Twig

13/2/2020

Overview

In this project, we learn to work with Twig. Twig is a modern template engine for PHP

- **Fast**: Twig *compiles* templates down to plain optimized PHP code. The overhead compared to regular PHP code was reduced to the very minimum.
- **Secure**: Twig has a *sandbox* mode to evaluate untrusted template code. This allows Twig to be used as a template language for applications where users may modify the template design.
- **Flexible**: Twig is powered by a flexible *lexer* and *parser*. This allows the developer to define its own custom tags and filters, and create its own DSL.

In this project, important aspects of TWIG are explored such as blocks, macros etc.
When it comes to template engines in PHP, many people will tell you that PHP itself is a template engine. But even if PHP started its life as a template language, it did not evolve like the one in recent years. As a matter of fact, it doesn't support many features modern template engines should have nowadays:

Concise: The PHP language is verbose and becomes ridiculously verbose when it comes to output escaping:

```
<?php echo $var ?>
<?php echo htmlspecialchars($var, ENT_QUOTES, 'UTF-8') ?>
```

In comparison, Twig has a very concise syntax, which make templates more readable:

```
{{ var }}
{{ var|escape }}
{{ var|e }}       {# shortcut to escape a variable #}
```

-

Template oriented syntax: Twig has shortcuts for common patterns, like having a default text displayed when you iterate over an empty array:

```
{% for user in users %}
    * {{ user.name }}
{% else %}
```

```
    No users have been found.
{% endfor %}
```

Full Featured: Twig supports everything you need to build powerful templates with ease: multiple inheritance, blocks, automatic output-escaping, and much more:

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

{% block content %}
    Content of the page...
{% endblock %}
```

- **Easy to learn**: The syntax is easy to learn and has been optimized to allow web designers to get their job done fast without getting in their way.

Of course, PHP is also the language for which you can find the more template engine projects. But most of them do not embrace web development best practices yet:

- **Extensibility**: Twig is flexible enough for all your needs, even the most complex ones. Thanks to an open architecture, you can implement your own language constructs (tags, filters, functions, and even operators) to create your very own DSL.
- **Unit tested**: Twig is fully unit-tested. The library is stable and ready to be used in large projects.
- **Documented**: Twig is fully documented, with a dedicated online book, and of course a full API documentation.
- **Secure**: When it comes to security, Twig has some unique features:

Automatic output escaping: To be on the safe side, you can enable automatic output escaping globally or for a block of code:

```
{% autoescape "html" %}
    {{ var }}
    {{ var|raw }}     {# var won't be escaped #}
    {{ var|escape }}  {# var won't be doubled-escaped #}
{% endautoescape %}
```

Sandboxing: Twig can evaluate any template in a sandbox environment where the user has access to a limited set of tags, filters, and object methods defined by the developer. Sandboxing can be enabled globally or locally for just some templates:

```
{{ include('page.html', sandboxed = true) }}
```

- **Clean Error Messages**: Whenever you have a syntax problem within a template, Twig outputs a helpful message with the filename and the line number where the problem occurred. It eases the debugging a lot.
- **Fast**: One of the goals of Twig is to be as fast as possible. To achieve the best speed possible, Twig compiles templates down to plain optimized PHP code. The overhead compared to regular PHP code was reduced to the very minimum.

Objectives of project

- Understand that it is a template engine
- Understand TWIG syntax and put it into practice
- Develop a project that implements the PROS of using template system.
- Improve your skills in architecture, separation of responsibilities, and good practices.

Table of Contents

Project requirements

- Requirements documentation.
- List of tasks to be performed.
    - Priority of each task
    - Title and description of each of them
    - Difficulty level
    - Estimated time for each task.
- Record of incidents that were detected during the execution of the pill.

- Documentation about the git WORKFLOW you are going to use
- Documentation about the tools used in the project
- Record of lessons learned.

Risk Management

Every project has risks. These risks must be taken into account to improve the workflow of the project. Managing these risks is important for due completion of the project. Unchecked risks could not only lead to hamper of project deliverance but also a badly executed project. The risks hence associated with the project are documented as follows:

Risk	Risk level
Unfamiliarity with Twig	Medium to High
Installation of Composer	Medium to Low
Health and computer issues	High
Delivering in time	Medium
Completing all the project requirements	Medium

Task Management

The following are the tasks which needed to be accomplished for the completion of the project. The tasks have been divided according to the project specifics. Each task has been clearly defined and their priority as well as their difficulty and time needed. Difficulty level is explained on a scale of 1 to 5 (5 being the most difficult). Priority is explained on the level of 1 to 5 (again 5 the highest parameter being the most prioritized work).

Task	Priority	Description	Difficulty	Time
Read the description of the project	5	This task involves complete understanding of requirements for the project	1	30 min
Create git repo	5	Creating of git repository for project execution and delivery	1	2 mins
Create files for the project	4	Creating the necessary files, folders and subfolders for this project. This is important especially for this project of PhP unit	1	5 mins
Download composer	3	Downloading the composer which is required for phpunit installation	2	10-20 mins
Install Twig	4	Installing the Php twig	3	30 mins
Create 4 blocks in the child	3	Create 4 blocks in the child.html to be rendered	4	2-2:30 hr
Create macros for the form	1	Create a macro for the form template and import it in the child	5	30 mins
Check if variable exists or not	1	Check if news exists then put all the the contents as not exists	4	30 mins
Use include	1	Use include to include vies	3	15 mins
Documentation	1	Write the documentation and push it to github	2	2 hr

Defining this part is crucial to the development of the project. It is important to make a good analysis of the situation to organize the project in a good way.

Chronogram

This shows the time-line it took for the project to finish and also the tasks that were achieved during those timeline.

The tasks have been mentioned in the left axis

Tasks	Wed 1700 hrs	Wed 1800 hrs	Thurs 1100 hrs	Thurs 1200 hrs	Thurs 1300 hrs	Thurs 1400 hrs	Thurs 1500 hrs
Read project description	X						
Git Repo	X						
Create files	X						
Download composer		X					
Install twig		X					
Create 4 blocks in child html		X	X	X			
Create macros for the form				X	X		
Check if news exists						X	
include						X	
Documentation						X	X

Git Workflow

For this project, all commits we'll be pushed directly to the Master branch. All commits will use a descriptive message, so that the admin and other internet user can follow through the processes.
For more information go to :
https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Incidents

- Problem of iterating over arrays which was successfully resolved

Technologies used

For this project, we will use the following technologies:

- HTML
- CSS & Bootstrap
- Php
- Twig
- Composer