PP\_Currency (test project) on SILEX micro-framework  
  
1. Installation and configuration  
2. Creation of the first Silex page “Hello World”  
3. Providing a first REQUEST  
4. Creation of the Form with validation and Helpers  
5. Let’s finish the CONVERTER – providing a correct result  
6. Using Foundation css framework to change the Application style

**Installation and configuretion**  
  
**1.1. Silex .zip project installation**  
 Download a latest Silex “fat” version as a zip file from the official silex download page.  
 Probably you should select a fat archive:  
 <http://silex.sensiolabs.org/download>  
   
 Unzip it into your future project directory. For me this is

*C:\export\www\test\silex.pp\_currency*   
  
**1.2. Silex composer instalation** …..

**2. Configuration**  
 Create .htaccess file into the root directory of the project:

<IfModule mod\_rewrite.c>

Options -MultiViews

RewriteEngine On

#RewriteBase /path/to/app

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteRule ^ index.php [QSA,L]

</IfModule>  
  
Add a project name to the windows *hosts* file (*C:\Windows\System32\Drivers\etc*):  
  
 *127.0.0.1 silex.pp\_currency*  
  
 After that it should be set in the vhost.conf file, where the redirection will be executed:  
  
 <VirtualHost \*:80>

ServerName silex.pp\_currency

ServerAlias silex.pp\_currency

DocumentRoot /export/www/test/silex.pp\_currency/web

DirectoryIndex index.php index.html

<Directory "/export/www/test/silex.pp\_currency/web">

RewriteEngine on

RewriteCond $1 !^(index\.php)

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteBase /

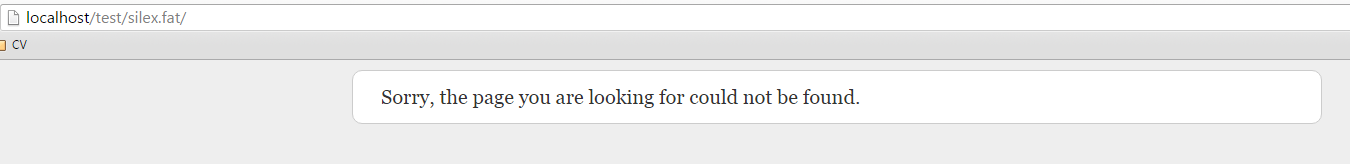
RewriteRule ^(.\*)$ index.php?q=$1 [L,QSA]

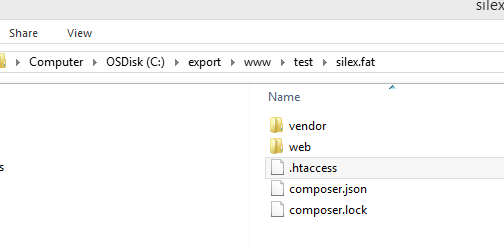
EnableSendfile Off

EnableMMAP Off

</Directory>

</VirtualHost>

Restart apche service. In PuTTy:  
  
 sudo service apache2 restart  
  
 Open the project in your browser:  
  
 <http://silex.pp_currency/>   
  
 If everything was ok you should see the following page:  
 

**3. Files structure**  
 On this step the project’s file structure should look like:  
  
   
 More silex installation and configuration information you can get from the  
 official silex documentation pages:  
   
 <http://silex.sensiolabs.org/download>   
 <http://silex.sensiolabs.org/doc/master/web_servers.html>

**Creation of the first silex page “Hello world”**  
**1. Into the existing project I am adding a few new directories and files: /*app* and /*src* with new files in them**  
  
 **/app**

**app.php  
 bootstrap.php  
 di.php  
 routes.php  
 /src  
 / PP\_Currency  
 /Controllers  
 CurrencyController.php  
 /Models  
 /Providers  
 ControllerProvider.php  
 /Views  
 index.html.twig**

**2. In the main application file (/app/app.php) we are including the bootstrap file,  
 registering twig provider, defining paths to Views and cache directories  
 and registering a controller provider**

/app/app.php

use Symfony\Component\HttpFoundation\Request,

Symfony\Component\HttpFoundation\Response;

require\_once \_\_DIR\_\_.'/bootstrap.php';  
Symfony\Component\Debug\ErrorHandler::register();

/\* here I am configuring twig views directory path and cache directory path \*/

$app = new Silex\Application();

$app->register(

new Silex\Provider\TwigServiceProvider(),

array(

'twig.path' => \_\_DIR\_\_ . '/../src/PP\_Currency/Views',

'twig.options' => array(

'cache' => \_\_DIR\_\_ . '/../cache',

)

)

);

$app->register(new Silex\Provider\SessionServiceProvider());  
/\* this line is important to be included to provide controllers work \*/

$app->register(new Silex\Provider\ServiceControllerServiceProvider());

$app->register(new PP\_Currency\Providers\ControllerProvider());

require\_once \_\_DIR\_\_.'/di.php';

$app['debug'] = true;

require\_once \_\_DIR\_\_ . '/../app/routes.php';

return $app;

**3. In the /app/bootsrap.php file I am providing booting of the autoloader   
 and the application namespace ‘PP\_Currency’**

/app/bootstrap.php  
  
//require\_once \_\_DIR\_\_.'/../vendor/autoload.php';  
/\* this is applying a namespase without a composer \*/

$loader = require\_once \_\_DIR\_\_.'/../vendor/autoload.php';

$loader->add('PP\_Currency', \_\_DIR\_\_ . '/../src');

**4. Providing triggering of the controller depending on the routing**

/app/routes.php  
  
$app->get('/', 'currency.controller:index');

**5. In the controller provider we are registering the currency controller**

/src/PP\_Currency/Providers/ControllerProvider.php  
  
namespace PP\_Currency\Providers;

use Silex\Application,

Silex\ServiceProviderInterface,

PP\_Currency\Controllers\CurrencyController;

class ControllerProvider implements ServiceProviderInterface

{

public function register(Application $app)

{

$app['currency.controller'] = $app->share(

function () use ($app) {

return new CurrencyController($app);

}

);

}

public function boot(Application $app){}

}

**6. In the currency controller we are setting the twig variable ‘test\_hello\_var’   
 to demo that the application is working**

/src/PP\_Currency/Controllers/CurrencyController.php  
  
namespace PP\_Currency\Controllers;

class CurrencyController

{

protected $app;

protected $twig;

protected $request;

public function \_\_construct($app)

{

$this->app = $app;

$this->twig = $app['twig'];

$this->request = $app['request'];

}

public function index()

{  
 /\* here is adding a twig variable to be shown in the index.twig.html file \*/

return $this->twig->render(

'/index.html.twig',

array(

'test\_hello\_var' => 'Hello dear Alexander!!!!'

)

);

}

}

**7. In the dependency injections file (di) I am sharing dependencies.  
For a while it is loading of the views files**

/app/di.php

…

$app['twig.loader'] = $app->share(

function () use ($app) {

return new Twig\_Loader\_Filesystem(\_\_DIR\_\_ . '/../src/PP\_Currency/Views');

}

);

…

**8. In the view file we are layouting this variable**

/src/PP\_Currency/Views/index.html.twig  
  
<!doctype html>

<!--[if lt IE 7]> <html class="no-js lt-ie9 lt-ie8 lt-ie7" lang="en"> <![endif]-->

<!--[if IE 7]> <html class="no-js lt-ie9 lt-ie8" lang="en"> <![endif]-->

<!--[if IE 8]> <html class="no-js lt-ie9" lang="en"> <![endif]-->

<!--[if gt IE 8]><!--> <html class="no-js" lang="en"> <!--<![endif]-->

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<title>PP Currency :: index</title>

<meta name="description" content="">

<meta name="author" content="">

<link rel="stylesheet" href="{{ app.request.getSchemeAndHttpHost }}/assets/css/style.css">

</head>

<body>

<h2>{{ test\_hello\_var }}</h2>

</body>

</html>

**9. The silex loading scheme is looking like that**

web/Index.php

app.php

di.php [dependency injection]

routes.php

Index.twig.html

bootstrap.php

Register  
MVC providers,  
Cache etc

autoload.php

Controllers,  
Models

Loading of classes and  
dependencies

Brief explaination of the scheme:  
\* From a browser a client is calling the project: <http://silex.pp_currency/>  
 According the the vhosts.cong file the file **/web/index.php** will be called;  
  
\* The **index.php** will include the **app.php** file, will create the *app* instance of it and   
 will it run;  
  
\* The **app.php** file will include **bootstrap.php** (which will include the **autoload.php**),  
 will trigger the ***Silex\Application*** constructor to create the *app* object and will return it,  
 the *app* object will the *TwigServiceProvider* and will set the cache path.  
 As well it will register controller providers,   
 will include **di.php** *[dependency injection]* file – it is making instances of dependencies:   
 repository, transporter etc,  
 will include **routes.php** – routing of the GET URLs and controllers  
   
 $app = new Silex\Application();  
  
 $app->register(

new Silex\Provider\TwigServiceProvider(),

array(

'twig.path' => \_\_DIR\_\_ . '/../src/PP\_Currency/Views',

'twig.options' => array(

'cache' => \_\_DIR\_\_ . '/../cache',

)

)

);

….   
 $app->register(new Silex\Provider\ServiceControllerServiceProvider());  
 $app->register(new PP\_Currency\Providers\ControllerProvider());

require\_once \_\_DIR\_\_.'/di.php';  
 require\_once \_\_DIR\_\_ . '/../app/routes.php';  
 …..  
 return $app;  
  
 \* **autoload.php** is loading all necessary vendor classes and dependencies;  
  
 \* When **app.php** gets a *GET* request it is checking it with **routes.php**   
 and is calling a related with the current URL *controller*;  
  
 \* The *Controller* is handle the URL, calling needle *Action* and  
 render a needle *twig template*, passing there necessary data;   
  
 \* The *twig html template* will be shown in the clients browser;

**Providing a first REQUEST**To continue working on the application I need to add more functionalities.  
It will be adding configuration files and adding additional app nodes as Transporter and Repository to make my first api request as well.

1. **Adding a configuration file(s) with remote api’s URLs**

***/app  
 /config  
 config.php  
 database.php  
 app.php  
 bootstrap.php  
 di.php  
 routes.php***

For now I do not need the database.php so I am going to leave it empty or  
 I can populate it using default settings.  
   
 /app/config/database.php  
  
 /\*\*\* Database \*\*\*/

return array(

'driver' => 'mysql',

'host' => 'localhost',

'database' => 'test',

'username' => 'root',

'password' => '',

'charset' => 'utf8',

'collation' => 'utf8\_unicode\_ci',

'prefix' => null

);

The another one will contain main configuration info so I need to populate it with  
 real data. In my Application I want to use free API provided by **Fixer.io**.   
 This is a free JSON API for current and historical foreign exchange rates   
 published by the European Central Bank. More information you can get from the   
 API’s official website <http://fixer.io/>.  
  
 /app/config/config.php  
  
 return array(

'GET\_REQUEST\_URL' => array(

'latest\_rates' => 'http://api.fixer.io/latest',

'rates\_by\_date' => 'http://api.fixer.io/{yyyy-mm-dd}',

'rates\_by\_base' => 'http://api.fixer.io/latest?base={CURRENCY\_CODE}',

'rates\_by\_codes' => 'http://api.fixer.io/latest?symbols={COMMA\_SEP\_CODES}',

'latest\_rates\_cross\_domains' => 'http://api.fixer.io/latest?callback=?'

),

'SSL\_GET\_REQUEST\_URL' => array(

'latest\_rates' => 'https://api.fixer.io/latest',

'rates\_by\_date' => 'https://api.fixer.io/{yyyy-mm-dd}',

'rates\_by\_base' => 'https://api.fixer.io/latest?base={CURRENCY\_CODE}',

'rates\_by\_codes' => 'https://api.fixer.io/latest?symbols={COMMA\_SEP\_CODES}',

'latest\_rates\_cross\_domains' => 'https://api.fixer.io/latest?callback=?'

),

'controller' => '',

'language' => '',

'curl\_connect\_timeout' => 3,

'curl\_request\_timeout' => 10,

'curl\_request\_debug' => 0

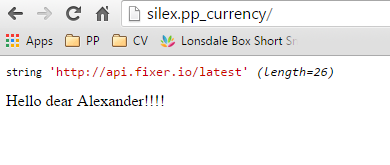
);  
   
 Here you can see that the file returns an array with configuration data.  
 For a while I will need two its elements only or to be honst one of them.   
 It is depending on which protocol I will use: HTTP or HTTPS.  
 So I will use simple HTTP and it means I will need the GET\_REQUEST\_URL data array.  
  
 Now when the configuration files are ready I need to include both of them into my   
 application clients code. Of course it should be included in the **app.php** where before   
 that I included the **bootstrap.php** and the **routes.php**.  
  
 /app/app.php  
  
 $app = new Silex\Application();

….  
  
 $app['config'] = require\_once CONFIG\_PATH.'/config.php';

$app->register(new Silex\Provider\SessionServiceProvider());  
 $app->register(new Silex\Provider\ServiceControllerServiceProvider());  
 $app->register(new PP\_Currency\Providers\ControllerProvider());

…..  
  
return $app;  
  
Here I am using the constant CONFIG\_PATH which I defined in the bootstrap.php.  
  
/app/bootstrap.php  
  
define('CONFIG\_PATH', \_\_DIR\_\_.'/config');  
  
The last one I’d like to do is to layout the configuration information to check it is accessable. For that I will trigger the following code inside the index action of the CurrencyController.  
  
/src/PP\_Currency/Controllers/CurrencyController.php  
  
namespace PP\_Currency\Controllers;

class CurrencyController

{  
 …  
 public function index() {  
 ….  
 var\_dump($this->config['GET\_REQUEST\_URL']['latest\_rates']);  
 ….  
 }  
   
Let’s run the application in the browser <http://silex.pp_currency/> and we have to see the ‘latest\_rates’ element’s value:  
  


1. **To my basic code directory /src/PP\_Currency I am adding two new ones:  
   *Repository* and *Transporter  
     
   /src  
    / PP\_Currency  
    …  
    /Transporter***

***Curl.php  
 TransporterInterface.php  
 /Repository  
 CurrencyRepository.php  
 CurrencyRepositoryInterface.php***

The Transporter directory will execute transporting functionalities. In my opinion It should contain two files: *TransporterInterface.php* and *Curl.php*  
  
In the *TransporterInterface.php* I am defining an interface with all necessary methods to be used in cURL operations.  
The *Curl.php* will define a class Curl wich will implement the Interface and realize the methods.  
  
/src/PP\_Currency/Transporter/TransporterInterface.php  
  
namespace PP\_Currency\Transporter;

interface TransporterInterface

{

public function exec();

public function setOpt($key, $value);

public function getOpt($key);

public function close();

public function getInfo();

public function post($url, $params);

public function get($url);

public function getContent($url);

}

/src/PP\_Currency/Transporter/Curl.php  
  
namespace PP\_Currency\Transporter;

/\*\*

\* Generic class that will perform curl requests

\* Class Curl

\* @package Colosseum\Transporter

\*/

class Curl implements TransporterInterface

{

private $handler;

private $opts = array();

protected $app;

public function \_\_construct($app)

{

$this->app = $app;

}

private function getHandler()

{

if (!is\_null($this->handler)) {

return $this->handler;

}

$this->handler = curl\_init();

if (!$this->handler) {

throw new \Exception('cURL session cannot be created');

}

return $this->handler;

}

/\*\*

\* Execute the request

\* @return string

\* @throws \Exception

\*/

public function exec()

{

$handler = $this->getHandler();

$result = curl\_exec($handler);

if($this->app['config']['curl\_request\_debug'] == 1) {

$this->debug();

}

return $result;

}

/\*\*

\* Debug the request

\* @return array

\*/

public function debug()

{

echo '<pre>';

print\_r($this->getInfo());

echo '</pre>';

}

/\*\*

\* Set curl specific option like these starting with CURLOPT\_\*

\* @param integer $key

\* @param string $value

\* @return $this

\*/

public function setOpt($key, $value)

{

$this->opts[$key] = $value;

curl\_setopt($this->getHandler(), $key, $value);

return $this;

}

/\*\*

\* Get curl option that was previously set with setOpt($key, $value)

\* @param integer $key

\* @return mixed

\*/

public function getOpt($key)

{

if (isset($this->opts[$key])) {

return $this->opts[$key];

}

}

/\*\*

\* Free current curl resource previously created with curl\_init()

\*/

public function close()

{

$handler = $this->getHandler();

if (is\_resource($handler)) {

$this->handler = null;

curl\_close($handler);

}

}

public function getInfo()

{

$info = curl\_getinfo($this->getHandler());

return $info;

}

public function getContent($url) {

$result = file\_get\_contents($url);

var\_dump($result);

return $result;

}

public function get($url)

{

$this

->setOpt(CURLOPT\_URL, $url)

->setOpt(CURLOPT\_RETURNTRANSFER, true)

->setOpt(CURLOPT\_PROXY, 'cntlm-proxy:3128')

->setOpt(CURLOPT\_PROXYAUTH, CURLAUTH\_NTLM)

->setOpt(CURLOPT\_CONNECTTIMEOUT , $this->app['config']['curl\_connect\_timeout'])

->setOpt(CURLOPT\_TIMEOUT, $this->app['config']['curl\_request\_timeout']);

$result = $this->exec();

return $result;

}

public function post($url, $data)

{

$this

->setOpt(CURLOPT\_SSL\_VERIFYPEER, 0)

->setOpt(CURLOPT\_CONNECTTIMEOUT, $this->app['config']['curl\_connect\_timeout'])

->setOpt(CURLOPT\_TIMEOUT, $this->app['config']['curl\_request\_timeout'])

->setOpt(CURLOPT\_URL, $url)

->setOpt(CURLOPT\_POST, count($data))

->setOpt(CURLOPT\_POSTFIELDS, http\_build\_query($data))

->setOpt(CURLOPT\_RETURNTRANSFER, true);

$result = $this->exec();

return $result;

}

public function postJson($url, $data)

{

$this

->setOpt(CURLOPT\_SSL\_VERIFYPEER, 0)

->setOpt(CURLOPT\_CONNECTTIMEOUT , $this->app['config']['curl\_connect\_timeout'])

->setOpt(CURLOPT\_TIMEOUT, $this->app['config']['curl\_request\_timeout'])

->setOpt(CURLOPT\_HEADER, false)

->setOpt(CURLOPT\_URL, $url)

->setOpt(CURLOPT\_HTTPHEADER, array("Content-type: application/json"))

->setOpt(CURLOPT\_POST, true)

->setOpt(CURLOPT\_POSTFIELDS, json\_encode($data))

->setOpt(CURLOPT\_RETURNTRANSFER, true);

$result = $this->exec();

return $result;

}

}

The Repository directory has files which work with data. They send CURL requests, receive responsies, handle data and return them to Controller.

In the Repository directory there are also two files: *CurrencyRepositoryInterface.php* and *CurrencyRepository.php.*

*CurrencyRepositoryInterface.php* is an interface and for a while it has only one method getLatestRates().

/src/PP\_Currency/Repository/CurrencyRepositoryInterface.php  
  
namespace PP\_Currency\Repository;

interface CurrencyRepositoryInterface {

public function getLatestRates($url);

}

*CurrencyRepository.php* has a class CurrencyRepository which implements the interface, triggers CURL methods (using the Curl class) to send requests and returns resulting JSON.

/src/PP\_Currency/Repository/CurrencyRepository.php  
  
namespace PP\_Currency\Repository;

use PP\_Currency\Helpers\ToolsHelper as Tools;

class CurrencyRepository implements CurrencyRepositoryInterface

{

protected $app;

protected $rates = array();

protected $session;

protected $config;

public function \_\_construct($app)

{

$this->app = $app;

$this->config = $app['config'];

$this->session = $app['session'];

}

public function getLatestRates($url)

{

$latestRates = $this->getJson($url);

return $latestRates;

}

protected function getJson($url)

{

$transporter = $this->app['transporter'];

$requestRes = $transporter->get($url);

$result = json\_decode($requestRes, true);

$info = $transporter->getInfo();

$transporter->close();

if ($info['http\_code'] != 200) {

$message = isset($result['message']) ? $result['message'] : 'General rates error';

throw new \Exception($message);

}

return $result;

}

}

After defining the classes I need to plugin them. For that I should creatie instances of the classes in the dependency injection file.

/app/di.php

…  
$app['currency.repository'] = $app->share(

function () use ($app) {

return new \PP\_Currency\Repository\CurrencyRepository($app);

}

);

$app['transporter'] = $app->share(

function () use ($app) {

return new \PP\_Currency\Transporter\Curl($app);

}

);

…In the controller CurrencyController now I need to call the Repository’s method *getCurrencyRepository()* to get the Repository instance (object) and after that I can receive my first JSON of rates via calling *getLatestRates(repositoryObject).* The Resulting JSON I am setting to the twig variable ‘test\_hello\_var’ and rendering html.

/src/PP\_Currency/Controllers/CurrencyController.php

namespace PP\_Currency\Controllers;

use PP\_Currency\Helpers\UrlHelper,

PP\_Currency\Helpers\ToolsHelper;

class CurrencyController

{

protected $app;

protected $twig;

protected $request;

protected $config;

public function \_\_construct($app)

{

$this->app = $app;

$this->twig = $app['twig'];

$this->request = $app['request'];

$this->config = $app['config'];

}

public function index()

{

$currencyRepository = $this->getCurrencyRepository();

$latestRatesByBase = $currencyRepository->getLatestRates($latestRatesByBaseUrl);

/\*

\* GET http://api.fixer.io/latest

\* GET http://api.fixer.io/latest?base=USD

\* base should have a Currency code

\*/

return $this->twig->render(

'/index.html.twig',

array(

'test\_hello\_var' => $latestRatesByBase

)

);

}

protected function getCurrencyRepository()

{

return $this->app['currency.repository'];

}

}

**Creation of the Form with validation and Helpers** In this section I want to prepare request URLs to use them to get working data  
 and also I’d like to create a Form “currencyForm” with four form elements:   
  
 - input text “amount” to type there how much money I want to convert;  
 - dropdown lists “currencyFrom” populated with Currency Codes;  
 - dropdown lists “currencyTo” populated with Currency Codes as well;  
 - submit button “Convert”;

1. **Creating Helpers tp prepare request URLs for using in transactions**First of all I am adding a new directory *Helpers* to the */src/PP\_Currency* ones. ***/src  
    / PP\_Currency  
    …  
    /Helpers***

***ToolsHelper.php  
 UrlHelper.php  
 …***

The Helpers directory will have two helpers files:  
  
 *ToolsHelper.php* – to solve small and abstract tasks regarding data treatment.   
 *UrlHelper.php* – to solve tasks regarding treatment and preparation request URLs. /src/PP\_Currency/Helpers/UrlHelper.php

namespace PP\_Currency\Helpers;

use PP\_Currency\Helpers\ToolsHelper;

class UrlHelper {

private $app;

protected $request;

protected $session;

protected $config;

public function \_\_construct(Application $app)

{

$this->app = $app;

$this->request = $app['request'];

$this->session = $app['session'];

$this->config = $app['config'];

}

/\*

\* The function is replacing URLs placeholders kind of "{CURRENCY\_CODE}"

\* with real values.

\*

\* @param $url string - request url with placeholders

\* @param $placeHolderCode string - a placeholder in braces like {CURRENCY\_CODE} to be replaced with real value

\* @param $newValue string - real data to replace the placeholder

\* @returns string URL

\*/

public static function replaceParams($url, $placeHolderCode, $newValue)

{

$newUrl = str\_replace($placeHolderCode, $newValue, $url);

return $newUrl;

}  
}

/src/PP\_Currency/Helpers/ToolsHelper.php

namespace PP\_Currency\Helpers;

class ToolsHelper

{

private $app;

protected $request;

protected $session;

protected $config;

public function \_\_construct(Application $app)

{

$this->app = $app;

$this->request = $app['request'];

$this->session = $app['session'];

$this->config = $app['config'];

}

/\*

\* The static method gets associative array contains currency rates and returns an associative array with currency codes only.

\*

\* array (size=3)

\* 'base' => string 'BGN' (length=3)

\* 'date' => string '2016-08-08' (length=10)

\* 'rates' =>

\* array (size=31)

\* 'AUD' => float 0.74241

\* 'BRL' => float 1.8018

\* 'CAD' => float 0.74512

\* 'CHF' => float 0.55645

\* ...

\*

\* @param $latestRatesByBase array - associative array contains currency rates

\* @returns associative array with currency codes only:

\* array (size=2)

\* 'AUD' => 'AUD'

\* 'BRL' => 'BRL'

\*/

public static function getCurrencyCodes ($latestRatesByBase)

{

if(!isset($latestRatesByBase['rates']) || !preg\_match('/^[A-Z]{3}$/', strtoupper(key($latestRatesByBase['rates'])))) {

return $latestRatesByBase;

}

$codesArray = array();

foreach($latestRatesByBase['rates'] as $key => $val) {

$codesArray[$key] = $key;

}

return $codesArray;

}

/\*

\* This static method receives by referrence an associative array and adds to it a new element.

\*

\* @param array &$currArray

\* @param array $newElement

\* returns by referrence a sorted array with the new element

\*/

public static function addAssocElement(&$currArray, $newElement = array('key' => "", 'value' => ""))

{

$currArray[$newElement['key']] = $newElement['value'];

asort($currArray);

}

}

1. **Showing the form in the Application View**To show our new generated Form in the Application I need to update the Views files.

/src/PP\_Currency/Views/index.html.twig  
  
<!doctype html>

<!--[if lt IE 7]> <html class="no-js lt-ie9 lt-ie8 lt-ie7" lang="en"> <![endif]-->

<!--[if IE 7]> <html class="no-js lt-ie9 lt-ie8" lang="en"> <![endif]-->

<!--[if IE 8]> <html class="no-js lt-ie9" lang="en"> <![endif]-->

<!--[if gt IE 8]><!--> <html class="no-js" lang="en"> <!--<![endif]-->

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<title>PP Currency :: index</title>

<meta name="description" content="">

<meta name="author" content="">

<link rel="stylesheet" href="{{ app.request.getSchemeAndHttpHost }}/assets/css/style.css">

</head>

<body>

{% for errors in app.session.getFlashBag.get('alerts') %}

<div data-alert class="alert-box {{ errors|raw }}" style="color:red">

<ul>

{% for key, error in errors %}

<li>{{ error|raw }}</li>

{% endfor %}

</ul>

</div>

{% endfor %}

{% if resultRates|length %}

{{ dump(resultRates) }}

{% endif %}

{{ form\_start(form, {'name' : 'currencyForm', 'method' : 'post'}) }}

{{ form\_widget(form, { 'attr': {'class': 'currency\_form'}}) }}

<div>

{{ form\_row(form.amount, {'label': 'Currency I have: ', 'label\_attr': {'class': 'label'}}) }}

{{ form\_widget(form.amount) }}

</div>

<div>

{{ form\_row(form.currencyFrom, {'label': 'Currency I have: ', 'label\_attr': {'class': 'label'}}) }}

{{ form\_widget(form.currencyFrom) }}

</div>

<div>

{{ form\_row(form.currencyTo, {'label': 'Currency I want: ', 'label\_attr': {'class': 'label'}}) }}

{{ form\_widget(form.currencyTo) }}

</div>

<div>

{{ form\_widget(form.convert, { 'attr': {'class': 'button'}}) }}

</div>

{{ form\_rest(form) }}

{{ form\_end(form) }}

</body>

</html>To show the flash session alerts I am using the twig looping:  
 …  
 {% for errors in app.session.getFlashBag.get('alerts') %}

<div data-alert class="alert-box {{ errors|raw }}" style="color:red">

<ul>

{% for key, error in errors %}

<li>{{ error|raw }}</li>

{% endfor %}

</ul>

</div>

{% endfor %}  
 …

To show the testing result:  
 …  
 {% if resultRates|length %}

{{ dump(resultRates) }}

{% endif %}

…  
  
To type the open Form tag with attributes:  
 …  
 {{ form\_start(form, {'name' : 'currencyForm', 'method' : 'post'}) }}

…  
  
To set a common css class for the Form:  
 …  
 {{ form\_widget(form, { 'attr': {'class': 'currency\_form'}}) }}

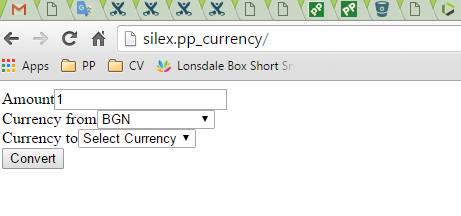
…

To type a form element’s label:  
 …  
 {{ form\_row(form.amount,   
 {'label': 'Currency I have: ', 'label\_attr': {'class': 'label'}}) }}  
 …

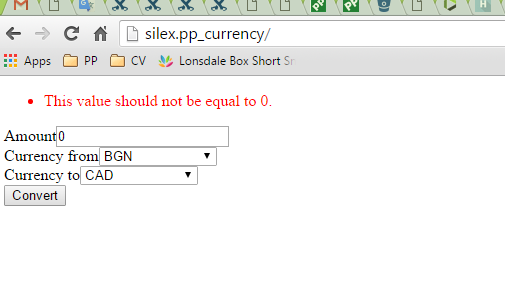
To type a form element wrapped with *div* tag:  
 …  
 {{ form\_widget(form.amount) }}

…  
  
To type the Form end tag:  
 …  
 {{ form\_end(form) }}

…

1. **Layout and check the Application working**  
     
   So, now it’s time to open the application in the browser to check what we have got at the moment: <http://silex.pp_currency/>   
     
   After opening it we can see the new form populated with some default values.   
     
   

Now I akm going to check how the Form is working.  
I am typing invalid values – letters or “0”. Press “Convert” to submit the form data.

  
As far as I expected the validation is working. I can see a red colored message  
“This value should not be equal to 0”.

**Let’s finish the CONVERTER – providing a correct result**To be continued……...

**Using Foundation css framework to change the Application style**  
  
To be continued……...