SF Initial guide

BUILDING EXAMPLE APPLICATION

MILJAN PANTIC

Table of Contents

1. Introduction

1.1. Purpose

Purpose of this guide is to show basic usage of the Simple Framework (SF in the text below).

1.2. Environment

SF is tested in the next environments (servers):

OS	Server	PHP	MySQL	Notes
Windows 7	Apache 2.4	5.6.14	10.1.8-MariaDB	XAMPP installation
Ubuntu 14.04	Apache 2.4.7	PHP 5.5.9	5.5.46-0ubuntu0.14.04.2	

Note:

1. Apache must have mod_rewrite enabled.
2. All test servers have *default* options from the installation
3. MySql server is installed on the local machine, along with an Apache server

SF automatically detects some parameters of the environment, and, since it is not yet tested enough (except on the listed environments) there might be some problems with different environments. In that case, contact developer of the SF.

1.3. SF Features

This version of the Simple Framework is very similar to what final version should be, but some features are yet to be developed. List of these features can be found in the table below:

Feature	Description
Loading of a JS Libraries	JS Libraries now can be loaded within the index.tpl file. Enable component, dependent JS library loading.
Add more features to the console	Console now has a lot of features, but additional ones could be added. E.G. Making a specific database connection permanent. Extending of the console functions is very easy thanks to the console architecture which allows easy adding of the new operations.
Loading configuration from the database	Currently, whole configuration is loaded from the PHP files. Also, console is able to work with this PHP files. This is extremely suitable for smaller website, hosted on servers with no database connection available. For the larger sites however, configuration (which contains things like users) should be kept in the dedicated database. This is also easy to achieve thanks to the fact that only database adapter for configuration manipulation should be created.

Session security and OWASP Top 10	Session should be secured and OWASP Top 10 threats should be addressed.
Import and export of the components	This should be done so users can have their components migrated between SF applications. This is left out of this version because of JS and PHP library dependency issues on a specific component.
Cache logic	Caching logic should be available within the logic components. Also, caching on the framework level should be enabled.
Help in the console	Currently, console sf_assist does not have any help text for the developers. This should be added.
APIs	APIs for forms, mails, notifications should be added.
Unit tests	Since SF is taking advantage of the dependency injection, unit test should not be hard to enable. At least for the logic components.
Composer integration	There are some dependency issues mentioned in this table. These might be resolved by using Composer.
SQL File manipulation	SQL Files are handles manually at the moment. This should be changed.

2. Example application

Example application will work with news articles. Users should be able to read articles and logged users should be able to add, remove and update articles.

2.1. Before you start

1. Make sure that you have some of the environments listed in the chapter #1.2.
2. Then, enable mod_rewrite on your Apache server (in case that it is not already enabled).

   This can be done:

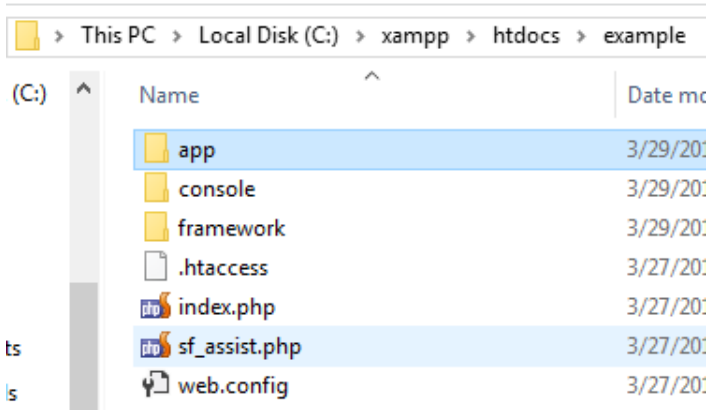
   On Windows (XAMPP Installation):
   a. Access httpd.conf
   b. Find the section which points to your webroot directory (usually: <Directory "C:/xampp/htdocs">)
   c. Add or update: AllowOverride All

   On Linux installation:

   a. Open terminal
   b. Enter: a2enmod rewrite

3. SF Deployment:

   a. Obtain SF from: https://github.com/Miljankg/simplef/tree/master/src
   b. Create a directory within your Webroot dir. (e.g. *C:\xampp\htdocs* and create directory example. Full path of your site is: *C:\xampp\htdocs\example*)
   c. Place SF into your directory that you have just created. You should have next structure:



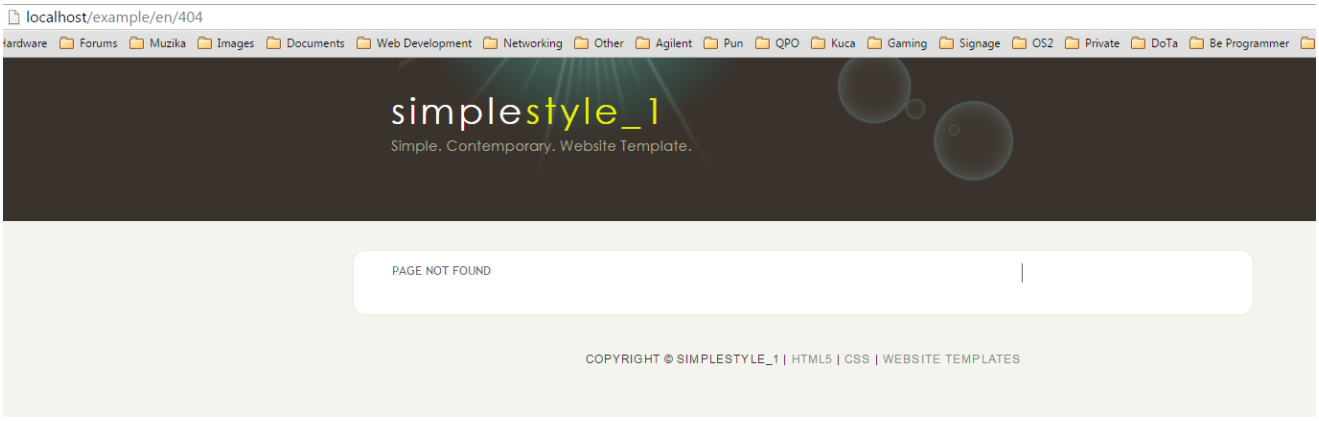
   d. Make sure that next directories have proper privileges (for the user under which Apache is executed):

      app/templates_c
      app/cache
      framework/log

      and next directories should have write permissions for user that runs SF Assist:

      framework/config
      app/

   e. Then, open your browser and enter: http://YUOUR_SERVER_NAME:PORT/example (e.g. http://localhost/example).
   f. You should see:



In case of any problems, you can check framework/log/sf.log file.

Now, you can move on to the implementation of the application.

Note: In case that you copy some code from this document, don't forget to check the quotes and replace them with proper PHP quotes. Apart from that, please note that some of the CSS and HTML is already added to the INDEX.TPL and INDEX.CSS in order to make it easier for you to follow important parts of this guide. This way, you are skipping simple copy paste step from downloaded free template used for this guide, to index.tpl and css files (see end of the document, for the credits).

2.2. SF Structure

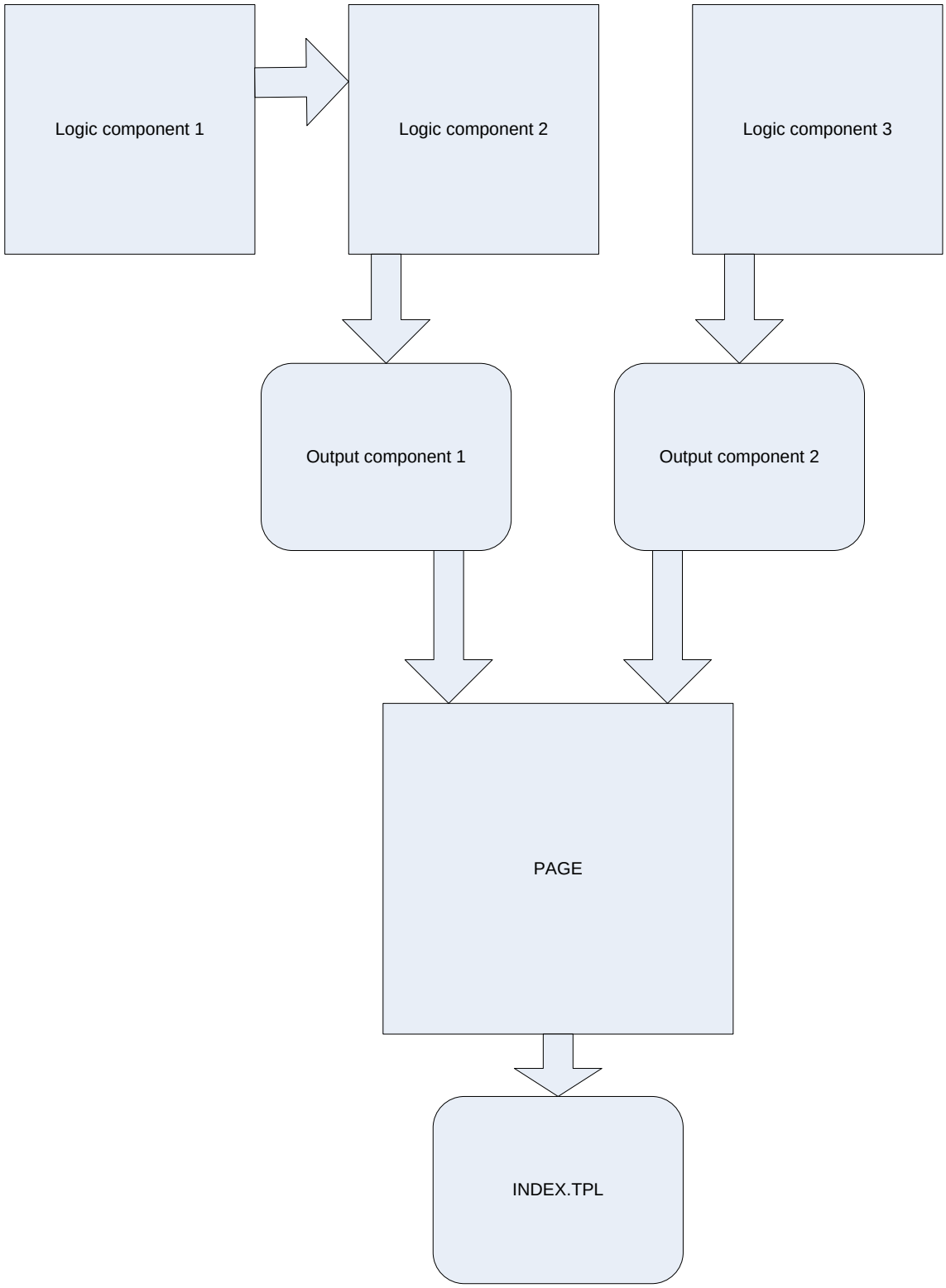
Before starting to work with SF, some of its implementation details should be known. There is couple of major parts of the SF:

1. Basic components:
    a. **Logic components:**
    These are basic components and their purpose is to provide APIs for logic operations within the application based on SF. For example, if you have an application which works with ads, you will likely have logic component which does all of the required work with ads. These components can be dependent on each-other and can be listed as a dependency for the output components. Also, these components are only ones that have access to the database objects.

    b. **Output components:**
    These components are used to be embedded within a page. Their purpose is to provide content for the specific modules on a page. If we have a website that works with ads, we will likely have an output component for the latest ads displaying. This component will probably use some logic component to get ad data, then it will process the data and embed it into its own template. Afterwards, this template is fetched and embedded into page template, which is displayed to the user. These components can be listed as dependency for a page, and they can be dependent on logic components. These components have their own tpl, js and css files, as well as php files.

    c. **Pages:**
    These are not actual components, but more a templates. They can contain placeholders for the output components they are dependent on and those placeholders will be replaced with the html content gathered from the fetched output components. Then also contain additional frontend code required for that page. Only one page can be loaded at a time. Each page is embedded into index.tpl file.

Diagram which should help on understanding the components hierarchy within the SF can be found below:


Logic component 1
Logic component 2
Logic component 3
Output component 1
Output component 2
PAGE
INDEX.TPL

2. Framework execution:
    a. **Initialization**:
    When a request is received, index.php file is invoked, and .htaccess (or a web.config file) does the job of transforming the URL. After that, config_main.php file is loaded first.

    b. **Boot up:**
    After main configuration is loaded, an SF instance is made, and then, an execute function is invoked. This function loads additional configuration files, sets up exception handling, logging, database connections, parses URL parameters and boots up required components.

    c. **Component loading:**
    According to the parsed URL parameters, SF determines which page should be loaded. Then it reads from the loaded configuration which output components should be loaded. Then, before loading output components, SF loads all required logic components. Each component can be loaded only once, and then, the reference is re-used. After loading logic components, for each output component an execute function is called, which executes all action related to one output component, and then it fetches a template which is bonded to one output component (each output component has its own template). During *execute()* function of the Output Component, developer can assign anything he needs into Output Component's template and then, assigned data can be further processed within the template using Smarty functions. Also, reference of each Logic component, which is listed as dependency for specific Output Component, can be retrieved within the *execute()* function of the Output Component and then, available functions from the Logic component can be used for various operations.

    d. **Page loading:**
    After that, each fetched output component is placed using template engine into placeholders in the page template. Then, page template is fetched. Fetched page template is embedded into intex.tpl and whole content is returned to the user.

2.3. SF Assist usage

SF Assist is an addition to the SF, and it provides a developer to adjust framework configuration without getting involved with configuration files itself.

Currently, only SF Assist only works with PHP files. This allows developers to have smaller scale application without a database server required. Developers are able to even add roles and users and similar. Database option will be added, and this will be done easily, since only adapter for the database configuration operations should be added.

At this point SF Assist is missing help options, so you need to rely on the text down below.

Usage:

1. Make sure that your PHP is accessible from the environment variables. Otherwise you can use full path to the php executable. **Make sure that your run SF Assist commands with sudo on**

Linux platform. This is due to usage of mkdir php function and it will be improved in the future revisions of the SF Framework.

2. Open command line and navigate to the SF directory (where you copied SF files).
3. Enter: php sf_assist.php <action_object> <operation> (action_object and operation will be replaced in the text below, where usage of the sf_assist is required). Example:

php sf_assist.php db_connection add

4. Script will then ask for any additional parameter (option where you can enter all the parameters directly in the command before execution will be added later).

2.4. Required application elements

For this application we will have next elements and their dependencies:

1. Home page
    a. Output component => Latest news main
        i. Logic component => news
2. Page for creating / removing / updating news
    a. Output component => add_news
        i. Logic component => news
    b. Output component => Edit / delete news
        i. Logic component => news
3. Common components (component that will not be dependent on a specific page, but it will be displayed everywhere, on every page):
    a. Output component => Latest news (smaller version of the latest_news_main)
        i. Logic component => news
    b. Output component => Menu

Note: Some of components are repeated. Those will be created only once, but they are enlisted multiple times so reader can have visual insight on the dependencies.

2.5. Creating database

Currently there is no special commands for handling SQL files. Because of that, in order to create database required for our example application, first create database “example” in your mysql server and then execute SQL file:

app/sql/20160329_040300_example.sql

After that, SF should be configured to connect to the database. That can be done using sf assist and you can add, remove and update database connection. In order to add connection, execute next command:

php sf_assist.php db_connection add

You will be asked for next parameters:

1. **Connection name:** enter: *example* (this will be used to get an instance of the connection within the application. When you use SF assist to create db connection, a PHP constant is also created, in this case DB_EXAMPLE, for easy referencing to the instance name of the connection.)
2. **DB_TYPE:** enter: *mysql*. (SF supports multiple connections as well as multiple connection types. Currently, oracle and mysql, but for this application we will use mysql.)
3. **DB_USER:** database username
4. **DB_PASSWORD:** database password
5. **DB_HOST:** database host (localhost)
6. **DB_NAME:** database name (in this case, example)

If connection is created successfully, refresh the page and if there is no "Database connection error" text, connection is working properly.

2.6. Creating logic components

From the above list we can see that we need one component "news". In order to create it, do next:

1. Execute sf assist command: php sf_assist.php logic_component add
    a. SF Assist will ask for component name, enter: *news*
    b. Then, you will be asked to enter dependencies, there is none in this case, just press enter.
    c. Message that operation was successful should appear, and component "news" should be found on under *app/components/logic/news*

2.7. Creating output components

We can see that we need next output components:

1. latest_news_main
2. add_news
3. edit_delete_news
4. latest_news
5. menu

In order to add output component, execute:

php sf_assist.php output_component add

You will be asked for next parameters:

1. Component name => e.g. latest_news_main
2. Component dependencies => enter *news* (logic component that we created earlier) for all components, except for *menu*, for *menu* leave it empty (just press Enter)
3. Enable JS => will component use Javascript (yes for all components)
4. Enable CSS => will component use CSS(yes for all components)

Except adding, you can remove output component, enable_js or disable_js (css as well).

Now, we should add two common components, to common components in the configuration. That can be done by executing next command:

php sf_assist.php output_component add_to_common

When asked about component name, enter: *menu*, and then in for the second component enter *latest_news*. Those components will be loaded for every page. There is an option to add exceptions to this (name of the pages where loading of some common component will be skipped) but this option is not yet available within the SF Assist.

2.8. Creating pages

Two pages will be created for this application:

1. main_page
2. news_edit_page

In order to create page, execute next sf_assist command:

php sf_assist.php page add

You will be asked for next parameters:

1. Page name: e.g. main_page
2. Page dependencies:
    a. For page main_page, enter latest_news_main
    b. For page news_edit_page, enter add_news,edit_delete_news

Then, we need to set default page when application is accessed without specific page in the url. Do that by executing next command:

php sf_assist.php page set_empty_page_index

When asked for page name, enter: main_page.

2.9. Enable login

We want to make page "news_edit_page" accessible only to logged users. This can be easily achieved using SF Assist.

First we need to add role that we want to access the page. In order to do this, execute next command:

php sf_assist.php roles add

When asked, for the role name enter: *authors*

When this is done, it would be good to add users to this role. This can be achieved using:

php sf_assist.php users add

When asked, enter username, password and role *authors*, which we created previously.

Now, we need to map the role to the page. SF will then automatically prevent access and it will redirect user to the login page, if user does not have specified role/s (page can have multiple roles assigned). This version of SF comes with already predefined Login page and login component. You can create your own login output component, but you need to say to the SF to map that component to the existing login page. This can be also achieved using SF Assist:

php sf_assist.php page add_dependency

When asked, enter *login* as page name, and your login output component name for pages dependencies. Apart from this, you need to open login page template (app\templates\pages\login\login.tpl) and add placeholder for your output component and to remove anything that you do not need anymore.

Old login component dependency can be removed from the login page using:

php sf_assist.php page remove_dependency

And when asked for the component name, enter: *login*

Note: As already mentioned, this version of SF has its own login page, so there is no need for creating new one.

Now, let's map the role to the page:

php sf_assist.php page add_role

And when asked, for page name, enter *news_edit_page*, and for role name enter *authors.*

Try to access:

http://YOUR_SERVER_IP_PORT/example/news_edit_page

You should be redirected to the login page, and you should be able to login to with the user credentials added previously.

3. Coding

Now, we need to add code to our application.

3.1. Logic components coding

First, let's code the logic component "news". Open your IDE and navigate to the app\components\logic\news\news.php.

You will see that there is already class "News" which extends "LogicComponent" and has empty implementation of function Init(). Next things are important:

1. Logic components are loaded first.
2. They represent APIs to be used by other logic or output components.
3. As already mentioned, they can be listed as a dependency both for other logic or output components.
4. When logic components are loaded, their Init function is called before anything else. This is important, because here, developer can initialize anything that needs to be loaded before some other component calls some of the functions within the logic component. For example, since SF can have multiple database connections, and your logic component needs only one, init function can contain next code:

```
$this->db = $this->dbFactory->GetDbInstance(DB_EXAMPLE);
```

5. In the above example, $this->db is class property specified by us, and DB_EXAMPLE is the PHP constant, created by SF Assist when we created database connection. Constants are used for easier reference to database connection names, page and component names etc.
6. After initialization, logic component will serve as API for other components which are dependent on it.

In our case, we have one logic component, called "news" and will add next code to it:

```php
<?php

namespace Components\Logic;

use Framework\Core\FrameworkClasses\Components\LogicComponent;
use Framework\Core\Database\DB;

class News extends LogicComponent
{
    /** @var DB */
    private $db = null;

    public function init()
    {
        $this->db = $this->dbFactory->GetDbInstance(DB_EXAMPLE);
    }

    public function getAllNews($limit = 10)
    {
        $query = "SELECT * FROM news LIMIT 0, $limit";

        return $this->db->ExecuteTableQuery($query);
    }
}
```

3.2. Output Component coding

After this, we need to setup some of the output components. For this, we will setup “menu” output component and “latest_news_main” component.

For the menu component, open app\components\output\menu\menu.php. In there, you will se existing class “Menu” which extends “OutputComponent” class. OutputComponent class has one abstract method, execute(), which needs to be implemented in its child classes. Execute method does not return anything and its purpose is to do any kind of processing that this is needed for the output component, and then to assign data to output component template.

So, for the menu component, add next code to previously opened menu.php file:

```php
<?php

namespace Components\Output;

use Framework\Core\FrameworkClasses\Components\OutputComponent;

class Menu extends OutputComponent
{
```

```
    protected function execute()
    {
        $skipPages = array('error_page', 'maintenance', 'login', '404');

        $pages = $this->sf->Config()->get('pages');
        $mainUrl = $this->sf->Config()->get('main_url');

        $currentPage = $this->sf->Url()->getCurrentPageName();

        $menu = array();

        foreach ($pages as $pageName => $pageConfig)
        {
            if (in_array($pageName, $skipPages))
                continue;

            $translation = "";

            try
            {
                $translation = $this->sf->LangPages()->get('page_' . $pageName);
            }
            catch (\Exception $ex)
            {
                $translation = $pageName;
            }

            $menuItem = array();
            $menuItem['page_translate'] = $translation;
            $menuItem['page_url'] = $mainUrl . $pageName;
            $menuItem['selected'] = ($currentPage == $pageName) ? 'selected' : '';

            $menu[] = $menuItem;
        }

        $this->tplEngine->assign('menuItems', $menu);
    }
}
```

Now, we can see that in the PHP code, we assign array of menu items into the template. In order for this array to be properly processed, we need to add code to the template file. For the menu component, open app\templates\out_components\menu\menu.tpl and add next code to it:

```
<ul id="menu">
    {foreach from=$menuItems item=menuItem}
    <li class="{$menuItem.selected}"><a
href="{$menuItem.page_url}">{$menuItem.page_translate}</a></li>
    {/foreach}
</ul>
```

Apart from the tpl file, you can add code to the output component's css and js files, but we don't need that in this example.

Now, we should do the same for the "latest_news_main" output component. Open app\components\output\latest_news_main\latest_news_main.php. Add next code to it:

```php
<?php

namespace Components\Output;

use Framework\Core\FrameworkClasses\Components\OutputComponent;
use Components\Logic\News;

class LatestNewsMain extends OutputComponent
{
    protected function execute()
    {
        /** @var $lcNews News */
        $lcNews = $this->getLogicComponent(LC_NEWS);

        $news = $lcNews->getAllNews($this->config->get('latest_news_count'));

        $this->tplEngine->assign('news', $news);
    }
}
```

We can see that “latest_news_main” output component uses its own configuration. So, we need to add ‘latest_news_count’ value to the component configuration. In order to this, open app\components\output\latest_news_main\config\latest_news_main_config.php file and add next code to it:

```php
<?php

$config['latest_news_count'] = 10;
```

After this, we need to add code to the template file of the output component. Open app\templates\out_component\lates_news_main\latest_news_main.tpl and add next code to it:

```
{foreach from=$news item=article}

    <h2>{$article.news_title}</h2><br/>
    {$article.news_text}<br/>
    {$article.news_timestamp}<br/><br/><hr/><br/>

{/foreach}
```

3.3. Pages coding

When we did this, we have coded our components required for the main_page. Now, we should tell main_page where to position its output components. This is done in the page’s template (we have already configured page main_page, to load latest_news_main output component, in the previous chapters). Open app\templates\pages\main_page\main_page.tpl and add next to it:

```
{$oc_latest_news_main}
```

For this example, no other html code is required for the main_page, but you can add anything you want. This time, only placeholder for the latest_news_main coomponent is required. In order to add the placeholder you need to add prefix “oc_” to the name of the output component.

3.4. Common output components coding

Now, we need to set “menu” component. This component will be shown on all of our pages. Therefore, it is not good to put it in the page template, since we will need to do it for every page we have and for every component we want to add later. Because of that, the best way is to set “menu” component in the index.tpl file because index.tpl is loaded always.

In order to do this open app\templates\index\index.tpl, and change the code as shown below (replace content of the “menubar” div with {$oc_menu}):

```
<div id="header">
    <div id="logo">
        <div id="logo_text">
            <!-- class="logo_colour", allows you to change the colour of the text -->
            <h1><a href="index.html">simple<span
class="logo_colour">style_1</span></a></h1>
            <h2>Simple. Contemporary. Website Template.</h2>
        </div>
    </div>
    <div id="menubar">
        {$oc_menu}
    </div>
</div>
```

Now, you can load main page, and you should see next:

If you click on the news_edit_page, then you will be redirected to the login. On the image above, you can see parts of the displayed content, marked so you can see what is part of the output component is and what is part of the page. Everything else, which is not marked is part of the index.tpl file and it is common for all pages.

Now, you can code rest of the components for the news_edit_page.

If you have any problems finishing these steps, you can download application with all of the mentioned steps completed here:

https://github.com/Miljankg/simplef/blob/master/doc/examples/example_app/example.zip

4. Credits

- **Template used in this example application:**
  http://www.html5webtemplates.co.uk/templates/simplestyle_1/index.html