

La fonction « `form_row()` » avec la valeur « `form.save` » va générer un bouton de validation de type « `submit` » et la valeur « `form.reset` » va générer un bouton de type « `reset` ».

NB : il existe dans Twig beaucoup d'autres fonctions relatives à la gestion de formulaire, c'est un composant très riche et je vous encourage à étudier la documentation de ce composant, et éventuellement à suivre des MOOC ou des tutoriaux spécifiques à ce composant.

Je rappelle que le code de départ de notre nouvelle route est le suivant :

```

$app->match('/albumupdate-draft/{id}', function (Request $request,
Silex\Application $app) {

    // TODO : nous ajouterons le code de génération de formulaire ici

})
->bind('albumupdate-draft');

```

Dans la route de la page précédente (albumupdate-draft), juste au dessus de notre TODO, nous allons ajouter le code suivant :

```
$form = $app['form.factory']->createBuilder(FormType::class)
->add('album', TextType::class, array(
    'constraints' => array(new Assert\NotBlank(),
        new Assert\Length(array('min' => 2)),
        new Assert\Length(array('max' => 30))
    ),
    'attr' => array('class'=>'form-control')
))
->add('auteur', TextType::class, array(
    'constraints' => array(new Assert\NotBlank(),
        new Assert\Length(array('min' => 2)),
        new Assert\Length(array('max' => 30))
    ),
    'attr' => array('class'=>'form-control')
))
->add('editeur', TextType::class, array(
    'constraints' => array(new Assert\NotBlank(),
        new Assert\Length(array('min' => 2)),
        new Assert\Length(array('max' => 30))
    ),
    'attr' => array('class'=>'form-control')
))
->add('parution', DateType::class, array(
    'constraints' => array(new Assert\NotBlank()),
    'attr' => array('class'=>'form-control'),
    'widget' => 'single_text',

    // do not render as type="date", to avoid HTML5 date pickers
    'html5' => true,

    // add a class that can be selected in JavaScript
    //      'attr' => ['class' => 'js-datepicker'],
))
->add('save', SubmitType::class, array(
    'attr' => array('label' => 'Enregistrer', 'class'=>'btn btn-success'),
))
->add('reset', ResetType::class, array(
    'attr' => array('label' => 'Effacer', 'class'=>'btn btn-default'),
))
->getForm();
```

J'écrivais précédemment que la génération du formulaire ne s'était pas faite par magie... c'est bien le code ci-dessus qui va conditionner la manière dont les champs de saisie du formulaire vont fonctionner.

Pour vous aider à comprendre :

Nous commençons par créer à la volée un objet \$form, en nous appuyant sur le service « form.factory » :

```
$form = $app['form.factory']->createBuilder(FormType::class)
```

Nous enchaînons tout de suite avec un premier appel à la méthode « add() », pour définir les caractéristiques du champ de formulaire « album » :

```
->add('album', TextType::class, array(
    'constraints' => array(
        new Assert\NotBlank(),
        new Assert\Length(array('min' => 2)),
        new Assert\Length(array('max' => 30))
    ),
    'attr' => array('class'=>'form-control')
))
```

Le premier paramètre de la méthode « add » permet de définir le nom du champ que nous voulons créer, ici c'est « album ».

La classe TextType indique qu'il s'agit d'un champ de type « texte ».

Le tableau associatif qui suit permet de définir les contraintes (en anglais : « constraints ») qui s'appliquent à ce champ de saisie.

L'objet assertion « NotBlank » indique qu'il s'agit d'un champ obligatoire, et les assertions « Length » permettent de définir la longueur minimale et maximale du champ.

Enfin, le poste de tableau « attr » permet de définir précisément un ou plusieurs attributs du champ de saisie généré. Dans notre exemple, nous appliquons la classe CSS « form-control » qui est une classe de Bootstrap :

```
'attr' => array('class'=>'form-control')
```

Attention, ça se corse un peu, donc faites une pause si vous le jugez utile, avant de lire ce qui suit.

Juste au dessus de notre TODO, nous allons ajouter une nouvelle portion de code. Cette portion de code ne va s'exécuter que dans le cas d'un premier affichage du formulaire, c'est donc le seul moment où une requête HTTP de type GET va être utilisée pour ce formulaire (dont je rappelle que nous avons décidé de le faire fonctionner avec la méthode POST).

Avant de lire le « vrai code », je vous propose de lire une version « ultralite » :

```
if($request->getMethod() == 'GET'){
    // Cas d'un premier affichage
    // On récupère l'ID reçu via la requête http, on regarde
    // si cet ID existe dans la base, et si c'est le cas on
    // charge un tableau $data que l'on transmettra au template
    // Si l'ID n'existe pas alors redirection vers un template d'avertissement
}
```

Quand j'écris que nous cherchons un « ID » (identifiant) dans la base, en fait je triche un peu, car pour l'instant nous faisons cette recherche dans notre tableau contenant une liste de bandes dessinées.

Voici maintenant le vrai code :

```
if($request->getMethod() == 'GET'){
    $id = (int)$request->get('id');
    $data = [] ;
    require_once 'tempdata/liste_bd_temp.php';
    $albums = getListeBD();

    if (!array_key_exists($id, $albums)) {
        // redirection vers une route spécifique si l'album n'existe pas
        return $app->redirect($app['url_generator']
            ->generate('albumnotfound'));
    } else {
        // Copie des données de l'album dans le tableau qui servira
        // à "peupler" le formulaire
        foreach($albums[$id] as $key=>$value) {
            $data[$key] = $app->escape($value);
        };
    }
}
```

Dès que l'utilisateur a fini de remplir le formulaire pour la première fois, il clique sur le bouton « Valider ». C'est alors une requête HTTP de type POST qui est

envoyée par le navigateur au serveur. Voici un rapide descriptif de ce qui va se passer du côté du serveur :

```
if($request->getMethod() == 'POST'){
    // Cas où le formulaire a été soumis
    // Si aucune anomalie, et si l'utilisateur a validé la modification
    // alors on passe à la mise à jour de la base de données (pas
    // implémenté dans l'exemple)
    // Si présence d'anomalies, alors on réaffiche le formulaire avec
    // les messages d'erreur
}
```

Voici maintenant le « vrai » code correspondant au traitement d'une requête « entrante » de type POST :

```
if ($request->getMethod() == 'POST') {
    // les données envoyées par le formulaire sont réinjectées
    // dans le nouveau objet $form
    $form->handleRequest($request);
    // les données du formulaire sont récupérées dans un tableau
    // pour traitement ultérieur
    $data = $form->getData();
    // si le formulaire a été soumis et qu'aucune anomalie n'a été détectée
    if ($form->isSubmitted()) {
        // redirection si Return ou Si Formulaire valide
        if (isset($_POST['form']['return'])) {
            return $app->redirect($app['url_generator']
                ->generate('listebd-crud'));
        }
        if ($form->isValid()) {
            return $app->redirect($app['url_generator']
                ->generate('albumbdregister'));
        }
    } else {
        // si formulaire non soumis ou si anomalie, c'est reparti pour un tour
        error_log('formulaire BAD :(');
        error_log(var_export($data, true));
    }
}
```

Dans une version précédente de ce même code, que j'avais écrite un peu trop vite, j'avais regroupé le test des méthodes `$form->isSubmitted` et `$form->isValid` sur un seul « if ». Mais c'était une erreur, car du coup je n'étais pas en mesure de traiter le cas particulier de l'utilisateur qui souhaite quitter le formulaire sans le valider (via le bouton « return »). En substance, cela donnait le code ci-dessous (vous pouvez le tester pour voir les problèmes qu'il pose, mais attention, le code « valide » est celui qui précède) :


```

if ($form->isSubmitted() && $form->isValid()) { // BAD !!!
    if (isset($data['return'])) {
        return $app->redirect($app['url_generator']
            ->generate('albumbdregister'));
    } else {
        return $app->redirect($app['url_generator']
            ->generate('albumbdregister'));
    }
} else {
    // si formulaire non soumis ou si anomalie, c'est reparti pour un tour
    error_log('formulaire BAD :(');
    error_log(var_export($data, true));
}

```

Nous pouvons maintenant supprimer notre ligne « TODO » et insérer la dernière portion de code :

```

// Affichage ou réaffichage du formulaire
return $app['twig']->render(
    'albumbd-form-draft.html.twig',
    array(
        'form' => $form->createView(),
        'data' => $data
    ));

```

Cette dernière portion de code est intéressante :

- c'est ici que nous définissons le template Twig à utiliser
- mais surtout nous passons deux éléments à notre template :
 - un formulaire qui est généré par l'objet \$form et sa méthode « createView() »
 - un tableau associatif « data » qui devrait nous permettre de « préalimenter » les champs de saisie du formulaire avec l'album de bande dessinée sélectionnée par l'utilisateur. En fait, cette préalimentation ne fonctionne pas du tout en l'état.

Il est temps de tester notre nouvelle route, nous pouvons le faire en saisissant l'URL suivante dans notre navigateur :

<http://localhost/firstsilex/web/albumupdate-draft/2>

Ouch ! Il semblerait que nous ayons oublié quelques chose :

Whoops, looks like something went wrong.

1/1 InvalidArgumentException in FormRegistry.php line 87:
Could not load type "FormType"

```
1. in FormRegistry.php line 87
2. at FormRegistry->getType('FormType') in FormFactory.php line 67
3. at FormFactory->createBuilder('FormType') in controllers.php line 226
4. at {closure}(object(Request), object(Application))
5. at call_user_func_array(object(Closure), array(object(Request), object(Application))) in HttpKernel.php line 153
6. at HttpKernel->handleRaw(object(Request), '1') in HttpKernel.php line 68
7. at HttpKernel->handle(object(Request), '1', true) in Application.php line 496
8. at Application->handle(object(Request)) in Application.php line 477
9. at Application->run() in index.php line 22
```

Ce message d'erreur est intéressant. Il nous indique concrètement que nous avons utilisé un objet « FormType » que notre application SILEX ne connaît pas.

Pour y remédier, nous allons ajouter au début du script src/controllers.php, les lignes en gras suivantes :

```
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpFoundation\RedirectResponse;
use Symfony\Component\HttpKernel\Exception\NotFoundHttpException;

use Symfony\Component\Form\Extension\Core\Type\FormType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\HiddenType;
use Symfony\Component\Form\Extension\Core\Type\DateType;
use Symfony\Component\Form\Extension\Core\Type\ButtonType;
use Symfony\Component\Form\Extension\Core\Type\ResetType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
//use Symfony\Component\Form\Extension\Core\Type\EmailType;
//use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
use Symfony\Component\Validator\Constraints as Assert;
```

La première ligne ajoutée correspond à l'objet FormType indiqué dans le message d'erreur, mais il nous faut aussi ajouter tous les types de champs que nous allons utiliser dans le formulaire. J'ai mis 2 lignes en commentaire, pour deux types de champs que n'utilisons pas pour le moment.

Essayez maintenant de resaisir l'URL suivante dans le navigateur :

<http://localhost/firstsilex/web/albumupdate-draft/2>

Normalement, vous devriez voir apparaître notre fameux formulaire « brouillon » :

The screenshot shows a web application interface for 'SILEX Project'. At the top is a dark navigation bar with links: 'FirstSilex Project', 'Accueil', and 'Travaux'. Below the navigation bar, the page title 'SILEX Project' is displayed. The main content area is titled 'Album de BD' and contains a form with the following fields: 'Album', 'Auteur', 'Editeur', and 'Parution'. The 'Parution' field is pre-filled with the text 'jj/mm/aaaa'. At the bottom of the form are two buttons: 'Enregistrer' (green) and 'Réinitialiser' (white).

Les champs de saisie de type « date »

Les champs de saisie « album », « auteur » et « editeur » fonctionnent de la même façon, mais le champ « parution » est un champ de type « date ». Il a donc des caractéristiques un peu différentes :

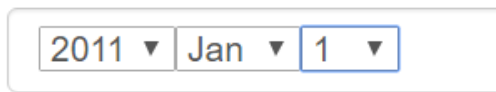
```
->add('parution', DateType::class, array(
    'constraints' => array(new Assert\NotBlank()),
    'attr' => array('class'=>'form-control'),
    'widget' => 'single_text',

    // do not render as type="date", to avoid HTML5 date pickers
    'html5' => true,

    // add a class that can be selected in JavaScript
    //      'attr' => ['class' => 'js-datepicker'],
))
```

Par défaut les champs de type « date » générés par Twig ressemblent à ceci :

Parution



Mais si l'on préfère désactiver ce comportement par défaut, on peut utiliser l'attribut suivant :

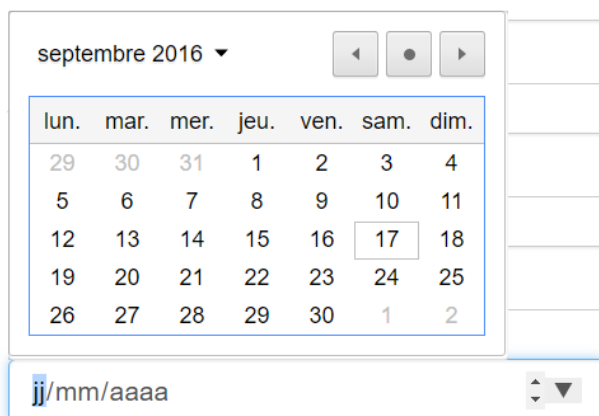
```
'widget' => 'single_text',
```

Dans ce cas, on pourra recourir au champ de type « date » défini dans la norme HTML5, en indiquant « true » sur le paramètre « html5 » :

```
'html5' => true,
```

Si Chrome gère bien les champs de saisie de type « date », en proposant un affichage de type calendrier, comme assistance à la saisie :

Album



lun.	mar.	mer.	jeu.	ven.	sam.	dim.
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2

... en revanche, Firefox ne propose pas d'affichage équivalent, ce qui est fort regrettable, car nous souhaitons proposer aux utilisateurs une saisie de date homogène pour l'ensemble des navigateurs. Nous avons donc tout intérêt à recourir à un plugin Javascript de type « datepicker ». jQueryUI, qui est une librairie de composants complémentaire à jQuery, propose justement un plugin de ce type. La mise en œuvre de ce plugin est en dehors du périmètre de ce dossier consacré à l'étude de SILEX. Mais on retrouvera en annexe un chapitre décrivant comment mettre en œuvre ce plugin dans SILEX, en particulier pour les champs de type date.

Notre formulaire présente un défaut : les champs de saisie sont vides, même si l'album de BD transmis en paramètre de l'URL existe bien dans notre tableau des albums de bande dessinée. Nous verrons dans le chapitre qui suit comment corriger ce problème.

Pour l'heure, je vous recommande d'enregistrer vos dernières modifications dans l'historique de Git, et de faire une pause.

Quelques dossiers intéressants concernant la gestion de formulaires avec Twig et Symfony :

<https://www.sitepoint.com/building-processing-forms-in-symfony-2/>

http://brentertainment.com/other/docs/cookbook/form/twig_form_customization.html

<https://knpuniversity.com/screencast/symfony2-ep2/form-submit>

<http://silex.sensiolabs.org/doc/providers/form.html>

http://symfony.com/doc/current/form/form_customization.html

http://symfony.com/doc/current/reference/forms/twig_reference.html

<http://symfony.com/doc/current/forms.html>

4.2. Le CRUD

Notre objectif dans ce chapitre, c'est de monter un petit module de gestion des albums de bande dessinée, avec des options de consultation, création, modification, suppression. C'est ce que les développeurs anglophones désignent sous le terme de CRUD, pour « Create, Retrieve, Update, Delete ».

Voici une copie d'écran du produit fini :

Albums de BD (CRUD)

Créer					
#	Titre	Auteur(s)	Editeur	Parution	
1	Garulfo	Ayroles, Maïorana, Leprévost	Delcourt	15/05/2011	Afficher Modifier Supprimer
2	Horologiom	Fabrice Lebeault	Delcourt	01/01/2005	Afficher Modifier Supprimer
3	Le château des étoiles	Alex Alice	Rue De Sevres	01/05/2014	Afficher Modifier Supprimer
4	Le voyage extraordinaire	Camboni, Filippi	Vents D'ouest	01/09/2012	Afficher Modifier Supprimer

Pour créer ce CRUD, nous avons pas mal de choses à faire... je vous propose de commencer par créer un nouveau template dans le répertoire « templates », et de l'appeler « albumbd-form.html.twig ». Nous allons nous efforcer de créer un seul formulaire, qui nous servira pour la consultation, la modification, la création, et la suppression.

Pourquoi créer un formulaire pour la consultation ? On pourrait en effet se contenter d'afficher les informations, par exemple dans un tableau HTML. Mais dans les applications de gestion, il est courant de proposer aux utilisateurs le même affichage de type formulaire en consultation et en modification... En revanche, dans le cas de la consultation, on fait en sorte que les champs du formulaire ne soient pas modifiables (on dit dans ce cas qu'ils sont « verrouillés »).

Et pourquoi un créer un formulaire pour la suppression ? Toujours dans un contexte d'application de gestion, il est courant de proposer aux utilisateurs un écran de confirmation de suppression, pour qu'ils puissent s'assurer qu'ils ont sélectionné la bonne donnée avant de valider la suppression. Quoi de plus

simple, dans ce cas, que d'afficher le même formulaire, toujours en verrouillant les champs de saisie.

Bon, assez de blabla... Action !

Commencez par créer une nouvelle option de menu dans le menu « travaux » du template « layout.html.twig » :

```
<li><a href="{{ path('listebd-crud') }}">Albums de BD (CRUD)</a></li>
```

Créer le template « listebd-crud » en dupliquant le template « listebd », puis modifiez le tableau HTML en ajoutant une colonne dans l'entête, et une colonne dans le corps du tableau :

```
{% extends "layout.html.twig" %}

{% block content %}
<div class="listebd_crud">
    <h2>Albums de BD (CRUD)</h2>
    <a class="btn btn-success btn-sm" href="{{ path('albumupdate', {'id':0, 'crud':'C'}) }}">Créer</a>

    <table class="table table-striped">
        <thead>
            <tr>
                <th>#</th>
                <th>Titre</th>
                <th>Auteur(s)</th>
                <th>Editeur</th>
                <th>Parution</th>
                <th>&nbsp;</th>
            </tr>
        </thead>
        <tbody>
            {% for album in listebd %}
            <tr>
                <td>{{ album.id }}</td>
                <td>{{ album.album|capitalize }}</td>
                <td>{{ album.auteur|title }}</td>
                <td>{{ album.editeur|title }}</td>
                <td>{{ album.parution|date("d/m/Y") }}</td>
```

```

        <td>
            <a class="btn btn-default btn-sm"
href="{{ path('albumupdate', {'id':album.id, 'crud':'R'}) }}" >Afficher</a>
            <a class="btn btn-success btn-sm"
href="{{ path('albumupdate', {'id':album.id, 'crud':'U'}) }}" >Modifier</a>
            <a class="btn btn-warning btn-sm"
href="{{ path('albumupdate', {'id':album.id, 'crud':'D'}) }}" >Supprimer</a>
        </td>
    </tr>
    {% endfor %}
</tbody>
</table>
</div>
{% endblock %}

```

Vous avez sans doute remarqué que l'on fait appel à la même route pour chacune des 4 opérations de base du CRUD :

```
path('albumupdate', {'id':album.id, 'crud':'R'})
```

Créez une nouvelle route permettant l'affichage du template créé précédemment :

```

$app->get('/liste-bd-crud/', function () use ($app) {
    require_once 'tempdata/liste_bd_temp.php';

    return $app['twig']->render('liste-bd-crud.html.twig',
        array(
            'liste-bd' => getListeBD()
        ));
})
->bind('liste-bd-crud')
;

```

Nous en profitons pour créer deux nouvelles routes :

- La première route nous servira à renvoyer l'utilisateur vers une page d'avertissement s'il a sélectionné un ID de livre incorrect
- La seconde route nous permettra d'afficher une page avertissant l'utilisateur du fait que sa modification a été prise en compte

Voici le code de la première des deux nouvelles routes :

```
$app->get('/albumbd-not-found/', function () use ($app) {

    return $app['twig']->render('albumbdnotfound.html.twig', array());
})
->bind('albumotfound')
;
```

Le template correspondant à 'albumbdnotfound.html.twig' :

```
{% extends "layout.html.twig" %}

{% block content %}
<div class="jumbotron">
    L'album de BD que vous avez demandé n'existe pas (ou plus)
    dans notre base de données. <br>
    Nous vous prions de bien vouloir nous en excuser.
</div>
{% endblock %}
```

Voici le code de la seconde route qui – je le rappelle ici - nous permettra d’afficher une page avertissant l’utilisateur du fait que sa modification a été prise en compte :

```
$app->get('/albumbd-register/', function () use ($app) {

    return $app['twig']->render('albumbdregister.html.twig', array());
})
->bind('albumbdregister')
```

Voici le template associé (albumbdregister.html.twig):

```
{% extends "layout.html.twig" %}

{% block content %}
<div class="jumbotron">
    Votre demande de modification d'un album de BD a été prise en compte.<br>
    Nous allons étudier votre demande et nous reviendrons vers vous dans les
    plus brefs délais.
</div>
{% endblock %}
```

Voici le code source complet du template « alumbd-form.html.twig » :

```
{% extends "layout.html.twig" %}
{% block content %}
    <form method="post" novalidate >
        <fieldset>
            {% set form_legend = '' %}
            {% if data.crud == 'C' %}
                {% set form_legend = 'Création' %}
            {% endif %}
            {% if data.crud == 'R' %}
                {% set form_legend = 'Consultation' %}
            {% endif %}
            {% if data.crud == 'U' %}
                {% set form_legend = 'Mise à jour' %}
            {% endif %}
            {% if data.crud == 'D' %}
                {% set form_legend = 'Confirmation de suppression' %}
            {% endif %}
            <legend>{{form_legend}}</legend>

            {{ form_widget(form.crud, {'value':data.crud}) }}
            {{ form_widget(form.id, {'value':data.id}) }}

            {{ form_errors(form) }}

            <div class="form-group">
                {{ form_label(form.album, "Titre", {'label_attr':
                    {'class': 'control-label'}}) }}
                {{ form_errors(form.album) }}
                {{ form_widget(form.album, {'value':data.album}) }}
            </div>
            <div class="form-group">
                {{ form_label(form.editeur, "Editeur", {'label_attr':
                    {'class': 'control-label'}}) }}
                {{ form_errors(form.editeur) }}
                {{ form_widget(form.editeur, {'value':data.editeur}) }}
            </div>

            <div class="form-group">
                {{ form_label(form.auteur, "Auteur", {'label_attr':
                    {'class': 'control-label'}}) }}
                {{ form_errors(form.auteur) }}
                {{ form_widget(form.auteur, {'value':data.auteur}) }}
            </div>
            <div class="form-group">
                {{ form_label(form.parution, "Parution", {'label_attr':
                    {'class': 'control-label'}}) }}
                {{ form_errors(form.parution) }}
                {{ form_widget(form.parution, {'value':data.parution}) }}
            </div>
        </form>
    </div>
```

```

<div class="form-inline">
    {% if form.return is defined %}
        {{ form_widget(form.return, {'label':'Retour'}) }}
    {% endif %}
    {% if form.save is defined %}
        {{ form_widget(form.save, {'label':'Valider'}) }}
    {% endif %}
    {% if form.reset is defined %}
        {{ form_widget(form.reset, {'label':'Effacer'}) }}
    {% endif %}
</div>
</fieldset>
</form>
{% endblock %}

```

Vous avez sans doute remarqué que, lorsqu'on sélectionne un « album de BD » qui existe dans notre base fictive, les champs du formulaire sont alimentés en conséquence. On dit que le formulaire est « peuplé » (en anglais : « populated ») à partir de données transmises par la couche « modèle ». C'est l'attribut « value » qui va permettre de pré-alimenter le champ de formulaire à partir de la donnée correspondante se trouvant dans le tableau « data » :

```

{{ form_widget(form.album, {'value':data.album}) }}

```

Dans cet exemple de module de type CRUD, j'ai créé une classe spécifique pour définir les caractéristiques du formulaire. Je me suis appuyé sur des exemples trouvés dans plusieurs tutoriels, mais je me suis finalement écarté des modèles proposés, car j'ai constaté qu'aucun des exemples trouvés ne fonctionnait convenablement. En effet, des modifications ont été apportées au framework Symfony, modifications qui ont eu des répercussions sur SILEX, rendant certains tutoriaux obsolètes. Plusieurs développeurs ont rencontré des difficultés avec ces modifications, comme j'ai pu le constater en consultant les forums.

Voici des informations trouvées à ce sujet dans quelques forums :

Message d'erreur rencontré : « Silex FormServiceProvider could not load type "form" when using Symfony 3 components »

Quelques liens vers des posts où le problème est mentionné :

<https://github.com/silexphp/Silex/issues/1302>

<http://stackoverflow.com/questions/34663227/silex-formserviceprovider-could-not-load-type-form-when-using-symfony3-compone>

You can actually use Symfony Form 3 with Silex, but createBuilder function now requires you to adhere to the new way of passing in Types as arguments. Silex documentation has not been updated to reflect this.

In pre 2.8, types used to be passed in as strings such as 'form', 'text', 'email' and so on and Symfony components resolved it to the proper class. Now you have to pass in a classname instead.

So, what used to be form now becomes `Symfony\Component\Form\Extension\Core\Type\FormType::class`. And text becomes `Symfony\Component\Form\Extension\Core\Type\TextType::class`. Of course, you can import these classes so that you don't have to use the full namespace.

Voici le code source de la classe AlbumbdForm, qui se trouve dans src/Form/AlbumbdForm.php :

```
namespace Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Validator\Constraints as Assert;

use Symfony\Component\Form\Extension\Core\Type\FormType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\HiddenType;
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\Form\Extension\Core\Type\DateType;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
use Symfony\Component\Form\Extension\Core\Type\ButtonType;
use Symfony\Component\Form\Extension\Core\Type\ResetType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

class AlbumbdForm {

    protected $form ;
    protected $crud_id;
    protected $crud_context;
    protected $default_attrs;

    public function __construct($app, $request) {
        $this->form = $app['form.factory']->createBuilder(FormType::class);

        if($request->getMethod() == 'GET'){
            $this->crud_id = (int)$request->get('id');
            $this->crud_context = strip_tags($request->get('crud'));
        } else {
            if($request->getMethod() == 'POST'){
                error_log(var_export($_POST, true));
                if (isset($_POST['form']['id'])) {
                    $this->crud_id = (int)$_POST['form']['id'];
                } else {
                    throw new \Exception('Id absent du formulaire') ;
                }
                if (isset($_POST['form']['crud'])) {
                    $this->crud_context =
                        strip_tags($_POST['form']['crud']);
                } else {
                    throw new \Exception(
                        'Contexte CRUD absent du formulaire') ;
                }
            } else {
                throw new \Exception(
                    'Le type de requête HTTP est incorrect') ;
            }
        }
    }
}
```

```

        $this->crud_context = strtoupper($this->crud_context);

        if ($this->crud_context == 'C') {
            $this->crud_id = 0 ;
        }

        // Classe CSS par défaut (Bootstrap)
        $this->default_attrs = array('class'=>'form-control');

        $this->setAttributes();
        $this->genFields();
        $this->genButtons();
    }

    /**
     * Définit les attributs HTML à appliquer par défaut à l'ensemble
     * des champs du formulaire (selon le contexte d'exécution)
     * @throws \Exception
     */
    protected function setAttributes() {
        // Ajout d'attributs HTML complémentaires selon le contexte
        switch ($this->crud_context) {
            case 'C': {
                break;
            }
            case 'R': {
                $this->default_attrs['disabled'] = 'disabled';
                break;
            }
            case 'U': {
                break;
            }
            case 'D': {
                $this->default_attrs['disabled'] = 'disabled';
                break;
            }
            default: {
                throw new \Exception(
                    'Le parametre crud_context est incorrect' ) ;
            }
        }
    }

    /**
     * Génération des champs du formulaire
     */
    protected function genFields() {
        if ($this->crud_context == 'U' || $this->crud_context == 'C') {
            $this->form
                ->add('crud', HiddenType::class, array(
                    'constraints' => array(new Assert\NotBlank())
                ))
        }
    }

```

```

->add('id', HiddenType::class, array(
    'constraints' => array(new Assert\NotBlank())
))
->add('album', TextType::class, array(
    'constraints' => array(new Assert\NotBlank(),
        new Assert\Length(array('min' => 2)),
        new Assert\Length(array('max' => 30))
    ),
    'attr' => $this->default_attrs
))

->add('auteur', TextType::class, array(
    'constraints' => array(new Assert\NotBlank(),
        new Assert\Length(array('min' => 2)),
        new Assert\Length(array('max' => 30))
    ),
    'attr' => $this->default_attrs
))
->add('editeur', TextType::class, array(
    'constraints' => array(new Assert\NotBlank(),
        new Assert\Length(array('min' => 2)),
        new Assert\Length(array('max' => 30))
    ),
    'attr' => $this->default_attrs
));
// ajout de la classe CSS js-datepicker pour le champ suivant
$field_attrs = $this->default_attrs;
$field_attrs['class'] .= ' js-datepicker';
$this->form
->add('parution', DateType::class, array(
    'constraints' => array(new Assert\NotBlank()),
    'widget' => 'single_text',

    // don't render as type="date", to avoid HTML5 date pickers
    'html5' => false,

    // add a class that can be selected in JavaScript
    'attr' => $field_attrs,
))
;
} else {
    // Si Consultation ou Suppression, affichage
    // des champs verrouillés et sans contrôle
    $this->form
    ->add('crud', HiddenType::class, array(
    ))
    ->add('id', HiddenType::class, array(
    ))
    ->add('album', TextType::class, array(
        'attr' => $this->default_attrs
    ))
}

```

```

        ->add('auteur', TextType::class, array(
            'attr' => $this->default_attrs
        ))
        ->add('editeur', TextType::class, array(
            'attr' => $this->default_attrs
        ))
        ->add('parution', TextType::class, array(
            'attr' => $this->default_attrs
        ))
        ;
    }
}

/**
 * Génération des boutons du formulaire
 */
protected function genButtons() {
    // Définition des boutons de validation selon le contexte
    switch ($this->crud_context) {
        case 'C': {
            $this->form
                ->add('save', SubmitType::class, array(
                    'attr' => array('label' => 'Enregistrer',
                        'class'=>'btn btn-success'),
                ))
                ->add('reset', ResetType::class, array(
                    'attr' => array('label' => 'Effacer',
                        'class'=>'btn btn-default'),
                ))
                ->add('return', SubmitType::class, array(
                    'attr' => array('label' => 'Annuler',
                        'class'=>'btn btn-default'),
                ));
            break;
        }
        case 'R': {
            $this->form
                ->add('return', SubmitType::class, array(
                    'attr' => array('label' => 'Retour',
                        'class'=>'btn btn-default'),
                ));
            break;
        }
        case 'U': {
            $this->form
                ->add('save', SubmitType::class, array(
                    'attr' => array('label' => 'Enregistrer',
                        'class'=>'btn btn-success'),
                ))
    }
}

```



```

        ->add('reset', ResetType::class, array(
            'attr' => array('label' => 'Effacer',
                'class'=>'btn btn-default'),
        ))
        ->add('return', SubmitType::class, array(
            'attr' => array('label' => 'Annuler',
                'class'=>'btn btn-default'),
        ));
        break;
    }
    case 'D': {
        $this->form
        ->add('save', SubmitType::class, array(
            'attr' => array('label' => 'Confirmer',
                'class'=>'btn btn-warning'),
        ))
        ->add('return', SubmitType::class, array(
            'attr' => array('label' => 'Annuler',
                'class'=>'btn btn-default'),
        ));
        break;
    }
};
}

/**
 * Renvoie la fonction getForm() de l'objet $this->form
 * @return object
 */
public function getForm() {
    return $this->form->getForm() ;
}

/**
 * Transmission de l'ID courant si besoin
 * @return integer
 */
public function getId() {
    return $this->crud_id;
}

/**
 * Transmission du contexte courant ("C", "R", "U" ou "D")
 * @return string
 */
public function getContext() {
    return $this->crud_context ;
}
}

```

Voici pour finir le code source de la route « albumupdate » :

```
$app->match('/albumupdate/{id}/{crud}',
    function (Request $request, Silex\Application $app) {

    $albumbdForm = new \Form\AlbumbdForm($app, $request);
    $form = $albumbdForm->getForm();

    if($request->getMethod() == 'GET'){
        $id = (int)$request->get('id');
        $crud = strtoupper(strip_tags($request->get('crud')));
        $data = [] ;
        require_once 'tempdata/liste_bd_temp.php';
        $albums = getListeBD();
        if ($crud == 'C') {
            $data = array(
                'crud' => $crud,
                'id' => 0,
                'album' => '',
                'auteur' => '',
                'editeur' => '',
                'parution' => ''
            );
        } else {
            if (!array_key_exists($id, $albums)) {
                // redirection
                return $app->redirect($app['url_generator']
                    ->generate('albumnotfound'));
            } else {
                foreach($albums[$id] as $key=>$value) {
                    if ($key == 'parution') {
                        // la date de parution doit être formatée selon le
                        // format attendu par le plugin jQuery
                        $tmpdate = new DateTime($value);
                        $data[$key] = $tmpdate->format('Y-m-d');
                    } else {
                        $data[$key] = $app->escape($value);
                    }
                };
                // On ajoute la notion de CRUD à notre jeu de données
                $data['crud'] = $crud ;
            }
        }
    }
}
```

```

if($request->getMethod() == 'POST'){
    $form->handleRequest($request);
    $data = $form->getData();
    if ($form->isSubmitted()) {
        // redirection si Return ou Si Formulaire valide
        if (isset($_POST['form']['return'])) {
            return $app->redirect($app['url_generator']
                                ->generate('listebd-crud'));
        }
        if ($form->isValid()) {
            return $app->redirect($app['url_generator']
                                ->generate('albumbdregister'));
        }
    } else {
        // Date de parution à reformater selon le format défini
        // sur jQueryUI Datepicker
        $data['parution'] = $data['parution']->format('Y-m-d');
    }
}
return $app['twig']->render(
    'albumbd-form.html.twig',
    array(
        'form' => $form->createView(),
        'data' => $data
    ));
})
->bind('albumupdate');

```

Vous remarquerez que dans certains cas, j'ai dû faire appel à un test un peu surprenant : `if (isset($_POST['form']['return']))`

... comme dans l'exemple suivant :

```

if (isset($_POST['form']['return'])) {
    return $app->redirect($app['url_generator']
                        ->generate('listebd-crud'));
} else {
    return $app->redirect($app['url_generator']
                        ->generate('albumbdregister'));
}

```

En effet, les boutons de type « submit » ne font pas partie des données renvoyées par `$form->getData()`. Pour autant, l'information comme quoi l'utilisateur a cliqué sur le bouton « return » est bien présente dans le tableau

`$_POST`. Si cette information est absente de `$_POST`, cela signifie que l'on a cliqué sur le bouton « Valider ».

Quelques remarques par rapport aux boutons de type Submit :

Il est intéressant de souligner que, lorsqu'un formulaire contient plusieurs boutons de type « Submit », seul le bouton de type « Submit » sélectionné par l'utilisateur fera partie du jeu de données renvoyé par le formulaire dans `$_FORM` (ou `$_GET` selon la méthode de formulaire utilisée). C'est la raison pour laquelle un test du genre `if (isset($_POST['form']['return'])) ...` suffit à savoir si c'est le bouton « return » ou l'autre bouton qui a été sélectionné.

Vous trouverez parfois des applications dans lesquelles le bouton « return » permettant de sortir du formulaire sans le valider, est un simple lien déguisé en bouton. Ce lien est dans ce cas rattaché à la page d'origine ayant servi à déclencher l'affichage du formulaire, il aura pour effet de sortir du formulaire par l'intermédiaire d'une requête de type GET. C'est une autre manière de faire qui est aussi pertinente que celle que nous avons étudiée ici. Pour ma part, j'aime bien que toutes les conditions de sortie du formulaire (que ce soit en cliquant sur « return » ou sur l'autre bouton) soient pilotées par le même « point de sortie » que constitue le bloc de code présenté dans la page précédente. C'est à ce bloc de code de déterminer s'il autorise la sortie du formulaire ou pas. Il existe bien évidemment d'autres variantes, mais elles impliquent souvent l'utilisation de Javascript (sujet passionnant qui est en dehors du périmètre de ce tutoriel).

Mais revenons à nos moutons ☺

Normalement, si vous n'avez pas fait d'erreur de syntaxe, l'association de la route « albumupdate » et de la classe `AlbumbdForm` devrait vous permettre de générer les 4 principales fonctions d'un module de type CRUD.

On rappelle que le résultat final attendu est celui-ci :

Commençons par un rappel concernant l'aspect de notre liste de sélection :

Albums de BD (CRUD)

Créer					
#	Titre	Auteur(s)	Editeur	Parution	
1	Garulfo	Ayroles, Maïorana, Leprévost	Delcourt	15/05/2011	Afficher Modifier Supprimer
2	Horologiom	Fabrice Lebeault	Delcourt	01/01/2005	Afficher Modifier Supprimer
3	Le château des étoiles	Alex Alice	Rue De Sevres	01/05/2014	Afficher Modifier Supprimer
4	Le voyage extraordinaire	Camboni, Filippi	Vents D'ouest	01/09/2012	Afficher Modifier Supprimer

Exemple de formulaire obtenu en cliquant sur le bouton « Afficher » de la première ligne (soit l'album qui a pour titre « Garulfo ») :

FirstSilex Project
Accueil
Travaux ▼

SILEX Project

Consultation

Titre

Editeur

Auteur

Parution

Exemple obtenu en cliquant sur le bouton « modifier » de la seconde ligne :

FirstSilex Project Accueil Travaux ▾

SILEX Project

Mise à jour

Titre

Editeur

Auteur

Parution

Exemple obtenu en cliquant sur le bouton « supprimer » de la 3^{ème} ligne :

FirstSilex Project Accueil Travaux ▾

SILEX Project

Confirmation de suppression

Titre

Editeur

Auteur

Parution

Exemple obtenu en cliquant sur le bouton « créer » qui se trouve au dessus de liste des albums :

FirstSilex Project

Accueil

Travaux ▾

SILEX Project

Création

Titre

Editeur

Auteur

Parution

Retour

Valider

Effacer

Voilà, nous avons un module CRUD complet, c'est cool 😊.

Mais si on veut pouvoir réutiliser ce principe sur d'autres données de notre application, et que l'on veut éviter autant que possible toute redondance de code, alors nous avons intérêt à nous pencher sur la classe AlbumbdForm, et à essayer de voir si nous pouvons en extraire des éléments réutilisables pour une classe plus « générique ». C'est l'objet du chapitre qui suit.

4.3. Refactoring du code

Dans le chapitre précédent, nous avons créé une classe `AlbumbdForm`, sans nous préoccuper de savoir si notre code pouvait être réutilisé pour d'autres formulaires. Il est maintenant de procéder à un peu de refactorisation, et de créer une classe générique qui pourra être réutilisée pour d'autres modules CRUD de notre application. D'ailleurs, nous allons appeler cette classe générique « `CrudstdForm` ».

Voici le code de la classe `CrudstdForm` :

```
<?php

namespace Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Validator\Constraints as Assert;

use Symfony\Component\Form\Extension\Core\Type\FormType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\HiddenType;
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\Form\Extension\Core\Type\DateType;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
use Symfony\Component\Form\Extension\Core\Type\ButtonType;
use Symfony\Component\Form\Extension\Core\Type\ResetType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

/**
 * Description of CrudstdForm
 *
 * @author Gregory Jarrige
 * @version 0.1 (2016-09-18)
 */
class CrudstdForm {

    protected $form ;
    protected $crud_id;
    protected $crud_context;
    protected $default_attrs;

    public function __construct($app, $request) {

        $this->form = $app['form.factory']->createBuilder(FormType::class);

        if($request->getMethod() == 'GET'){
            $this->crud_id = (int)$request->get('id');
```



```

        $this->crud_context = strip_tags($request->get('crud'));
    } else {
        if($request->getMethod() == 'POST'){
            if (isset($_POST['form']['id'])) {
                $this->crud_id = (int)$_POST['form']['id'];
            } else {
                throw new \Exception('Id absent du formulaire') ;
            }
            if (isset($_POST['form']['crud'])) {
                $this->crud_context = strip_tags($_POST['form']['crud']);
            } else {
                throw new \Exception('Contexte CRUD absent du formulaire') ;
            }
        } else {
            throw new \Exception('Le type de requête HTTP est incorrect') ;
        }
    }

    $this->crud_context = strtoupper($this->crud_context);

    if ($this->crud_context == 'C') {
        $this->crud_id = 0 ;
    }

    // Classe CSS par défaut (Bootstrap)
    $this->default_attrs = array('class'=>'form-control');

    // Définition des attributs par défaut
    $this->setAttributes();

    // Définition des champs cachés "crud" et "id"
    if ($this->crud_context == 'U' || $this->crud_context == 'C') {
        $this->form
            ->add('crud', HiddenType::class, array(
                'constraints' => array(new Assert\NotBlank())
            ))
            ->add('id', HiddenType::class, array(
                'constraints' => array(new Assert\NotBlank())
            ))
        ;
    } else {
        // Si Consultation ou Suppression, affichage
        // des champs verrouillés et sans contrôle
        $this->form
            ->add('crud', HiddenType::class, array(
            ))
            ->add('id', HiddenType::class, array(
            ))
        ;
    }
    $this->genFields();
    $this->genButtons();
}

```

```

/**
 * Définit les attributs HTML à appliquer par défaut à l'ensemble
 * des champs du formulaire (selon le contexte d'exécution)
 * @throws \Exception
 */
protected function setAttributes() {
    // Ajout d'attributs HTML complémentaires selon le contexte
    switch ($this->crud_context) {
        case 'C': {
            break;
        }
        case 'R': {
            $this->default_attrs['disabled'] = 'disabled';
            break;
        }
        case 'U': {
            break;
        }
        case 'D': {
            $this->default_attrs['disabled'] = 'disabled';
            break;
        }
        default: {
            throw new \Exception('Le parametre crud_context est incorrect') ;
        }
    }
}
/**
 * Génération des champs du formulaire
 * Cette méthode doit impérativement être "overridee" dans les classes
 * filles
 */
protected function genFields() {
    if ($this->crud_context == 'U' || $this->crud_context == 'C') {
        $this->form
        // champ exemple (à remplacer)
        ->add('xxxxx', TextType::class, array(
            'constraints' => array(new Assert\NotBlank(),
                new Assert\Length(array('min' => 2)),
                new Assert\Length(array('max' => 30))
            ),
            'attr' => $this->default_attrs
        ))
        ;
    } else {
        // Si Consultation ou Suppression, affichage
        // des champs verrouillés et sans contrôle
        $this->form
        // champ exemple (à remplacer)
        ->add('xxxxx', TextType::class, array(
            'attr' => $this->default_attrs
        ))
    }
}

```

```

        ;
    }
}

/**
 * Génération des boutons du formulaire
 */
protected function genButtons() {
    // Définition des boutons de validation selon le contexte
    switch ($this->crud_context) {
        case 'C': {
            $this->form
                ->add('save', SubmitType::class, array(
                    'attr' => array('label' => 'Enregistrer',
                                    'class'=>'btn btn-success'),
                ))
                ->add('reset', ResetType::class, array(
                    'attr' => array('label' => 'Effacer',
                                    'class'=>'btn btn-default'),
                ))
                ->add('return', SubmitType::class, array(
                    'attr' => array('label' => 'Annuler',
                                    'class'=>'btn btn-default'),
                ));
            break;
        }
        case 'R': {
            $this->form
                ->add('return', SubmitType::class, array(
                    'attr' => array('label' => 'Retour',
                                    'class'=>'btn btn-default'),
                ));
            break;
        }
        case 'U': {
            $this->form
                ->add('save', SubmitType::class, array(
                    'attr' => array('label' => 'Enregistrer',
                                    'class'=>'btn btn-success'),
                ))
                ->add('reset', ResetType::class, array(
                    'attr' => array('label' => 'Effacer',
                                    'class'=>'btn btn-default'),
                ))
                ->add('return', SubmitType::class, array(
                    'attr' => array('label' => 'Annuler',
                                    'class'=>'btn btn-default'),
                ));
            break;
        }
    }
}

```

```

        case 'D': {
            $this->form
                ->add('save', SubmitType::class, array(
                    'attr' => array('label' => 'Confirmer',
                                    'class'=>'btn btn-warning'),
                ))
                ->add('return', SubmitType::class, array(
                    'attr' => array('label' => 'Annuler',
                                    'class'=>'btn btn-default'),
                ));
            break;
        }
    };
}

/**
 * Renvoie la fonction getForm() de l'objet $this->form
 * @return object
 */
public function getForm() {
    return $this->form->getForm() ;
}

/**
 * Transmission de l'ID courant si besoin
 * @return integer
 */
public function getId() {
    return $this->crud_id;
}

/**
 * Transmission du contexte courant ("C", "R", "U" ou "D")
 * @return string
 */
public function getContext() {
    return $this->crud_context ;
}
}

```

Et voici maintenant le code source de la classe AlbumbdForm :

```

<?php

namespace Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Validator\Constraints as Assert;

use Symfony\Component\Form\Extension\Core\Type\FormType;

```

```

use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\HiddenType;
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\Form\Extension\Core\Type\DateType;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
use Symfony\Component\Form\Extension\Core\Type\ButtonType;
use Symfony\Component\Form\Extension\Core\Type\ResetType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

/**
 * Description of AlumbdForm
 *
 * @author Gregory Jarrige
 * @version 0.1 (2016-09-18)
 */
class AlumbdForm extends \Form\CrudstdForm {

    /**
     * Génération des champs du formulaire
     */
    protected function genFields() {
        if ($this->crud_context == 'U' || $this->crud_context == 'C') {
            $this->form
                ->add('crud', HiddenType::class, array(
                    'constraints' => array(new Assert\NotBlank())
                ))
                ->add('id', HiddenType::class, array(
                    'constraints' => array(new Assert\NotBlank())
                ))
                ->add('album', TextType::class, array(
                    'constraints' => array(new Assert\NotBlank(),
                        new Assert\Length(array('min' => 2)),
                        new Assert\Length(array('max' => 30))
                    ),
                    'attr' => $this->default_attrs
                ))
                ->add('auteur', TextType::class, array(
                    'constraints' => array(new Assert\NotBlank(),
                        new Assert\Length(array('min' => 2)),
                        new Assert\Length(array('max' => 30))
                    ),
                    'attr' => $this->default_attrs
                ))
                ->add('editeur', TextType::class, array(
                    'constraints' => array(new Assert\NotBlank(),
                        new Assert\Length(array('min' => 2)),
                        new Assert\Length(array('max' => 30))
                    ),
                    'attr' => $this->default_attrs
                ));
            // ajout de la classe CSS js-datepicker pour le champ suivant
            $field_attrs = $this->default_attrs;
            $field_attrs['class'] .= ' js-datepicker';
        }
    }
}

```

```

    $this->form
    ->add('parution', DateType::class, array(
        'constraints' => array(new Assert\NotBlank()),
        'widget' => 'single_text',

        // do not render as type="date", to avoid HTML5 date pickers
        'html5' => false,

        // add a class that can be selected in JavaScript
        'attr' => $field_attrs,
    ))
    ;
} else {
    // Si Consultation ou Suppression, affichage
    // des champs verrouillés et sans contrôle
    $this->form
    ->add('crud', HiddenType::class, array(
    ))
    ->add('id', HiddenType::class, array(
    ))
    ->add('album', TextType::class, array(
        'attr' => $this->default_attrs
    ))
    ->add('auteur', TextType::class, array(
        'attr' => $this->default_attrs
    ))
    ->add('editeur', TextType::class, array(
        'attr' => $this->default_attrs
    ))
    ->add('parution', TextType::class, array(
        'attr' => $this->default_attrs
    ))
    ;
}
}
}

```

D'un point de vue fonctionnel, cette nouvelle version produit strictement le même résultat que la version précédente, mais on voit que le code source de la classe AlbumbdForm a été considérablement allégé. La classe CrudstdForm va pouvoir être réutilisée pour générer d'autres formulaires de type CRUD, au sein de notre application SILEX. Et nous pourrions même envisager de la réutiliser sur d'autres projets, mais ça c'est une autre histoire...

4.4. Traduction

Lors de vos tests du module CRUD, vous avez sans doute remarqué que les messages d'erreur étaient en anglais.

Auteur

- This value should not be blank.

Autre cas de figure :

Auteur

- This value is too short. It should have 2 characters or more.

Voici pour rappel le code définissant les contraintes liées au champ de saisie « auteur » :

```
->add('auteur', TextType::class, array(
    'constraints' => array(new Assert\NotBlank(),
        new Assert\Length(array('min' => 2)),
        new Assert\Length(array('max' => 30))
    ),
    'attr' => $this->default_attrs
))
```

Une première solution pour obtenir des messages en français consiste à les définir « en dur » sur chacun des contraintes, de la façon suivante :

```
->add('editeur', TextType::class, array(
    'constraints' => array(new Assert\NotBlank(
        array( 'message' => 'Zone obligatoire')
    ),
    new Assert\Length(array('min' => 2,
        'minMessage' => 'Saisir au moins 2 caractères')),
    new Assert\Length(array('max' => 30,
        'maxMessage' => 'Saisir au plus 30 caractères'))
),
    'attr' => $this->default_attrs
));
```

Cette solution fonctionne et elle est rapide à mettre en œuvre. Mais elle présente plusieurs inconvénients :

- on va être obligé de définir le même message (par exemple « Zone obligatoire ») à tous les niveaux où ce message est nécessaire (redondance de code en prévision... pas joli ☹)
- comment faire si on souhaite ajouter à notre application, de nouvelles langues, et les traductions correspondantes ?

SILEX propose une autre approche, au travers des services Locale et Translation, services qui sont fournis par Symfony.

Pour mettre en route ce mécanisme de traduction, commencez par modifier le script `src/app.php`, en ajoutant les lignes en gras suivantes :

```
// formulaire
use Silex\Provider\FormServiceProvider;
use Silex\Provider\LocaleServiceProvider;
use Silex\Provider\TranslationServiceProvider;

$app = new Application();

...

// formulaire et traduction
$app->register(new FormServiceProvider());
$app->register(new Silex\Provider\LocaleServiceProvider());
$app->register(new Silex\Provider\TranslationServiceProvider(), array(
    'locale_fallbacks' => array('en'),
));
```

Modifiez les contraintes sur le champ « editeur » de la façon suivante :

```
->add('editeur', TextType::class, array(
    'constraints' => array(
        new Assert\NotBlank(array('message'=>'not_blank')),
        new Assert\Length(array(
            'min' => 2,
            'minMessage'=> "min_length")),
        new Assert\Length(array(
            'max' => 30,
            'maxMessage'=> "max_length"))
    ),
    'attr' => $this->default_attrs
));
```

Dans `src/controllers.php`, juste en dessous des « use ... », ajoutez les lignes suivantes :

```
$app->register(new Silex\Provider\LocaleServiceProvider());
$app->register(new Silex\Provider\ValidatorServiceProvider());

$app['translator.domains'] = array(
    'messages' => array(
        'en' => array(
            'hello'      => 'Hello %name%',
            'goodbye'    => 'Goodbye %name%',
        ),
        'fr' => array(
            'hello'      => 'Bonjour %name%',
            'goodbye'    => 'Au revoir %name%',
        ),
    ),
    'validators' => array(
        'fr' => array(
            'not_numeric' => 'Cette valeur doit être un nombre.',
            'not_blank'  => 'Cette valeur ne peut être à blanc',
            'min_length' => 'Saisir au moins {{ limit }} caractères',
            'max_length' => 'Saisir au plus {{ limit }} caractères'
        ),
    ),
);

$app->before(
    function (Request $request) use ($app) {
        $app['translator']->setLocale(
            $request->getPreferredLanguage(['en', 'fr'])
        );
    }
);
```

Remarque : pour les messages d'erreur contenant une variable (telle que la longueur minimale ou maximale d'un champ de saisie), le paramètre « limit » va permettre d'insérer la valeur limite relative au message :

```
'min_length' => 'Saisir au moins {{ limit }} caractères',
```

Pour tester la traduction d'un message comme « hello », modifiez maintenant la route « testparam » :

```
$app->get('/testparam/{id}', function ($id) use ($app) {
    return $app['twig']->render('testparam.html.twig',
        array(
            'param1' => $id,
            'titi' => 'gros minet'
        ));
})
->bind('testparam')
;
```

Modifier également le template « testparam.html.twig » :

```
{% extends "layout.html.twig" %}

{% block content %}
<div class="jumbotron">
    Ceci est une page de test avec paramètre<br>
    Le paramètre reçu a la valeur suivante : {{param1}}<br>
    {{ 'hello'|trans({'%name%':titi}) }}
</div>
{% endblock %}
```

Ca fonctionne :

Ceci est une page de test avec paramètre
Le paramètre reçu a la valeur suivante : 200
Bonjour gros minet

Il nous reste à tester la bonne traduction des messages d'erreur :

Editeur

- Saisir au plus 30 caractères

[illegible]

Quelques liens utiles :

<http://symfony.com/doc/current/translation.html>

<http://stackoverflow.com/questions/25482856/basic-use-of-translationserviceprovider-in-silex>

<http://stackoverflow.com/questions/22320805/how-to-pass-parameter-to-translated-validation-error-message>

4.5. Les Modèles

Pour mettre en place une couche « modèle » au niveau de notre application, il nous faut faire plusieurs choses :

- installer le composant « doctrine/dbal » et le configurer au niveau de l'application
- créer une base MySQL (si pas déjà fait)
- créer une table MySQL contenant un jeu de données significatif
- établir une connexion entre MySQL et PHP (via Doctrine)
- créer un nouveau module de type CRUD (avec de nouvelles routes et templates), mais sur une vraie table SQL cette fois

Une table SQL avec 4 ou 5 albums de bande dessinée... ça va vite devenir ennuyeux. Nous allons créer une table contenant une liste de personnes, nous l'appellerons « persons » et nous allons y injecter une centaine de lignes. Je vous sens pâlir en vous demandant si vous allez devoir créer un jeu de données de 100 lignes. Que nenni ! On va générer ce jeu de données en utilisant un utilitaire gratuit qui s'appelle Mockaroo :

<https://www.mockaroo.com/>

Mockaroo propose de personnaliser la structure de la table à générer, d'appliquer éventuellement des formules pour le remplissage de certaines colonnes, et de préciser le format de sortie (SQL ou autre). Voici les paramètres que j'ai utilisés pour constituer mon jeu de données :

Field Name	Type	Options
id	Row Number	blank: 0 % fix x
first_name	First Name	blank: 0 % fix x
last_name	Last Name	blank: 0 % fix x
email	Email Address	blank: 0 % fix x
gender	Gender	blank: 0 % fix x

Add another field

Rows: 100 Format: SQL Table Name: MOCK_DATA ☐ include create table

Download Data Preview More Want to save this for later? [Sign up for free.](#)

Après avoir cliqué sur « Download Data », j'ai récupéré un fichier SQL contenant 100 lignes. Il me faut maintenant une table pour injecter ces données. En voici une, à la « sauce » MySQL :

```
CREATE TABLE persons (
  id int(6) NOT NULL PRIMARY KEY AUTO_INCREMENT,
  first_name varchar(30) NOT NULL DEFAULT '',
  last_name varchar(30) NOT NULL DEFAULT '',
  email varchar(60) NOT NULL DEFAULT '',
  gender char(1) NOT NULL DEFAULT ''
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

A noter : dans le jeu de données fourni par Mockaroo, la colonne « gender » contenait les valeurs « Female » et « Male ». J'ai fait un remplacement en masse au profit de « F » et « M ». J'ai également remplacé le nom de la table proposé par Mockaroo (MOCK_DATA) par le nom de ma propre table (persons). L'injection des 100 lignes dans la table « persons » est passée « comme une lettre à la poste ».

Maintenant que nous avons un jeu de données SQL, occupons-nous de la configuration de Doctrine. Nous devons d'abord l'ajouter à notre projet, par exemple en utilisant la ligne de commande de Composer :

```
composer require doctrine/dbal
```

Dans le fichier « src/app.php », nous allons ajouter le support de Doctrine via les lignes suivantes (avant le « return \$app ») :

```
use Silex\Provider\DoctrineServiceProvider;
$app->register(new DoctrineServiceProvider());
// Mot de passe MySQL par défaut :
// - sur WampServer et XAMPP : ""
// - sur MAMP : "root"
$app['db.options'] = array(
  'driver' => 'pdo_mysql',
  'host' => '127.0.0.1',
  'dbname' => 'firstsilex',
  'user' => 'root',
  'password' => '',
  'charset' => 'utf8'
);
```

Nous allons ajouter une nouvelle route et le template qui va avec :

- nouvelle route :

```
$app->get('/liste-persons/', function () use ($app) {
    $sql = 'SELECT id, first_name, last_name, email, gender FROM persons';
    $persons = $app['db']->fetchAll($sql);

    return $app['twig']->render('persons-list.html.twig', array(
        'persons' => $persons
    ));
})
->bind('liste-persons')
;
```

- nouveau template :

```
{% extends "layout.html.twig" %}

{% block content %}
<div class="listepersons_crud">
    <h2>Liste de personnes (CRUD)</h2>

    <a class="btn btn-success btn-sm" href="{{ path('person-crud', {'id':0,
'crud':'C'}) }}">Créer</a>

<table class="table table-striped">
    <thead>
        <tr>
            <th>Nom</th>
            <th>Prénom</th>
            <th>Email</th>
            <th>Genre</th>
            <th>&nbsp;</th>
        </tr>
    </thead>
    <tbody>
        {% for person in persons %}
            <tr data-id="{{ person.id }}">
                <td>{{ person.first_name|title }}</td>
                <td>{{ person.last_name|capitalize }}</td>
                <td>{{ person.email }}</td>
                <td>{{ person.gender }}</td>
                <td>
                    <a class="btn btn-default btn-sm" href="{{ path('person-crud',
{'id':person.id, 'crud':'R'}) }}">Afficher</a>
                    <a class="btn btn-success btn-sm" href="{{ path('person-crud',
{'id':person.id, 'crud':'U'}) }}">Modifier</a>
                    <a class="btn btn-warning btn-sm" href="{{ path('person-crud',
{'id':person.id, 'crud':'D'}) }}">Supprimer</a>
                </td>
            </tr>
        {% endfor %}
    </tbody>
</table>
</div>
{% endblock %}
```

```

        </td>
    </tr>
{% endfor %}
</tbody>
</table>
</div>
{% endblock %}

```

Vous pouvez mettre en commentaire les boutons qui font appel à la route « person-crud », étant donné qu'elle n'existe pas encore. Cela vous permettra de vérifier que la liste des personnes fonctionne correctement :

SILEX Project

Liste de personnes (CRUD)

Nom	Prénom	Email	Genre
Edward	Anderson	eanderson1@devhub.com	M
Philip	Alexander	palexander2@imdb.com	M
Juan	Crawford	jcrawford3@harvard.edu	M
Wayne	Cunningham	wcunningham4@liveinternet.ru	M
Gerald	Oliver	goliver5@ftc.gov	M

Nous devons aussi créer un nouveau formulaire dans « src/Form ». Il sera très proche du formulaire « AlumbdForm », hormis pour le champ « gender » qui est un champ de type « select » (type de champ que nous n'avons pas encore utilisé) :

```

<?php
namespace Form;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Validator\Constraints as Assert;
use Symfony\Component\Form\Extension\Core\Type\FormType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\HiddenType;
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\Form\Extension\Core\Type\DateType;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
use Symfony\Component\Form\Extension\Core\Type\ButtonType;
use Symfony\Component\Form\Extension\Core\Type\ResetType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

```



```

/**
 * Description of PersonForm
 *
 * @author Gregory Jarrige
 * @version 0.1 (2016-09-24)
 */
class PersonForm extends \Form\CrudstdForm {

    /**
     * Génération des champs du formulaire
     */
    protected function genFields() {
        if ($this->crud_context == 'U' || $this->crud_context == 'C') {
            $this->form
                ->add('crud', HiddenType::class, array(
                    'constraints' => array(new Assert\NotBlank())
                ))
                ->add('id', HiddenType::class, array(
                    'constraints' => array(new Assert\NotBlank())
                ))
                ->add('first_name', TextType::class, array(
                    'constraints' => array(new Assert\NotBlank(),
                        new Assert\Length(array('min' => 2)),
                        new Assert\Length(array('max' => 30))
                    ),
                    'attr' => $this->default_attrs
                ))
                ->add('last_name', TextType::class, array(
                    'constraints' => array(new Assert\NotBlank(),
                        new Assert\Length(array('min' => 2)),
                        new Assert\Length(array('max' => 30))
                    ),
                    'attr' => $this->default_attrs
                ))
                ->add('email', EmailType::class, array(
                    'constraints' => array(
                        new Assert\NotBlank(array('message'=>'not_blank')),
                        new Assert\Length(array(
                            'min' => 2,
                            'minMessage'=> "min_length")),
                        new Assert\Length(array(
                            'max' => 60,
                            'maxMessage'=> "max_length"))
                    ),
                    'attr' => $this->default_attrs
                ))
        }
    }
}

```

```

->add('gender', ChoiceType::class, array(
    'choices' => array(
        'Femme' => 'F',
        'Homme' => 'M',
        'Transgenre' => 'T',
    ),
    'attr' => $this->default_attrs ,
    'constraints' => array(
        new Assert\NotBlank(
            array(
                'message' => 'not_blank'
            )
        )
    )
)
)
;
} else {
    // Si Consultation ou Suppression, affichage
    // des champs verrouillés et sans contrôle
    $this->form
    ->add('crud', HiddenType::class, array(
    ))
    ->add('id', HiddenType::class, array(
    ))
    ->add('first_name', TextType::class, array(
        'attr' => $this->default_attrs
    ))
    ->add('last_name', TextType::class, array(
        'attr' => $this->default_attrs
    ))
    ->add('email', TextType::class, array(
        'attr' => $this->default_attrs
    ))
    ->add('gender', TextType::class, array(
        'attr' => $this->default_attrs
    ))
    ;
}
}
}

```

Nous devons maintenant créer la route « person-crud ». Pour ce faire, nous nous baserons sur la route « albumupdate », mais cette fois nous allons gérer une vraie base de données :

```
$app->match('/person-crud/{id}/{crud}', function (Request $request,
Silex\Application $app) {

    $personForm = new \Form\PersonForm($app, $request);
    $form = $personForm->getForm();

    if ($request->getMethod() == 'GET') {
        $id = (int) $request->get('id');
        $crud = strtoupper(strip_tags($request->get('crud')));
        $data = [];
        if ($crud == 'C') {
            $data = array(
                'crud' => $crud,
                'id' => 0,
                'first_name' => '',
                'last_name' => '',
                'email' => '',
                'gender' => ''
            );
        } else {
            $sql = 'SELECT id, first_name, last_name, email, gender
                FROM persons WHERE id = ?';
            $person = $app['db']->fetchAssoc($sql, array($id));
            if (!$person) { // (!array_key_exists($id, $albums)) {
                // redirection
                return $app->redirect($app['url_generator']
                    ->generate('personnotfound'));
            } else {
                foreach ($person as $key => $value) {
                    $data[$key] = $app->escape($value);
                };
                // On ajoute la notion de CRUD à notre jeu de données
                $data['crud'] = $crud;
            }
        }
    }
}
```

```

if ($request->getMethod() == 'POST') {
    $form->handleRequest($request);
    $data = $form->getData();
    if ($form->isSubmitted()) {
        // redirection si Return ou Si Formulaire valide
        if (isset($_POST['form']['return'])) {
            return $app->redirect($app['url_generator']
                                ->generate('liste-persons'));
        }
        if ($form->isValid()) {
            $sql_response = false;
            switch($data['crud']) {
                case 'C' : {
                    $sql_response = $app['db']->insert('persons', array(
                        'first_name'=>$data['first_name'],
                        'last_name'=>$data['last_name'],
                        'email'=>$data['email'],
                        'gender'=>$data['gender']
                    ));
                    break;
                }
                case 'U' : {
                    $sql_response = $app['db']->update('persons', array(
                        'first_name'=>$data['first_name'],
                        'last_name'=>$data['last_name'],
                        'email'=>$data['email'],
                        'gender'=>$data['gender']
                    ), array(
                        'id' => (int)$data['id'],
                    ));
                    break;
                }
                case 'D' : {
                    $sql_response = $app['db']->delete('persons', array(
                        'id'=>(int)$data['id']
                    ));
                    break;
                }
            }
            // redirection
            if ($data['crud'] == 'D') {
                return $app->redirect($app['url_generator']
                                    ->generate('persondeleted'));
            } else {
                return $app->redirect($app['url_generator']
                                    ->generate('personregistered'));
            }
        }
    }
}

```

```

return $app['twig']->render(
    'person-form.html.twig', array(
        'form' => $form->createView(),
        'data' => $data
    ));
})
->bind('person-crud');

```

N'oublions pas le template « person-form.html.twig », indispensable pour que notre nouvelle route fonctionne :

```

{% extends "layout.html.twig" %}
{% block content %}
    <form method="post" novalidate >
        <fieldset>
            {% set form_legend = '' %}
            {% if data.crud == 'C' %}
                {% set form_legend = 'Création' %}
            {% endif %}
            {% if data.crud == 'R' %}
                {% set form_legend = 'Consultation' %}
            {% endif %}
            {% if data.crud == 'U' %}
                {% set form_legend = 'Mise à jour' %}
            {% endif %}
            {% if data.crud == 'D' %}
                {% set form_legend = 'Confirmation de suppression' %}
            {% endif %}
            <legend>{{form_legend}}</legend>

            {{ form_widget(form.crud, {'value':data.crud}) }}
            {{ form_widget(form.id, {'value':data.id}) }}

            {{ form_errors(form) }}

            <div class="form-group">
                {{ form_label(form.first_name, "Nom", {'label_attr':
                    {'class': 'control-label'}}) }}
                {{ form_errors(form.first_name) }}
                {{ form_widget(form.first_name, {'value':data.first_name}) }}
            </div>
            <div class="form-group">
                {{ form_label(form.last_name, "Prénom", {'label_attr':
                    {'class': 'control-label'}}) }}
                {{ form_errors(form.last_name) }}
                {{ form_widget(form.last_name, {'value':data.last_name}) }}
            </div>
            <div class="form-group">
                {{ form_label(form.email, "Email", {'label_attr':
                    {'class': 'control-label'}}) }}

```

```

        {{ form_errors(form.email) }}
        {{ form_widget(form.email, {'value':data.email}) }}
    </div>
    <div class="form-group">
        {{ form_label(form.gender, "Genre", {'label_attr':
            {'class': 'control-label'}}) }}
        {{ form_errors(form.gender) }}
        {{ form_widget(form.gender, {'value':data.gender}) }}
    </div>
    <div class="form-inline">
        {% if form.return is defined %}
        {{ form_widget(form.return, {'label':'Retour'}) }}
        {% endif %}
        {% if form.save is defined %}
        {{ form_widget(form.save, {'label':'Valider'}) }}
        {% endif %}
        {% if form.reset is defined %}
        {{ form_widget(form.reset, {'label':'Effacer'}) }}
        {% endif %}
    </div>
</fieldset>
</form>
{% endblock %}

```

Et enfin 2 routes et 2 templates très simples :

```

$app->get('/personregistered/', function () use ($app) {

    return $app['twig']->render('personregistered.html.twig', array());
})
->bind('personregistered')
;

$app->get('/persondeleted/', function () use ($app) {

    return $app['twig']->render('persondeleted.html.twig', array());
})
->bind('persondeleted')
;

```

Je ne mets ci-dessous que le template « personregistered.html.twig », sachant que le template « persondeleted.html.twig » sera sur le même modèle :

```

{% extends "layout.html.twig" %}

{% block content %}
<div class="jumbotron">
    L'enregistrement de la personne en base de données est terminé.<br>
    Cliquez <a href="{{ path('liste-persons') }}">ici</a> pour réafficher la liste
des personnes.
</div>
{% endblock %}

```

Il ne nous reste plus qu'à tester tout ça. Si vous n'avez rien oublié, et si je n'ai moi-même rien oublié dans mes explications, vous devriez avoir un module CRUD opérationnel pour votre table des personnes.

FirstSilex Project	Accueil	Travaux ▾
--------------------	---------	-----------

SILEX Project

Liste de personnes (CRUD)

Créer

Nom	Prénom	Email	Genre			
Edward	Anderson	eanderson1@devhub.com	M	Afficher	Modifier	Supprimer
Philip	Alexander	palexander2@imdb.com	M	Afficher	Modifier	Supprimer
Juan	Crawford	jcrawford3@harvard.edu	M	Afficher	Modifier	Supprimer
Wayne	Cunningham	wcunningham4@liveinternet.ru	M	Afficher	Modifier	Supprimer
Gerald	Oliver	ooliver5@ftc.gov	M	Afficher	Modifier	Supprimer

Vous aurez sans doute constaté qu'il manque quelque chose, c'est un mécanisme de pagination. Sachant que notre table contient une centaine de lignes, nous les affichons en totalité dans la page, ce qui n'est franchement pas terrible.

Je vous propose de traiter cette notion de pagination dans la première partie du chapitre « Les Bonus ».

4.6. Sessions

Faute de temps, ce chapitre n'a pas été traité dans cette version du support. Il fera l'objet d'un développement ultérieur.

TODO : chapitre à rédiger

SILEX fournit un mécanisme de gestion de sessions, qui s'appuie sur la gestion des sessions de PHP.

Si vous ne maîtrisez pas parfaitement cette notion de session, je vous recommande de prendre le temps de lire au moins l'exemple simple proposé par la documentation officielle de PHP :

<http://php.net/manual/fr/session.examples.basic.php>

Vous trouverez ci-dessous plusieurs pages intéressantes traitant de ce sujet :

<http://silex.sensiolabs.org/doc/master/providers/session.html>

http://silex.sensiolabs.org/doc/master/cookbook/session_storage.html

<http://silex.sensiolabs.org/doc/providers/session.html>

<https://www.marcoalbarelli.eu/projects/a-silex-example/>

4.7. Tests unitaires

Faute de temps, ce chapitre n'a pas été traité dans cette version du support. Il fera l'objet d'un développement ultérieur.

TODO : chapitre à rédiger

On trouvera dans la documentation de SILEX une introduction sur la manière dont le framework intègre la gestion des tests unitaires.

<http://silex.sensiolabs.org/doc/master/testing.html>

5. Les bonus

Les techniques qui sont présentées dans ce chapitre ne sont pas propres à SILEX, pas plus qu'à Symfony. Ce sont des techniques que vous pouvez utiliser dans des applications développées en « pur PHP », ou développées au travers d'autres frameworks. Mais nous allons voir de quelle manière nous pouvons les mettre en œuvre dans SILEX.

5.1. Pagination et tri dynamique des colonnes

5.1.1 Pagination

Vous aurez remarqué que notre liste de personnes présente un gros défaut, elle n'intègre pas de mécanisme de pagination.

Nous allons voir comment intégrer un tel mécanisme, ce qui nous permettra d'afficher une liste avec barre de pagination comme dans l'exemple suivant :

FirstSilex Project	Accueil	Travaux ▾
--------------------	---------	-----------

Liste de personnes (CRUD)

[Créer](#)

Nom	Prénom	Email	Genre			
Joshua	Fowler	jfowlerl@state.tx.us	M	Afficher	Modifier	Supprimer
Jeffrey	Chapman	jchapmanm@merriam-webster.com	M	Afficher	Modifier	Supprimer
Justin	Turner	jturnern@reuters.com	M	Afficher	Modifier	Supprimer
Arthur	Perez	aperezo@google.fr	M	Afficher	Modifier	Supprimer
Janet	Bradley	jbradley@livejournal.com	F	Afficher	Modifier	Supprimer

[<< Préc.](#)
[1-5](#)
[6-10](#)
[11-15](#)
[16-20](#)
[...](#)
[96-99](#)
[Suiv. >>](#)

Je pensais m'appuyer, pour la rédaction de ce support, sur l'un des composants dédiés à la pagination proposé sur Packagist.org. Mais je dois dire que je n'ai pas été convaincu par les composants que j'y ai trouvés. Aussi, ai-je décidé d'utiliser dans ce cours une classe que j'avais créée précédemment, en m'inspirant assez

librement d'un jeu de fonctions proposé dans le livre de Adam Trachtenberg et David Sklar, « PHP Cookbook », aux éditions O'Reilly (2006). Comme le code source de cette classe est assez long, je l'ai placé en annexe du présent support. Je vous recommande de créer cette classe à l'intérieur du répertoire « src ».

En nous basant sur la route « liste-persons », qui existe déjà, je vous propose de créer une nouvelle route, « liste-personsv2 », dont voici le code source :

```
$app->get('/liste-personsv2/{offset}', function ($offset) use ($app) {
    $offset = (int)$offset;
    $nbl_par_page = 5 ;

    // comptage du nombre de lignes total
    $sql2 = 'SELECT count(*) as comptage FROM persons';
    $nblines = $app['db']->fetchAssoc($sql2);

    $offset_sql = $offset - 1;

    $sql1 = "SELECT id, first_name, last_name, email, gender
              FROM persons limit $offset_sql, $nbl_par_page";
    $persons = $app['db']->fetchAll($sql1);

    $barre_pagination = new \Pagination($nblines['comptage'],
        $offset, $nbl_par_page, '', array(), true);

    return $app['twig']->render('persons-listv2.html.twig', array(
        'persons' => $persons,
        'barre_pagination' => $barre_pagination->navBarBootstrap(),
        'offset' => $offset
    ));
})
->bind('liste-personsv2')
;
```

Vous constaterez que nous avons ajouté une notion d'offset, comme paramètre d'entrée de notre nouvelle route. Cette notion va nous permettre de paginer à l'intérieur du jeu de données renvoyé par SQL, en exploitant la clause « limit » de MySQL.

Nous avons aussi ajouté une requête SQL qui effectue un comptage du nombre de lignes total, et nous avons défini un nombre de lignes par page (ici fixé à 5 mais vous êtes libre d'augmenter cette valeur). Ces deux informations vont nous permettre de déterminer combien de liens générer dans notre barre de navigation. Vous noterez enfin que nous générons le code HTML de la barre de pagination avant de le passer au template.

Le code du template « persons-listv2.html.twig » est presque identique à celui du template « persons-list.html.twig ». Pour afficher notre barre de pagination sous le tableau HTML, nous avons simplement ajouté le code suivant, juste après la balise </table>.

```
{% autoescape false %}
{{barre_pagination}}
{% endautoescape %}
```

5.1.2 Tri dynamique des colonnes

Certains utilisateurs sont habitués à des fonctionnalités telles que le tri dynamique des colonnes, et ils ne vont pas tarder à vous le réclamer. Autant prendre les devants, en ajoutant tout de suite cette fonctionnalité. Ce type de fonctionnalité pourrait alourdir considérablement notre code PHP, heureusement nous allons nous appuyer sur un composant Javascript qui gère ça très bien et nécessite très peu de modifications, j'ai nommé « Tablesort ».

Ce composant nous est généreusement offert sous licence MIT, il est téléchargeable ici :

<http://tristen.ca/tablesort/demo/>

Une fois téléchargé le fichier Javascript, installez-le dans le répertoire « src/js ». Modifiez ensuite le fichier « web/css/main.css » en y ajoutant le code CSS suivant :

```
/* CSS pour l'affichage des datagrids avec tablesort */
/* http://localhost/tablesort-gh-pages/demo/index.html */

th.sort-header::-moz-selection { background:transparent; }
th.sort-header::selection { background:transparent; }
th.sort-header { cursor:pointer; }
table th.sort-header:after {
  content:'';
  float:right;
  margin-top:7px;
  border-width:0 4px 4px;
  border-style:solid;
  border-color:#404040 transparent;
  visibility:hidden;
}
table th.sort-header:hover:after {
  visibility:visible;
```

```

    }
table th.sort-up:after,
table th.sort-down:after,
table th.sort-down:hover:after {
    visibility:visible;
    opacity:0.4;
}
table th.sort-up:after {
    border-bottom:none;
    border-width:4px 4px 0;
}

table>tbody>tr.sort:nth-child(odd) {
    background-color: #e4e4e4;
}

table>tbody>tr.sort:hover {
    color: #000000;
    text-decoration: none;
    cursor: pointer;
    opacity: 0.4;
    filter: alpha(opacity=60);
}

.container_900 {
    width: 900px;
}
.container_800 {
    width: 800px;
}
.container_700 {
    width: 700px;
}
.container_600 {
    width: 600px;
}

```

Modifiez enfin le template « persons-listev2.html.twig » de la façon suivante :

```

{% extends "layout.html.twig" %}

{% block content %}

<script src="{ { asset('js/tablesort_gh-pages.min.js') }}"></script>

<div class="listepersons_crud">
    <h2>Liste de personnes (CRUD)</h2>

    <a class="btn btn-success btn-sm" href="{ { path('person-crud', {'id':0,
'crud':'C'}) }}">Créer</a>

<table class="table table-striped" id="datagrid">

```

```

<thead>
  <tr>
    <th>Nom</th>
    <th>Prénom</th>
    <th>Email</th>
    <th>Genre</th>
    <th>&nbsp;</th>
  </tr>
</thead>
<tbody>
{% for person in persons %}
  <tr data-id="{{person.id}}">
    <td>{{person.first_name|title}}</td>
    <td>{{person.last_name|capitalize}}</td>
    <td>{{person.email}}</td>
    <td>{{person.gender}}</td>
    <td>
      <a class="btn btn-default btn-sm" href="{{ path('person-crud',
{'id':person.id, 'crud':'R'}) }}">Afficher</a>
      <a class="btn btn-success btn-sm" href="{{ path('person-crud',
{'id':person.id, 'crud':'U'}) }}">Modifier</a>
      <a class="btn btn-warning btn-sm" href="{{ path('person-crud',
{'id':person.id, 'crud':'D'}) }}">Supprimer</a>
    </td>
  </tr>
{% endfor %}
</tbody>
</table>
{% autoescape false %}
{{barre_pagination}}
{% endautoescape %}
</div>

<script>
  $(document).ready(function() {
    var datagrid = new Tablesort(document.getElementById('datagrid'));
  });
</script>
{% endblock %}

```

Rafraichissez enfin la page, vous devriez avoir maintenant la possibilité de trier dynamiquement les colonnes, de manière ascendante ou descendante, simplement en cliquant sur leurs entêtes :

Nom	▲ Prénom ▼	Email
James	Ramirez	jramirezd@canalblog.com
Matthew	Morris	mmorrisb@photobucket.com

Peut être avez-vous entendu parler de composants Javascript tels que Datatables ou JQGrid, qui gèrent tous deux la pagination et le tri dynamique des colonnes. Si c'est le cas, vous vous demandez probablement pourquoi je n'ai pas utilisé l'un de ces composants au lieu de vouloir gérer la pagination en PHP.

Les composants Datatables et JQGrid sont très puissants, et surtout très pratiques à utiliser. A tel point que beaucoup de développeurs ont tendance à en abuser, et n'hésitent pas à charger dans les pages HTML des tableaux de plusieurs centaines de lignes, laissant à Datatables (ou JQGrid) le soin de gérer la pagination à leur place. Ainsi, des milliers de lignes de tableaux HTML, chargées inutilement, engorgent chaque jour les réseaux des entreprises, dont la bande passante n'est pas extensible (et c'est encore plus sensible en Wifi). Ce trafic réseau inutile ralentit le chargement des pages et pénalise les utilisateurs dans leurs tâches quotidiennes. Plus grave encore, ce travail inutile produit par des serveurs entraîne un surcroît de travail pour les processeurs, qui chauffent davantage et consomment davantage d'électricité... pas très bon pour la facture d'électricité de votre entreprise, et surtout pas très bon pour la planète ☹.

Vous pourriez penser que j'exagère, aussi je vous recommande la lecture d'un article de Scott Hayes, publié en 2009, article qui s'accompagne d'une vidéo dans laquelle Scott nous montre l'accroissement de consommation électrique induite par des traitements SQL mal optimisés :

http://www.dbsoftware.com/blog/db2_performance.php?id=127

Plutôt que de lire trop de données SQL, et de laisser au navigateur le soin de gérer la pagination sur ces données, je préfère confier au serveur la responsabilité de la pagination, de manière à ne faire transiter sur les réseaux que les données qui doivent être réellement affichées. En revanche, j'apprécie de ne pas avoir à gérer le tri dynamique des colonnes. C'est une fonctionnalité

dont les navigateurs peuvent s'acquitter sans intervention du serveur, et ils le font très bien grâce à un composant comme Tablesort.

Il est d'ailleurs intéressant de noter que Tablesort est un composant très léger et très performant (il est écrit en pur Javascript, et n'a donc pas besoin de jQuery pour fonctionner).

5.1.3 Pagination combinée avec des Filtres de sélection

TODO : chapitre à rédiger

5.2. Graphiques avec Highcharts

Beaucoup d'applications affichent des tableaux de bord graphiques, à base de camemberts, de donuts et d'histogrammes en tous genres. Ces tableaux de bord sont produits à partir de données statistiques obtenues le plus souvent via des requêtes SQL, quand ce n'est pas à partir de fichiers aux formats XML, CSV ou JSON.

A la fin des années 90, et même au début des années 2000, on utilisait des bibliothèques de code PHP pour produire des graphiques d'entreprise. Des projets tels que JGraph étaient couramment utilisés pour cela. Les images obtenues étaient des images de type « bitmap », le rendu était généralement correct, mais le niveau d'interactivité avec ce type de graphique était nul.

Vers le milieu des années 2000, on a commencé à voir apparaître des projets écrits en Javascript, qui produisaient des graphiques impressionnants, offrant un niveau d'interactivité impressionnant. Un des premiers projets Javascript à avoir proposé des graphismes réellement impressionnants, c'était le framework Dojo. A partir de là, il devenait évident que les générateurs de graphiques écrits en Javascript allaient enterrer les solutions plus anciennes écrites en PHP. Et ça n'a pas traîné.

Qu'ils s'appellent D3.js, Amcharts ou Highcharts, les générateurs de graphiques destinés à s'afficher dans des navigateurs sont tous écrits en Javascript. Ils s'appuient soit sur l'API Canvas, soit sur la norme SVG. Ces deux techniques (Canvas et SVG) sont des API de la norme HTML5, elles utilisent des fonctions de tracé vectoriel, à la différence que Canvas produit en sortie des images de type « bitmap » alors que SVG est « vectoriel » de bout en bout. Les deux techniques sont très bien supportées par les navigateurs récents, qu'ils « tournent » sur desktop ou sur terminal mobile (tablette ou smartphone).

Il me fallait en choisir un, alors j'ai jeté mon dévolu sur Highcharts. Ce n'est pas tout à fait un hasard, si j'ai choisi ce générateur : il est très puissant, facile à mettre en œuvre, propose plein d'exemples très clairs et aussi très beaux, et il est gratuit tant que vous l'utilisez pour un usage personnel, comme nous sommes en train de le faire dans le cadre de cette formation.

Le site officiel de Highcharts est le suivant :

<http://www.highcharts.com/>

Vous pourrez télécharger Highcharts à partir du site officiel, mais pour l'heure, si vous disposez d'une connexion internet, vous pouvez le tester au sein de votre projet SILEX sans installer quoi que ce soit. Nous allons voir comment dans un instant.

Pour notre exemple de tableau de bord, nous allons nous baser sur un exemple très simple proposé sur le site officiel du projet Highcharts. Nous allons procéder en deux temps :

- 1- tout d'abord nous allons créer un contrôleur et un template exploitant l'intégralité d'un exemple du site officiel
- 2- ensuite nous allons modifier l'exemple pour injecter des données qui proviendront d'une base de données fictive, afin de nous rapprocher au maximum d'un cas réel tel que vous pourriez en rencontrer en entreprise.

L'exemple proposé par Highcharts que nous allons utiliser est le suivant :

<http://www.highcharts.com/demo/line-basic>

Dans votre projet SILEX, commencez par créer un nouveau contrôleur :

```
$app->get('/dashboard-01/', function () use ($app) {
    return $app['twig']->render('dashboard.html.twig', array(
        'data' => array(),
    ));
})
->bind('dashboard-01')
;
```

Créez ensuite un nouveau template « dashboard.html.twig » :

```
{% extends "layout.html.twig" %}

{% block content %}

<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/exporting.js"></script>

<div id="dashboard-container" style="min-width: 310px; height: 400px; margin: 0
auto"></div>

<script type="text/javascript">
$(function () {
    $('#dashboard-container').highcharts({
        title: {
            text: 'Monthly Average Temperature',
            x: -20 //center
        },
        subtitle: {
            text: 'Source: WorldClimate.com',
            x: -20
        },
        xAxis: {
            categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
        },
        yAxis: {
            title: {
                text: 'Temperature (°C)'
            },
            plotLines: [{
                value: 0,
                width: 1,
                color: '#808080'
            }]
        },
        tooltip: {
            valueSuffix: '°C'
        },
        legend: {
            layout: 'vertical',
            align: 'right',
            verticalAlign: 'middle',
            borderWidth: 0
        },
        series: [{
            name: 'Tokyo',
            data: [7.0, 6.9, 9.5, 14.5, 18.2, 21.5, 25.2, 26.5, 23.3, 18.3, 13.9, 9.6]
        }, {
            name: 'New York',
            data: [-0.2, 0.8, 5.7, 11.3, 17.0, 22.0, 24.8, 24.1, 20.1, 14.1, 8.6, 2.5]
        }, {
```

```

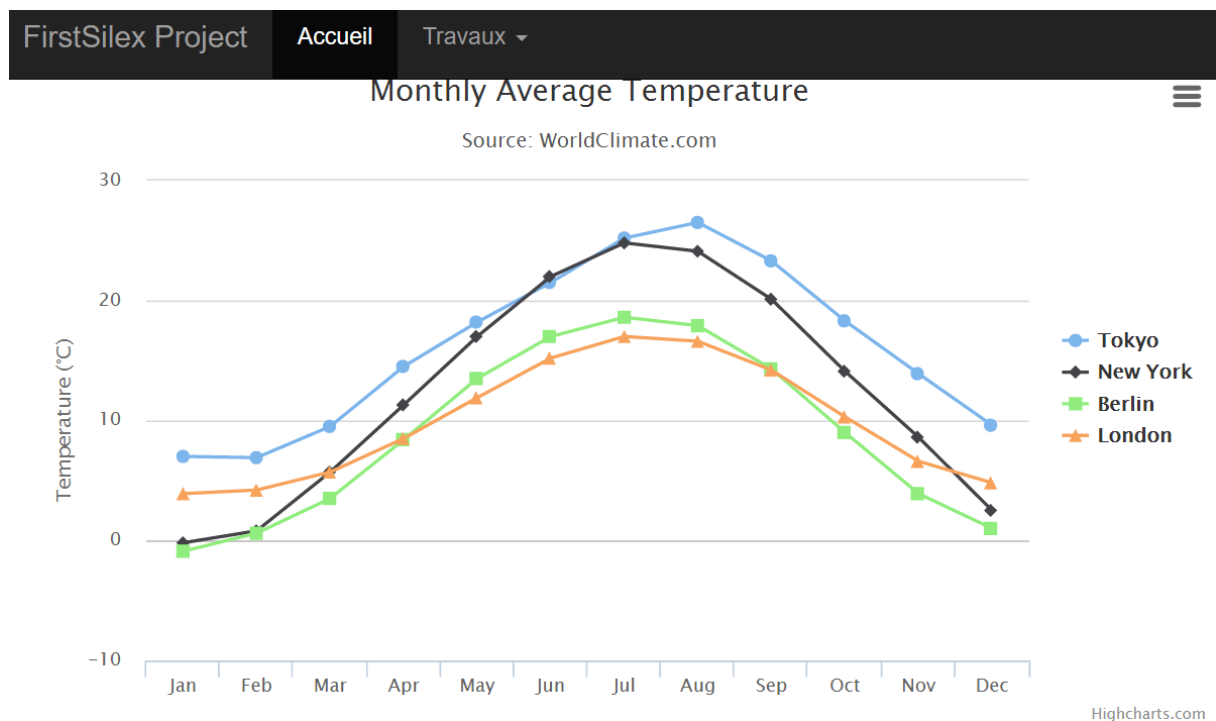
        name: 'Berlin',
        data: [-0.9, 0.6, 3.5, 8.4, 13.5, 17.0, 18.6, 17.9, 14.3, 9.0, 3.9, 1.0]
    }, {
        name: 'London',
        data: [3.9, 4.2, 5.7, 8.5, 11.9, 15.2, 17.0, 16.6, 14.2, 10.3, 6.6, 4.8]
    }
  });
});
</script>

{% endblock %}

```

Testez la nouvelle route dans votre navigateur :

<http://localhost/firstsilex/web/dashboard-01/>



C'est la classe !!! 😊

Bon, mais dans la « vraie vie », les données du graphique proviendraient très certainement de données extraites en live d'une base de données. Comment dans ce cas les injecter dans le graphique, en remplacement des données de l'exemple ? Vous allez voir que c'est très simple.

Si l'on jette un coup d'œil au code exemple de Highcharts, on voit que les données à l'origine des jolies courbes du graphique se situent dans le paramètre « series » :

```

series: [{
  name: 'Tokyo',
  data: [7.0, 6.9, 9.5, 14.5, 18.2, 21.5, 25.2, 26.5, 23.3, 18.3, 13.9, 9.6]
}, {
  name: 'New York',
  data: [-0.2, 0.8, 5.7, 11.3, 17.0, 22.0, 24.8, 24.1, 20.1, 14.1, 8.6, 2.5]
}, {
  name: 'Berlin',
  data: [-0.9, 0.6, 3.5, 8.4, 13.5, 17.0, 18.6, 17.9, 14.3, 9.0, 3.9, 1.0]
}, {
  name: 'London',
  data: [3.9, 4.2, 5.7, 8.5, 11.9, 15.2, 17.0, 16.6, 14.2, 10.3, 6.6, 4.8]
}]

```

Le paramètre « series » est en fait un tableau d'objets, au sens Javascript du terme. Il ressemble comme deux gouttes d'eau à un tableau associatif au sens PHP du terme. Voyons à quoi ce jeu de données pourrait ressembler dans sa version PHP :

```

$series =
[[
  'name'=> 'Tokyo',
  'data'=> [7.0, 6.9, 9.5, 14.5, 18.2, 21.5, 25.2, 26.5, 23.3, 18.3, 13.9, 9.6]
], [
  'name'=> 'New York',
  'data'=> [-0.2, 0.8, 5.7, 11.3, 17.0, 22.0, 24.8, 24.1, 20.1, 14.1, 8.6, 2.5]
], [
  'name'=> 'Berlin',
  'data'=> [-0.9, 0.6, 3.5, 8.4, 13.5, 17.0, 18.6, 17.9, 14.3, 9.0, 3.9, 1.0]
], [
  'name'=> 'London',
  'data'=> [3.9, 4.2, 5.7, 8.5, 11.9, 15.2, 17.0, 16.6, 14.2, 10.3, 6.6, 4.8]
]];

```

Très bien, injectons ce jeu de données (plus un titre en français et une liste de mois en français) dans notre route, pour pouvoir les transmettre au template :

```
$app->get('/dashboard-01/', function () use ($app) {

    $series =
        [[
            'name'=> 'Tokyo',
            'data'=> [7.0, 6.9, 9.5, 14.5, 18.2, 21.5, 25.2, 26.5, 23.3, 18.3, 13.9, 9.6]
        ], [
            'name'=> 'New York',
            'data'=> [-0.2, 0.8, 5.7, 11.3, 17.0, 22.0, 24.8, 24.1, 20.1, 14.1, 8.6, 2.5]
        ], [
            'name'=> 'Berlin',
            'data'=> [-0.9, 0.6, 3.5, 8.4, 13.5, 17.0, 18.6, 17.9, 14.3, 9.0, 3.9, 1.0]
        ], [
            'name'=> 'London',
            'data'=> [3.9, 4.2, 5.7, 8.5, 11.9, 15.2, 17.0, 16.6, 14.2, 10.3, 6.6, 4.8]
        ]
    ];

    $mois = ['Jan', 'Fév', 'Mar', 'Avr', 'Mai', 'Jui',
            'Jul', 'Aou', 'Sep', 'Oct', 'Nov', 'Déc'];

    return $app['twig']->render('dashboard.html.twig', array(
        'datas' => json_encode($series),
        'titre' => 'Température moyenne mensuelle',
        'mois'  => json_encode($mois),
    ));
})
->bind('dashboard-01')
;
```

Vous constatez qu'on utilise la fonction PHP `json_encode()` pour transformer notre tableau associatif en données au format JSON, avant de l'envoyer au template. On fait d'ailleurs la même chose pour le tableau des mois.

Nous avons quelques modifications à faire dans le template, elles se situent à l'intérieur de la balise « script » définissant le graphique à afficher (cf. les parties en rouge) :

```
<script type="text/javascript">
$(function () {
    // désactivation de l'échappement automatique de Twig
    {% autoescape false %}
    var datas = {{datas}};
    var mois = {{mois}};
    {% endautoescape %}
    $('#dashboard-container').highcharts({
        title: {
            text: '{{titre}}',
            x: -20 //center
        },
        subtitle: {
            text: 'Source: WorldClimate.com',
            x: -20
        },
        xAxis: {
            categories: mois
        },
        yAxis: {
            title: {
                text: 'Temperature (°C)'
            },
            plotLines: [{
                value: 0,
                width: 1,
                color: '#808080'
            }]
        },
        tooltip: {
            valueSuffix: '°C'
        },
        legend: {
            layout: 'vertical',
            align: 'right',
            verticalAlign: 'middle',
            borderWidth: 0
        },
        // injection ici des données provenant de la route
        series: datas
    });
});
</script>
```

Attention, le code ci-dessous est très important, sans lui les jeux de données ne seraient pas du vrai JSON, l'interpréteur Javascript ne saurait pas les exploiter et le graphique ne s'afficherait pas :

```
// désactivation de l'échappement automatique de Twig
{% autoescape false %}
    var datas = {{datas}};
    var mois = {{mois}};
{% endautoescape %}
```

Voilà, vous savez maintenant injecter des données provenant de PHP dans un graphique généré en Javascript par Highcharts. Vous n'avez plus qu'à remplacer les jeux de données fictifs par de vrais jeux de données.

Pensez à ajouter une nouvelle option de menu dans le menu « travaux » du template « layout.html.twig » :

```
<li><a href="{{ path('dashboard-01') }}">Dashboard Highcharts</a></li>
```

Au fait, pensez à « commiter » les dernières modifications.

Avant d'en finir avec ce chapitre, on rappelle que dans notre exemple, les fichiers Javascript de Highcharts sont chargés directement à partir du site officiel (cf. extrait ci-dessous) :

```
<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/exporting.js"></script>
```

En cas de coupure réseau, ces fichiers ne sont plus accessibles et votre application n'affichera plus aucun graphique. Vous avez donc tout intérêt à télécharger et à installer ces fichiers dans le répertoire « web/js » de manière à pouvoir en disposer en toutes circonstances.

```
{#<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/exporting.js"></script>#}
<script src="{{ asset('js/highcharts.js') }}"></script>
<script src="{{ asset('js/highcharts-exporting.js') }}"></script>
```


Il faut aussi souligner que Highcharts a besoin de jQuery pour fonctionner. Cela tombe bien, car nous avons installé jQuery précédemment, pour faire fonctionner les menus de Bootstrap, ainsi que pour la saisie de date dans les formulaires (le fameux « datepicker »).

5.3. Export PDF

Pour générer du PDF, nous allons court-circuiter Twig et recourir à un autre composant.

Il existe plusieurs composants PHP spécialisés dans la production de documents PDF, tels que :

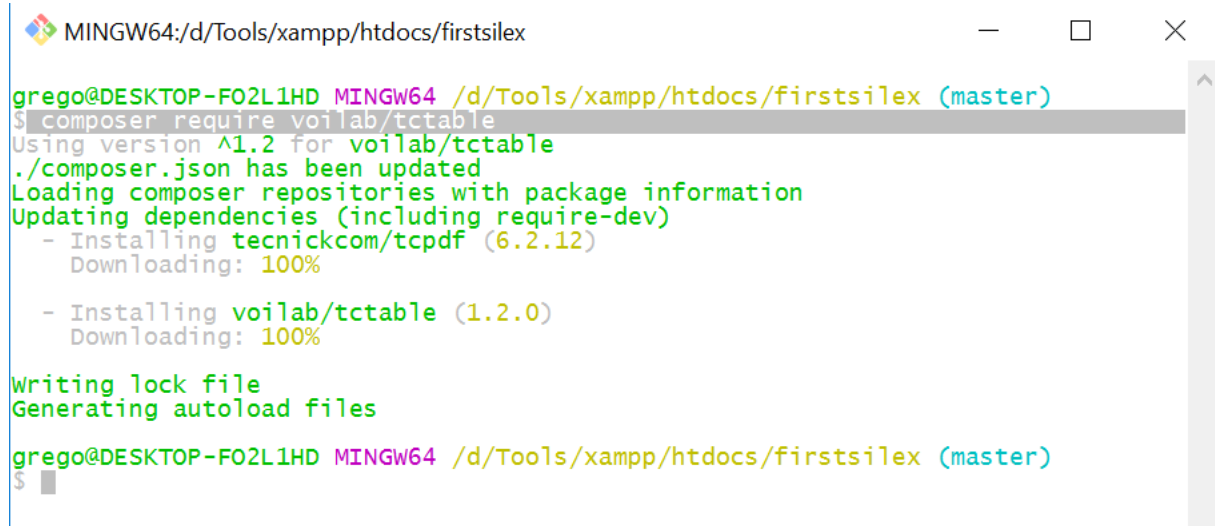
- FPDF : <http://www.fpdf.org>
- TCPDF : <https://tcpdf.org>
- MPDF : <http://www.mpdf1.com> (projet arrêté courant 2016)
- PHPPdf : <https://github.com/psliwa/PHPPdf>
- Zend_PDF : <https://framework.zend.com/manual/1.10/en/zend.pdf.html>

NB : Zend_PDF est un excellent composant qui est apparu sur Zend Framework 1, et qui n'a jamais été reconduit sur Zend Framework 2. C'est regrettable car ce composant offre des fonctionnalités que les autres n'ont pas, comme la possibilité d'analyser le contenu de fichiers PDF et d'en extraire les métadonnées. Si le projet ZF1 est aujourd'hui arrêté, le composant Zend_PDF est toujours utilisable à l'heure actuelle, et je vous encourage à l'essayer si vous avez des besoins spécifiques non couverts par les autres composants.

Pour la suite de ce chapitre, je vous propose de travailler avec FPDF ou TCPDF, ces deux composants étant probablement les plus utilisés actuellement. Il faut juste trouver la manière la plus pratique d'intégrer l'un ou l'autre de ces composants dans un projet Twig. Après quelques recherches sur Packagist.org, j'ai trouvé le composant « voilab/tctable », spécifiquement conçu pour utiliser TCPDF au sein de projets Symfony (donc forcément compatible avec Twig). Il est de surcroît très pratique pour générer des listes sous forme de tableaux, et ça tombe bien, c'est le type de document que je souhaite produire 😊.

Commencez par installer, avec l'aide de Composer, le composant voilab/tctable (ce qui aura pour effet d'embarquer aussi TCPDF) :

```
composer require voilab/tctable
```



```
MINGW64:/d/Tools/xampp/htdocs/firstsilex
grego@DESKTOP-FO2L1HD MINGW64 /d/Tools/xampp/htdocs/firstsilex (master)
$ composer require voilab/tctable
Using version ^1.2 for voilab/tctable
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
- Installing tecnickcom/tcpdf (6.2.12)
  Downloading: 100%
- Installing voilab/tctable (1.2.0)
  Downloading: 100%
Writing lock file
Generating autoload files
grego@DESKTOP-FO2L1HD MINGW64 /d/Tools/xampp/htdocs/firstsilex (master)
$
```

Pour créer notre exemple de document PDF, je me suis basé sur un exemple emprunté à la doc officielle du composant TCTABLE :

<https://github.com/voilab/tctable>

Nous allons donc créer une nouvelle route dans script src/controllers.php. Je vais vous fournir ci-dessous l'intégralité du code relatif à cette nouvelle route, vous allez ainsi pouvoir l'ajouter dans votre projet et le tester, et nous allons ensuite le détailler ensemble.

Voici le code source complet de cette nouvelle route :

```
$app->get('/listebd-pdf/', function () use ($app) {
    require_once 'tempdata/liste_bd_temp.php';
    $tmprows = getListeBD();
    // le composant tctable attend un tableau d'objets en entrée
    // alors on reformate notre liste pour qu'elle corresponde
    // au format attendu (et on la stocke dans $rows)
    $rows = array();
    foreach($tmprows as $tmprow) {
        $tmpobj = new stdClass();
        $tmpobj->album = $tmprow['album'];
        $tmpobj->auteur = $tmprow['auteur'];
        $tmpobj->editeur = $tmprow['editeur'];
        $tmpobj->parution = $tmprow['parution'];
        $rows[] = $tmpobj;
    }

    $pdf = new \TCPDF();
    $minRowHeight = 6; //mm

    $tctable = new \voilab\tctable\TcTable($pdf, $minRowHeight);
    // on prépare les entête de colonne de notre tableau PDF
    $tctable->setColumns([
        'titre' => [
            'isMultiLine' => true,
            'header' => 'Titre',
            'width' => 60
            // check inline documentation to see what options are available.
            // Basically, it's everything TCPDF Cell and MultiCell can eat.
        ],
        'auteur' => [
            'isMultiLine' => true,
            'header' => 'Auteur',
            'width' => 40,
        ],
        'editeur' => [
            'isMultiLine' => true,
            'header' => 'Editeur',
            'width' => 40,
        ],
        'parution' => [
            'header' => 'Parution',
            'width' => 10,
            // 'align' => 'R'
        ]
    ]);
});
```

```

// add a page so the content can be printed on something
$pdf->AddPage();
// draw body
$tctable->addBody($rows, function (\voilab\tctable\TcTable $table, $row) {
    $change_rate = 0.8;
    // map row data to TcTable column definitions
    return [
        'titre' => $row->album,
        'auteur' => $row->auteur,
        'editeur' => $row->editeur,
        'parution' => $row->parution,
    ];
});

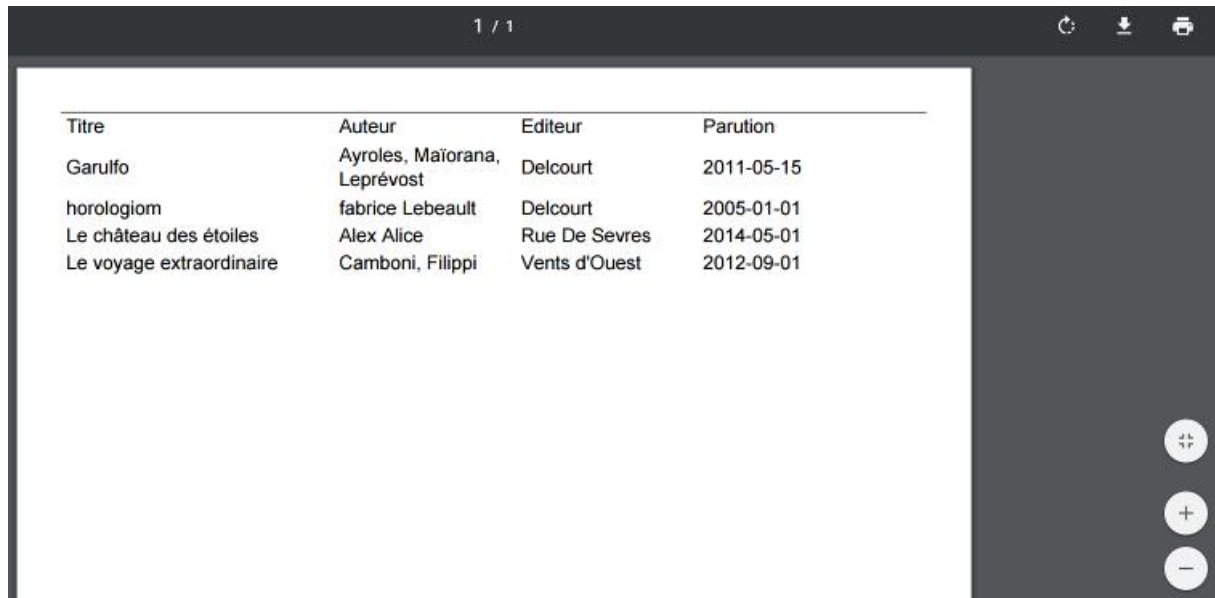
// Définition des entêtes HTTP qui vont permettre au navigateur
// de comprendre que le document reçu est un PDF
$nom_fichier = 'listeBD';
$params = array();
if (strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE')) {
    // Paramètres spécifiques à Internet Explorer
    $params['Pragma'] = 'public';
    $params['Expires'] = 0;
    $params['Cache-Control'] = 'must-revalidate, post-check=0, pre-check=0';
    $params['Content-type'] = 'application-download';
    // $params['Content-Length'] = $size;
    $params['Content-Disposition'] = "attachment; filename={$nom_fichier}.pdf";
    $params['Content-Transfer-Encoding'] = 'binary';
} else {
    // Paramètres qui fonctionnent bien sur Firefox et Chrome
    // $params['Content-type'] = 'application/pdf';
    $params['Content-type'] = 'application-download';
    // $params['Content-Length'] = $size;
    $params['Content-Disposition'] = "attachment; filename={$nom_fichier}.pdf";
    $params['Pragma'] = 'no-cache';
    $params['Cache-Control'] = 'no-store, no-cache, must-revalidate';
    $params['Expires'] = 'Sat, 26 Jul 1997 05:00:00 GMT';
}

$response = new Response();
foreach($params as $paramkey=>$paramval) {
    $response->headers->set($paramkey, $paramval);
}
$response->setStatusCode(200, 'OK');
$response->sendHeaders();
$response->sendContent($pdf->Output('tctable.pdf', 'I'));
return $response;
})
->bind('listeBD-pdf')
;

```

Pour tester cette nouvelle route, saisissez dans votre navigateur l'URL suivante :
<http://localhost/firstsilex/web/listebd-pdf/>

Si vous n'avez pas fait d'erreur, voici ce que vous devriez voir apparaître dans Chrome, par exemple :



The screenshot shows a web browser window with a table of book data. The table has four columns: Titre, Auteur, Editeur, and Parution. The data is as follows:

Titre	Auteur	Editeur	Parution
Garulfo	Ayroles, Maïorana, Leprévost	Delcourt	2011-05-15
horologiom	fabrice Lebeault	Delcourt	2005-01-01
Le château des étoiles	Alex Alice	Rue De Sevres	2014-05-01
Le voyage extraordinaire	Camboni, Filippi	Vents d'Ouest	2012-09-01

Vous aurez sans doute noté la présence d'un bloc de code destiné à modifier certains paramètres de la requête HTTP qui va être envoyée au navigateur. En PHP « pur » nous aurions utilisé la fonction `header()`, mais avec SILEX nous utiliserons la portion de code suivante :

```
$response = new Response();
foreach($params as $paramkey=>$paramval) {
    $response->headers->set($paramkey, $paramval);
}
$response->setStatusCode(200, 'OK');
$response->sendHeaders();
```

Cette modification des entêtes HTTP est indispensable si l'on souhaite envoyer au navigateur autre chose que du HTML. On se servira aussi de cette technique dans les chapitres suivants (pour l'export au format XML, JSON et CSV), mais avec des paramètres différents. Nous reviendrons un peu plus loin sur le contenu du tableau `$params`.

Dans le code de notre nouvelle route dédiée à la génération de PDF, vous aurez probablement noté la présence du test suivant :

```
if (strstr($_SERVER['HTTP_USER_AGENT'], 'MSIE')) {
    // entêtes HTTP pour IE
    ...
} else {
    // entêtes HTTP pour Firefox, Chrome, IE11 et Edge
    ...
}
```

Le tableau `$_SERVER` est un tableau réservé fourni par PHP, il n'est pas spécifique à SILEX, pas plus qu'à Symfony, vous pouvez donc l'exploiter sur des projets n'utilisant pas ces frameworks.

Concernant `$_SERVER`, la doc officielle sur « php.net » indique ceci :

`$_SERVER` est un tableau contenant des informations comme les en-têtes, dossiers et chemins du script. Les entrées de ce tableau sont créées par le serveur web. Il n'y a aucune garantie que tous les serveurs les rempliront tous ; certains en oublieront quelques-unes et en rajouteront de nouvelles non mentionnées ici. Cependant, un grand nombre de ces variables fait partie des » [spécifications CGI/1.1](#), et vous pouvez donc vous attendre à les retrouver.

Le paramètre qui nous intéresse ici, c'est le « `HTTP_USER_AGENT` », voici ce qu'en dit la doc officielle :

`'HTTP_USER_AGENT'`

Contenu de l'en-tête *User-Agent*: de la requête courante, si elle existe. C'est une chaîne qui décrit le client HTML utilisé pour voir la page courante. Par exemple : Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586). Entre autres choses, vous pouvez utiliser cette valeur avec [get_browser\(\)](#) pour optimiser votre page en fonction des capacités du client.

Si vous souhaitez approfondir ce sujet voici le lien :

<http://php.net/manual/fr/reserved.variables.server.php>

Donc en l'occurrence, je me sers de ce paramètre « `HTTP_USER_AGENT` » pour essayer de déterminer si mon navigateur fait partie de la famille des Microsoft Internet Explorer. Ce test fonctionnait bien il y a une dizaine d'années, que je l'utilisais sur certains projets, mais est-il encore pertinent en 2016 ? Pour le savoir, je vous propose d'ajouter dans le code notre nouvelle route, la ligne suivante :

```
error_log($_SERVER['HTTP_USER_AGENT']);
```

Si après chaque requête HTTP, vous consultez la log des erreurs PHP, vous constaterez comme moi que le test sur HTTP_USER_AGENT n'est plus franchement pertinent, sauf peut être pour les vieilles versions d'Internet Explorer. Voici en effet les valeurs que je récupère dans la log PHP, pour quelques navigateurs :

- Firefox :

```
Mozilla/5.0 (Windows NT 10.0; WOW64; rv:47.0) Gecko/20100101 Firefox/47.0
```

- Chrome :

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.116 Safari/537.36
```

- Microsoft Edge :

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/46.0.2486.0 Safari/537.36 Edge/13.10586
```

- Internet Explorer 11 (IE11) :

```
Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; Touch; rv:11.0) like Gecko
```

Si vous utilisez Safari ou Opera, je vous invite à faire le même test que moi, par curiosité.

En tout cas, on voit que même pour IE11, le test sur la chaîne « MSIE » n'est plus pertinent, et on voit même qu'il est impossible de déterminer avec certitude quel est le navigateur utilisé par l'internaute.

Selon le type de projet sur lequel vous travaillez, vous pouvez considérer que le test sur « MSIE » n'est pas pertinent pour vous, et vous êtes en droit de ne conserver que la partie de code relative aux navigateurs récents, soit :

```
// Paramètres qui fonctionnent bien sur Firefox et Chrome
$params['Content-type'] = 'application-download';
$params['Content-Disposition'] = "attachment; filename={$nom_fichier}.pdf";
$params['Pragma'] = 'no-cache';
$params['Cache-Control'] = 'no-store, no-cache, must-revalidate';
$params['Expires'] = 'Sat, 26 Jul 1997 05:00:00 GMT';
```

Je vous recommande de faire très attention à la syntaxe des différents paramètres ci-dessus, car le protocole HTTP est normalisé : la casse, l'emplacement des espaces, virgules et autres points virgules doivent être

scrupuleusement respectés, sous peine de produire un entête HTTP non conforme et surtout incompréhensible par les navigateurs.

Sachant que cette portion de code est susceptible d'être réutilisée pour d'autres modules de l'application, vous pouvez refactoriser ce code en le plaçant dans une classe spécifique, histoire aussi d'alléger le code de votre route.

Mais au fait, on pourrait « plugger » notre nouvelle route sur la page « listebd-crud » par exemple. OK, let's go !

Modifions le début du template « listebd-crud.html.twig » en ajoutant la ligne en gras suivante, ce qui aura pour effet d'ajouter un bouton « PDF » à droite du bouton « Créer » :

```
{% block content %}
<div class="listebd_crud">
    <h2>Albums de BD (CRUD)</h2>

    <a class="btn btn-success btn-sm" href="{{ path('albumupdate', {'id':0,
'crud':'C'}) }}">Créer</a>
    <a class="btn btn-success btn-sm" href="{{ path('listebd-pdf' ) }}"><span
class="glyphicon glyphicon-download-alt"> PDF</span></a>

<table class="table table-striped">
```

Voyons ce que ça donne côté navigateur :

Albums de BD (CRUD)

Créer		PDF			
#	Titre	Auteur(s)	Editeur	Parution	
1	Garulfo	Ayroles, Maïorana, Leprévost	Delcourt	15/05/2011	Afficher Modifier Supprimer
2	Horologiom	Fabrice Lebeault	Delcourt	01/01/2005	Afficher Modifier Supprimer

Cliquez sur le bouton « PDF » et vérifiez que vous obtenez bien une jolie liste au format PDF. Si cela ne fonctionne pas, vérifiez la log des erreurs PHP pour voir si vous y trouvez un indice.

N'oubliez pas de « commiter » votre travail dans Git, avant de poursuivre.

5.4. Export XML

Pour l'export au format XML, nous allons reprendre le principe utilisé pour l'export PDF, mais en le simplifiant quelque peu, puisque cette fois nous nous appuierons sur l'extension DOM, qui est activée par défaut dans tout environnement _AMP (WAMP, XAMP, MAMP..) qui se respecte.

Voici le code source de notre nouvelle route dédiée à l'export au format XML :

```
$app->get('/listebd-xml/', function () use ($app) {
    require_once 'tempdata/liste_bd_temp.php';
    $books = getListeBD();

    $doc = new DOMDocument("1.0", "UTF-8");

    $r = $doc->createElement("books");
    $doc->appendChild($r);

    foreach ($books as $book) {
        $b = $doc->createElement("book");
        $author = $doc->createElement("author");
        $author->appendChild(
            $doc->createTextNode($book['auteur'])
        );
        $b->appendChild($author);
        $title = $doc->createElement("title");
        $title->appendChild(
            $doc->createTextNode($book['album'])
        );
        $b->appendChild($title);
        $publisher = $doc->createElement("publisher");
        $publisher->appendChild(
            $doc->createTextNode($book['editeur'])
        );
        $b->appendChild($publisher);
        $datepub = $doc->createElement("publication_date");
        $datepub->appendChild(
            $doc->createTextNode($book['parution'])
        );
        $b->appendChild($datepub);
        $r->appendChild($b);
    }

    // Définition des entêtes HTTP qui vont permettre au navigateur
    // de comprendre que le document reçu est un fichier XML
    $nom_fichier = 'listeBD';
```

```

$params = array();
$params['Content-type'] = 'application/xml; charset=UTF-8';
$params['Content-Disposition'] = "attachment; filename={$nom_fichier}.xml";

$params['Pragma'] = 'no-cache';
$params['Cache-Control'] = 'no-store, no-cache, must-revalidate';
$params['Expires'] = 'Sat, 26 Jul 1997 05:00:00 GMT';

$response = new Response();
foreach ($params as $paramkey => $paramval) {
    $response->headers->set($paramkey, $paramval);
}
$response->setStatusCode(200, 'OK');
$response->setContent($doc->saveXML());

return $response;
})
->bind('listebd-xml')
;

```

Attention, à la fin du code, ne placez surtout pas un « `send()` » avant le `return`, car cela aurait pour effet de renvoyer le XML deux fois. Soit vous faites un « `return response` », soit vous faites un « `response->send()` » suivi d'un « `return false` », mais surtout pas ça :

```

$response->send(); //=> BAD
return $response; //=> BAD

```

Testez cette nouvelle route dans votre navigateur :

<http://localhost/firstsilex/web/listebd-xml/>

Si tout s'est bien passé, vous devriez retrouver un fichier « `listebd.xml` » dans l'espace « Téléchargements » (ou « Downloads ») de votre système. Certains navigateurs vous proposent même d'ouvrir le fichier, plutôt que de l'enregistrer.

Vous constaterez à la lecture du code source, que nous avons là encore manipulé les entêtes HTTP de notre réponse, et principalement le paramètre « `Content-type` », ici légèrement différent de ce que nous avons défini pour l'export PDF. Sachant que cette portion de code (relative à la manipulation des entête HTTP) est susceptible d'être réutilisée pour d'autres modules de l'application, vous pouvez refactoriser ce code en le plaçant dans une classe spécifique, ce qui permettra d'alléger le code de votre nouvelle route.

Pour un approfondissement concernant la manipulation de fichiers XML (en lecture et en écriture), je vous recommande la lecture de l'article suivant de Jack Herrington, toujours aussi bon 11 ans après sa parution :

<http://www.ibm.com/developerworks/opensource/library/os-xmldomphp/>

Pour finaliser cette partie, modifions le début du template « listebd-crud.html.twig » en ajoutant la ligne en gras suivante, ce qui aura pour effet d'ajouter un bouton « XML » à droite du bouton « PDF » :

```
{% block content %}
<div class="listebd_crud">
    <h2>Albums de BD (CRUD)</h2>

    <a class="btn btn-success btn-sm" href="{{ path('albumupdate', {'id':0,
'crud':'C'}) }}">Créer</a>
    <a class="btn btn-success btn-sm" href="{{ path('listebd-pdf' ) }}"><span
class="glyphicon glyphicon-download-alt"> PDF</span></a>
    <a class="btn btn-success btn-sm" href="{{ path('listebd-xml' ) }}"><span
class="glyphicon glyphicon-download-alt"> XML</span></a>

<table class="table table-striped">
```

Voyons ce que ça donne côté navigateur :

Albums de BD (CRUD)

<div> Créer PDF XML </div>				
#	Titre	Auteur(s)	Editeur	Parution
1	Garulfo	Ayroles, Maïorana, Leprévost	Delcourt	15/05/2011
2	Horologiom	Fabrice Lebeault	Delcourt	01/01/2005

Testez ce nouveau bouton... ça marche ? Oui ? « Pretty cool !! » diraient nos amis anglophones.

Si l'on prend un peu de recul par rapport à ce que nous venons de faire, on peut dire que cette fonction d'export XML constitue votre premier pas dans l'univers des API. En effet, beaucoup d'API sont conçues pour produire en sortie des données aux formats texte, texte qui pourra contenir du XML, du JSON, du CSV ou autre.

N'oubliez pas de « commiter » votre travail dans Git, avant de poursuivre.

5.5. Export JSON

L'export au format JSON va être une promenade de santé, après ce que vous venez de vivre. Voici le code, je ne le détaille même pas :

```
$app->get('/listebd-json/', function () use ($app) {
    require_once 'tempdata/liste_bd_temp.php';
    $books = getListeBD();

    $json = json_encode($books);

    // Définition des entêtes HTTP qui vont permettre au navigateur
    // de comprendre que le document reçu est un fichier XML
    $nom_fichier = 'listeBD';

    $params = array();
    $params['Content-type'] = 'text/plain; charset=UTF-8';
    //$params['Content-Disposition'] = "attachment; filename={$nom_fichier}.json";
    $params['Pragma'] = 'no-cache';
    $params['Cache-Control'] = 'no-store, no-cache, must-revalidate';
    $params['Expires'] = 'Sat, 26 Jul 1997 05:00:00 GMT';

    $response = new Response();
    foreach ($params as $paramkey => $paramval) {
        $response->headers->set($paramkey, $paramval);
    }
    $response->setStatusCode(200, 'OK');
    $response->setContent($json);

    return $response;
})
->bind('listebd-json')
;
```

Testez-le sur votre navigateur :

<http://localhost/firstsilex/web/listebd-json/>

Ca roule ? C'est cool !

Et hop, encore un petit « commit » !! ☺.

5.5. Export CSV

Pour obtenir un export de notre liste d'albums de bande dessinée au format CSV, donc pouvant être ouvert facilement dans Excel, il nous faut ajouter une nouvelle route :

```
$app->get('/listebd-csv/', function () use ($app) {
    require_once 'tempdata/liste_bd_temp.php';
    $books = getListeBD();
    $csv = 'titre;auteur;editeur;parution'.PHP_EOL;
    foreach ($books as $book) {
        $csvdata = array();
        foreach (array_values($book) as $tmpdata) {
            // c'est moche de faire un "utf8_decode" mais si on veut
            // que les utilisateurs puissent ouvrir facilement le fichier
            // avec Excel, on n'a pas le choix.
            // (Libre Office sait importer de l'UTF-8 sans problème)
            $csvdata [] = ' '.trim(utf8_decode($tmpdata)).' ';
        }
        $csv .= implode($csvdata, ';'). PHP_EOL;
    }

    // Définition des entêtes HTTP qui vont permettre au navigateur
    // de comprendre que le document reçu est un fichier XML
    $nom_fichier = 'listeBD';

    $params = array();
    $params['Content-type'] = 'application/vnd.ms-excel';
    $params['Content-Disposition'] = "attachment; filename={$nom_fichier}.csv";
    $params['Pragma'] = 'no-cache';
    $params['Cache-Control'] = 'no-store, no-cache, must-revalidate';
    $params['Expires'] = 'Sat, 26 Jul 1997 05:00:00 GMT';

    $response = new Response();
    foreach ($params as $paramkey => $paramval) {
        $response->headers->set($paramkey, $paramval);
    }
    $response->setStatusCode(200, 'OK');
    $response->setContent($csv);

    return $response;
})
->bind('listebd-csv')
;
```

Le fichier CSV ouvert dans Excel :

	A	B	C	D	E
1	titre	auteur	editeur	parution	
2		1 Garulfo	Ayroles, Maïorana, Leprévost	Delcourt	15/05/2011
3		2 horologiom	fabrice Lebeault	Delcourt	01/01/2005
4		3 Le château des étoiles	Alex Alice	Rue De Sevres	01/05/2014
5		4 Le voyage extraordinaire	Camboni, Filippi	Vents d'Ouest	01/09/2012
6					

Vous noterez le commentaire que j'ai ajouté à l'intérieur d'une des boucles « foreach » :

```
foreach (array_values($book) as $tmpdata) {
    // c'est moche de faire un "utf8_decode" mais si on veut
    // que les utilisateurs puissent ouvrir facilement le fichier
    // avec Excel, on n'a pas le choix.
    // (Libre Office sait importer de l'UTF-8 sans problème)
    $csvdata [] = ' '.trim(utf8_decode($tmpdata)).' ';
}
```

Eh oui, quelquefois on est obligé de mettre des « rustines », pour obtenir rapidement une solution qui fonctionne.

Le format CSV n'est pas normalisé, du coup on peut utiliser la virgule ou le point-virgule comme caractère séparateur. J'ai remarqué que Excel interprète mieux les fichiers CSV contenant des points-virgules comme caractères séparateurs. De plus, pour les libellés composés de plusieurs mots, il est vivement recommandé de les « encadrer » avec des guillemets ou des apostrophes. Dans l'exemple précédent, j'ai utilisé les guillemets comme caractères d'encadrement, car je savais que les libellés provenant de ma table fictive n'en contenaient pas. S'ils en contenaient, je serais dans l'obligation de les « échapper ».

Le format CSV est assez pratique. Il est reconnu par Microsoft Excel et par Libre Office, il est facile à lire et à écrire en PHP, et dans de nombreux autres langages, bref c'est un format qui est encore souvent utilisé par les utilisateurs en entreprise. Mais on ne peut pas personnaliser sa présentation comme on le ferait avec un fichier Excel. Du coup, les utilisateurs sont souvent demandeurs de solutions d'export plus sophistiquées que ce que nous venons de mettre en œuvre. Il convient dans ce cas de se tourner vers des librairies permettant de

produire de vrais fichiers XLS (ou XLSX). C'est le cas notamment du projet PHPExcel :

<https://phpexcel.codeplex.com>

Vous pouvez ajouter une nouvelle option d'export dans le template « listebd-crud.html.twig », juste après le bouton « Export XML » :

```
<a class="btn btn-success btn-sm" href="{{ path('listebd-csv' ) }}"><span class="glyphicon glyphicon-download-alt"> CSV</span></a>
```

Pensez à mettre à jour votre historique Git avec toutes ces modifications.

6. ANNEXES

6.1. Le plugin Datepicker

Le plugin jQueryUI Datepicker permet de bénéficier de champs de saisie de type « date » homogène sur l'ensemble des navigateurs.

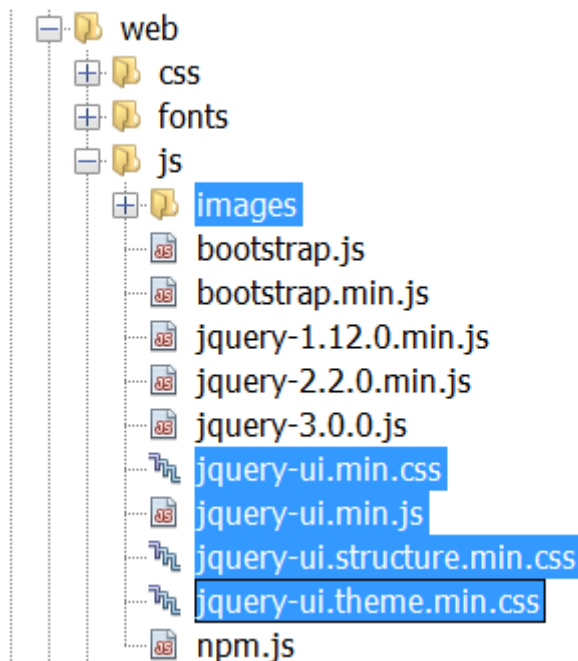
On peut tester ce plugin à partir de la page suivante :

<https://jqueryui.com/datepicker/>

On peut télécharger tout ou partie de jQueryUI à partir de son site officiel :

<https://jqueryui.com/>

Une fois le zip récupéré localement, on va pouvoir ajouter à notre projet SILEX les fichiers et répertoires suivants (surlignés en bleu) :



On va ensuite ajouter à notre fichier « layout.html.twig », les lignes en gras suivantes :

```
<script src="{ asset('js/jquery-1.12.0.min.js') }" ></script>
<script src="{ asset('js/bootstrap.min.js') }" ></script>
<script src="{ asset('js/jquery-ui.min.js') }" ></script>
<link href="{ asset('js/jquery-ui.min.css') }" rel="stylesheet" type="text/css" />
```

Enfin, pour pouvoir activer le « datepicker » pour l'ensemble de nos formulaires, on ajoutera à la fin du même fichier « layout.html.twig » les lignes de code Javascript suivantes (en gras) :

```
<footer class="footer">
    <div class="container">
        <p class="text-muted">Premier projet avec SILEX</p>
    </div>
</footer>
<script>
    $(document).ready(function() {
        $('<strong>.js-datepicker</strong>').datepicker({
            dateFormat: 'yy-mm-dd'
        });
    });
</script>
</body>
</html>
```

On devra veiller à bien paramétrer le format de la date au niveau du « datepicker », en fonction du format de nos dates telles qu'elles sont récupérées à partir de la base de données. Dans l'exemple ci-dessus, les dates transmises au formulaire doivent avoir la structure suivante : AAA/MM/DD. Le choix, en termes de formats de date, du plugin « datepicker » étant limité, il sera nécessaire de convertir le format des dates pour les adapter au plugin, plutôt que l'inverse (plus difficile à réaliser).

L'un des formats de date qui fonctionne correctement avec « datepicker » étant le format « yy/mm/dd », nous devons veiller à formater les dates provenant de notre base de données (ou de notre fonction getListeBD), de la même manière avant de les envoyer au formulaire.

Cette contrainte nous a conduit à modifier la boucle « foreach » dans la partie du code traitant les requêtes HTTP de type GET :

```

if($request->getMethod() == 'GET'){
    $id = (int)$request->get('id');

    /* ... Version « allégée » pour la lisibilité de l'exemple ... */

    } else {
        if (!array_key_exists($id, $albums)) {
            // redirection
            Return $app->redirect($app['url_generator']
                ->generate('albumotfound'));
        } else {
            foreach($albums[$id] as $key=>$value) {
                if ($key == 'parution') {
                    // la date de parution doit être formatée selon le
                    // format attendu par le plugin jQuery
                    $tmpdate = new DateTime($value);
                    $data[$key] = $tmpdate->format('Y-m-d');
                } else {
                    $data[$key] = $app->escape($value);
                }
            };
            // On ajoute la notion de CRUD à notre jeu de données
            $data['crud'] = $crud ;
        }
    }
}

```

Nous avons également dû modifier la partie du code traitant les requêtes HTTP de type POST, pour la même raison :

```

if ($request->getMethod() == 'POST') {
    $form->handleRequest($request);
    $data = $form->getData();
    if ($form->isSubmitted()) {
        // redirection si Return ou Si Formulaire valide
        if (isset($_POST['form']['return'])) {
            return $app->redirect($app['url_generator']
                ->generate('listebd-crud'));
        }
        if ($form->isValid()) {
            return $app->redirect($app['url_generator']
                ->generate('alumbdregister'));
        }
    } else {
        // Date de parution à reformater selon le format défini
        // sur jQueryUI Datepicker
        $data['parution'] = $data['parution']->format('Y-m-d');
    }
}

```

6.2. Différences de version concernant les formulaires

Il existe quelques différences significatives sur Symfony, concernant la déclaration des assertions. Il vaut mieux le savoir si vous voulez réutiliser du code provenant d'anciens projets SILEX :

Avant la version 2.1

```
->add('auteur', 'text', array(
    'constraints' => array(
        new Assert\NotNull(array(
            'message' =>
                'Veuillez saisir l\'auteur de l\'album.'
        )),
        new Assert\MinLength(array(
            'limit' => 2,
            'message' =>
                'Le nom de l\'auteur doit comporter au minimum deux caractères.'
        )),
        new Assert\MaxLength(array(
            'limit' => 30,
            'message' =>
                'la longueur du nom de l\'auteur ne peut excéder 30 caractères.'
        ))
    )
))
)
```

A partir de la version 2.1

```
->add('auteur', 'text', array(
    'constraints' => array(
        new Assert\NotNull(array(
            'message' => 'Veuillez saisir l\'auteur de l\'album.'
        )),
        new Assert\Length(array(
            'min' => 2,
            'minMessage' => 'Le titre doit comporter au minimum deux
caractères.',
            'max' => 30,
            'maxMessage' => 'la longueur du titre ne peut excéder 30
caractères.'
        )),
    )
))
)
```

6.3. Versions des composants PHP utilisés

Dans le cas où vous rencontreriez des difficultés en suivant ce support de cours, il n'est pas impossible que ces difficultés soient dûes à des problèmes de version. Pour vous aider dans vos recherches, vous trouverez ci-dessous le fichier `composer.json` que j'ai utilisé sur ma version du projet :

```
{
  "name": "fabpot/silex-skeleton",
  "description": "A pre-configured skeleton for the Silex microframework",
  "license": "MIT",
  "type": "project",
  "require": {
    "php": ">=5.5.9",
    "silex/silex": "~2.0",
    "silex/web-profiler": "~2.0",
    "symfony/asset": "~2.8|3.0.*",
    "symfony/browser-kit": "~2.8|3.0.*",
    "symfony/class-loader": "~2.8|3.0.*",
    "symfony/config": "~2.8|3.0.*",
    "symfony/console": "~2.8|3.0.*",
    "symfony/css-selector": "~2.8|3.0.*",
    "symfony/debug": "~2.8|3.0.*",
    "symfony/finder": "~2.8|3.0.*",
    "symfony/form": "~2.8|3.0.*",
    "symfony/monolog-bridge": "~2.8|3.0.*",
    "symfony/process": "~2.8|3.0.*",
    "symfony/security": "~2.8|3.0.*",
    "symfony/translation": "~2.8|3.0.*",
    "symfony/twig-bridge": "~2.8|3.0.*",
    "symfony/validator": "~2.8|3.0.*",
    "voilab/tctable": "^1.2",
    "doctrine/dbal": "^2.5"
  },
  "autoload": {
    "psr-0": { "" : "src/" }
  },
  "extra": {
    "branch-alias": {
      "dev-master": "2.0.x-dev"
    }
  },
  "scripts": {
    "run": [
      "echo 'Started web server on http://localhost:8888'",
      "php -S localhost:8888 -t web web/index_dev.php"
    ]
  }
}
```

6.4. Paginer avec classe

Voici le code source de la classe utilisée pour gérer la pagination des listes, dans le cadre de ce support de cours.

```
<?php

/*
 * Classe permettant de générer une barre de pagination
 * Très largement inspirée de l'exemple proposé dans le livre :
 *   PHP Cookbook (2nd Edition), par Adam Trachtenberg et David Sklar, O'Reilly (2006)
 * D'importantes modifications ont été apportées au code initial, telles que :
 * - le regroupement des 2 fonctions du livre dans une classe
 * - la possibilité de passer la page d'appel aux 2 méthodes, ceci afin de faciliter
 *   la réutilisation de ces 2 méthodes sur différentes pages
 * - il a été nécessaire d'ajouter un tableau $params permettant de transmettre d'une
 *   page à l'autre des paramètres autres que l'offset, tels que les critères de
 *   sélection saisis sur un formulaire de recherche par exemple.
 * - le nombre de pages directement "appelables" a été limité à 5, des points de suspension
 *   sont ajoutés ensuite, et le lien vers la dernière page est ajouté en fin de barre de
 *   pagination (la version initiale proposait un lien vers chaque page, ce qui
 *   donnait des résultats particulièrement laids sur des jeux de données de grande taille.
 * - la prise en compte des spécificités de l'architecture MVC dans la manière
 *   de générer l'URL de chacun des éléments de la barre de navigation
 * - la possibilité de générer une barre de pagination selon le style de Bootstrap
 *
 * Exemple d'utilisation :
 *     $barre_pagination = new \Pagination($nblines['comptage'], $offset,
 *                                     $nbl_par_page, '', array(), true);
 *
 * @author Gregory Jarrige
 * @version 0.1 (2016-09-25)
 */

class Pagination {

    private $total ;
    private $offset ;
    private $by_page ;
    private $curpage ;
    private $parmpage ;
    private $mvc;

    /**
     * Constructeur d'une barre de pagination
     * @param integer $total
     * @param integer $offset
     * @param integer $by_page
     * @param string $curpage
     * @param array $parmpage
     * @param boolean $mvc
     */
    public function __construct($total, $offset, $by_page, $curpage, $parmpage, $mvc) {
        $this->total = (int)$total;
```

```

        $this->offset = (int)$offset;
        $this->by_page = (int)$by_page;
        $this->curpage = (string)$curpage;
        $this->parmpage = (is_array($parmpage))?$parmpage:array();
        $this->mvc = (boolean)$mvc ;
    }

    private function printLink($inactive, $text, $offset, $current_page,
        $params_page, $bootstrap) {
        // on prépare l'URL avec tous les paramètres sauf "offset"
        if (!isset($offset) or $offset == '' or $offset == '0') {
            $offset = '1';
        }
        $url = '';
        $params_page ['offset'] = $offset;
        if ($this->mvc) {
            $url = implode('/', $params_page);
        } else {
            $url = '?' . http_build_query($params_page);
        }
        $output = '' ;
        $current_page = htmlentities($current_page) ;
        if (!$bootstrap) {
            if ($inactive) {
                $output = "<span class='inactive'>$text</span>".PHP_EOL;
            } else {
                $output = "<span class='active'>" . "<a href='" .
                    $current_page . $url . "'>$text</a></span>".PHP_EOL;
            }
        } else {
            if ($inactive) {
                $output = "<li class='disabled'><a href='#'>$text</a></li>".PHP_EOL;
            } else {
                $output = "<li class='activex'><a href='" .
                    $current_page . $url . "'>$text</a></li>".PHP_EOL;
            }
        }
        return $output ;
    }

    private function indexedLinks($total, $offset, $by_page, $curpage,
        $parmpage, $bootstrap) {
        $separator = ' | ';
        $list_links = [];

        $list_links [] = self::printLink($offset == 1, '<< Précédent',
            $offset - $by_page, $curpage, $parmpage, $bootstrap);

        $compteur = 0;
        $top_suspension = false;

        // affichage de tous les groupes à l'exception du dernier
        for ($start = 1, $end = $by_page; $end < $total; $start += $by_page,
            $end += $by_page) {
            $compteur += 1;
            if ($compteur < 5) {
                if (!$bootstrap) {
                    $list_links [] = $separator;
                }
            }
        }
    }

```

```

        $list_links [] = self::printLink($offset == $start, "$start-$end",
            $start, $curpage, $parmpage, $bootstrap);
    } else {
        if (!$top_suspension) {
            $top_suspension = true;
            if (!$bootstrap) {
                $list_links [] = ' | ... ';
            } else {
                $list_links [] = '<li class="disabled"><a href="#"> ... </a></li>';
            }
        }
    }
}

$end = ($total > $start) ? '-' . $total : '';

if (!$bootstrap) {
    $list_links [] = $separator;
}
$list_links [] = self::printLink($offset == $start, "$start$end",
    $start, $curpage, $parmpage, $bootstrap);

if (!$bootstrap) {
    $list_links [] = $separator;
}
$list_links [] = self::printLink($offset == $start, 'Suiv. >>',
    $offset + $by_page, $curpage, $parmpage, $bootstrap);

return $list_links;
}

public function navBarText() {
    $links = self::indexedLinks($this->total, $this->offset,
        $this->by_page, $this->curpage, $this->parmpage, false);
    return implode(' ', $links);
}

public function navBarBootstrap() {
    $links = self::indexedLinks($this->total, $this->offset,
        $this->by_page, $this->curpage, $this->parmpage, true);
    $html = '<nav><ul class="pagination">'.PHP_EOL;
    $html .= implode(' ', $links);
    $html .= '</ul></nav>'.PHP_EOL;
    return $html;
}
}

```