Simple Framework Idea and Concepts

Miljan Pantić

Date: 20.12.2015

Table of Contents

1. Introduction

Simple Framework should be an easy to use, modular framework. This framework, should enable developers to develop their part of the application without exactly knowing how other parts works, apart from interfaces (APIs) which they need to use.

Except this, easy debugging should be available. Modularity of this framework will make possible easy code fixes, replacements of the whole modules and easy handling of problematic situations, like new members in the development team and similar.

Except this features, this framework will have possibility of exporting / importing modules between different projects. Except pure export import, there will be an option which will enable users to include most often used modules into Simple Framework as a product, which means, that each "clean" copy of their simple framework will have their favorite modules already included.

2. Concept

2.1. Centralized environment

This is something which must be considered as a must, when any framework is written. Centralized starting point, configuration, logging and similar features, will allow easy to use, and easy to control and eventually debug, environment.

But, simple framework will also allow developers to write their configuration for each module, which means that, configuration related only to their module, should stay beside that module, and it should not interfere with global configuration. This way, global configuration is kept clear, and module-related configuration is smaller and easy to find. Framework API will provide easy to identify from where configuration values are pulled, from global configuration or from module-related configuration.

2.2. Framework APIs

Framework APIs will provide basic functionalities to the developer, either through the global variables or through the module parameters (yet to be decided). Those functionalities are logging, query execution, configuration values retrieval and similar. They will be loaded at the beginning of the framework boot process.

Since the general idea (see chapter #1) says that there will be modules that can be included into a Simple Framework as a product, it is important to distinguish what is the difference between basic Framework APIs and those modules.

Modules are APIs and functionalities, which are developed by framework user (developer) and they are meant to be just for one or more web applications, but might not be applicable to all of the web application (although there might be modules that will be very useful for any web developer). This means that these functionalities are meant to be adapted for usage in the specific scenarios, and framework itself can easily function without them.

On the other side, Framework APIs provides developer with basic functionalities, which are considered most usable during development of some project. Although, those functionalities can be excluded from some project (e.g. Database APIs for project that does not have database), they are still considered basic and needed by developers in most cases.

2.3. Logical units

This part is related to the units which will be developed by the Framework user. There two major parts, that needs to be considered:

1. OutputComponent
2. LogicalComponent

1. OutputComponent

This component can be considered as a place where you can output anything you want for your web application. This components have access to the templates (read more about Templates in the #2.4 chapter below). Developer can write any kind of logic in the OutputComponent and he can either output this using echo function (it will be available in the module API) or it can assign any values into the corresponding template. In case when developer assign values to the template, template engine will allow him to manipulate those values within the template and this template will be automatically loaded by the framework. Multiple output components can be loaded for one page to be generated.

For example, if you have a page, which displays list of ads, this page can also have parts for displaying newest ads, most visited ads, sponsored ads and similar. Each one of the mentioned page elements, can be considered and developed as a separate OutputComponent with its own template. How these templates are processed and merged to form a page, read in the #2.4 chapter bellow.

It is import to mention, that Output Components are completely independent of each - other, but they can be dependent on the LogicalComponents, which is described below.

Except this, OutputComponents can be configured to run before other OutputComponents in order to achieve pre-output-rendering checks (access rights and similar).

2. LogicalComponent

This components can be considered as a logical interfaces that can be used by an OutputComponent. Main difference between logical and output components is the fact that only LogicalComponents will have an access to the database. The reason for this organization is best explainable in the ads example.

If you have one page that displays list of ads, newest ads and most popular ads for example, and each of these is a separate OutputComponent, each of these components needs to retrieve a list of ads using query (some will retrieve all ads, some only newest, but each one needs the query from the database). This when advantage of having LogicalComponent is really showing up. Instead of executing query in each of the OutputComponent, you can get all required data in the logical component, fetch it and create needed statistics (newest ads, most popular ads and similar).

After that, each OutputComponent can have few logical components specified as its own dependencies, and then, it can use already gathered data for its own purposes.

This centralized approach will allow one more advantage, and that is data caching. That means that in case of large amount of data, LogicalComponent can cache data, and then OutputComponents will have fast access to the data.

It is important to know that logical components will actually represent APIs, and except the fact that OutputComponent can have LogicalComponents as its dependencies, some LogicalComponent can also have other LogicalComponents as its own dependencies.

2.4. Templates

Template engine used in the Simple Framework is Smarty.

Each OutputComponent has its own template. This template contains only html/css/js code required for that component. Except this, OutputComponent template will contain assigned PHP values, from the OutputComponent PHP Code.

NOTE	Not every component needs its own template. For example, if component is called asynchronous and it needs to output only some JSON data, it does not need it's template to be loaded, and developer will be able to do this, buy calling Simple Framework function which will output desired content instead of fetching the template. In this case, there is no need for executing more than one OutputComponent, and because of that, after outputting desired content which is not a template, execution of the application will stop.

At the end of the execution process, each of OutputComponent templates is parsed and assigned to the template of the Page.

Page is another term which is used in the Simple Framework. Page is actually Route, which means, that some URL is mapped to load specific Page.

Page does not have any PHP code, except configuration, which has two mains parts:

1. Template path
2. OutputComponents to load

When request comes, URL mapping is read from the configuration and proper OutputComponents are loaded and their templates are parsed. After that, those templates which are fetched from the OutputComponents, are placed in the template of the loaded Page, which contains placeholders for each loaded template from OutputComponents. This is enabled because of Smarty usage.

Mentioned placeholders can be wrapped by HTML/CSS in the template of a Page in order to control their positioning and style within the page.

After template of a Page is loaded it is assigned to the main template. This template contains html head section, and page placeholder. This placeholder will be filled with parsed template of a page.

Note that this, main template, can also contain placeholder for some OutputComponents (menu can be one of them for example). This is especially useful when some OutputComponent is loaded on each page.

In order to achieve this, configuration will contain an option to specify which component will show up on every page (you will also be able to specify which pages to skip in case that you need something like that).

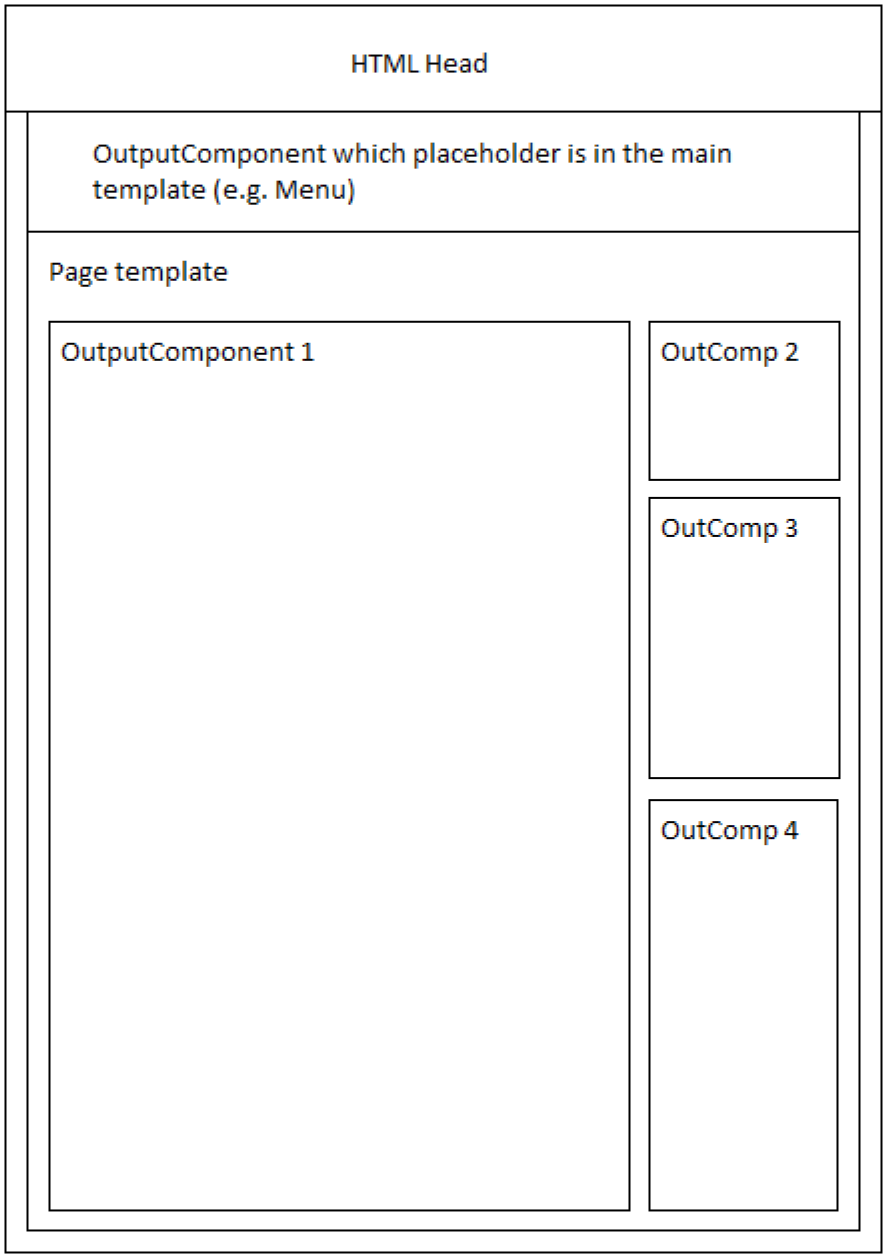
HTML Head
OutputComponent which placeholder is in the main template (e.g. Menu)
Page template
OutputComponent 1
OutComp 2
OutComp 3
OutComp 4

Image 1 – Simple Framework parsed template stack

2.5. Execution process

Execution process will first boot main configuration, loaded from the PHP file. This configuration will contain only options that Simple Framework requires in order to be started. After that, next steps will be performed:

1. Main instance of the Core class will be made. This instance will be used for further startup process.

2. When this is done, Framework APIs will be loaded (logging and other tools and similar).

3. In case that database connections are configured, this connections will be established, and in case of an error, execution will stop.

4. Depending on what is set in the main configuration file, rest of global configuration can be loaded either from the PHP file, or from Database (more in the chapter 2.6).

5. After everything is started up, all of the required component will be loaded. URL from the request is parsed, and compared with the configured pages in the global configuration. In case when proper page is not found, execution will stop, otherwise, required OutputComponents are loaded.

6. First, Framework needs to read, which components and their dependencies to load. This is achieved by reading previously mentioned configuration (each OutputComponent has its dependencies listed).

7. Dependencies (LogicComponents) will be loaded first, along with their own configuration files and their Init methods will be called.

8. After that, each of configured OutputComponents will be loaded, and then their Execute methods will be called.

9. When all components are loaded and executed, templates will be processed and main output will be generated and echoed out.

2.6. “Utility” parts of the framework

These are feature that will probably be developed, but they are considered as luxurious, and they are intended for improving the developer’s framework experience.

Loading configuration from DB instead of PHP file.

This will allow development of a GUI for manipulating framework preferences. This should provide user easy to use interface for controlling the framework and its components.

Console for import / export and creation / removing an object. This console should allow developer to easily export or import Components and their dependencies. These functionalities can be also added to the GUI for managing framework preferences.

Automatic form generator on database tables

This should be the biggest addition to the framework, and it can be even considered as a completely separate application.

This should allow any developer to automatically generate forms with CRUD functionalities over his database tables. This forms should also be configurable and validation will also be included.

This form generator should allow developer to code its own extension of some form in case when basic 1 on 1 form is not enough.

It can be used to provide GUI for the Framework preferences.

All of these functionalities will be provided depending on the time remaining for the development.

PHP 7

PHP 7 has just been released. Its main advantages is speed, scalar type hints, return types and similar. Since PHP 7 has been just released, it would be good for Simple Framework to be fully tested before it is rewritten into PHP 7.