Symfony Framework

-Hemalatha
What is Symfony

- Symfony is an Open source web application framework for PHP5 projects.
- PHP is a general purpose scripting language designed for web development.
- The best use of PHP is in creating dynamic web pages in client side.
- Symfony is designed to optimize the development of web applications.
- It adds a new layer on top of PHP providing tools that speedup the development of complex web applications.
- Used in high demand e-business websites.
- Compatible with most of the available database servers including MySQL, Oracle, Microsoft SQL server.
Symfony History

- First released on October 2005 by founder Fabien Potencier
- When PHP5 was released, Fabien decided that the available tools had reached a mature enough stage to be integrated into a full featured framework
- He then developed Symfony code basing his work on MVC architecture, the propel object relational mapping and ruby on rails templating helpers
- After successful testing of a few projects Fabien released Symfony under Open Source licence
- Then Fabien along with his colleague authored a book with excellent documentation as he believed extensive documentation in english is important to be adopted globally
Features of Symfony

- Simple to use
- Easy to install and configure on most platforms
- It is independent of database engine
- Easily adaptable for complex class definitions
- It has a lot of inbuilt functionalities that are conventional
- The user has to configure only unconventional functions
- Compliant with most web practices and design patterns
The MVC Architecture

- Symfony is based on the classic web design pattern called the MVC pattern

- This consists of three levels
  - The model: The application logic that is the information using which the application operates
  - The view: This renders the model into a webpage that is suited for usage by the end user
  - The controller: This responds to user actions and it invokes changes in model or view as required

Source: http://en.wikipedia.org/wiki/Model-view-controller
## Symfony VS. MVC

### The Components of MVC

- **Model layer**
  - Database abstraction
  - Data access

- **View layer**
  - View
  - Template
  - Layout

- **Controller layer**
  - Front controller
  - Action

### The Components of Symfony

- **Action**
- **Template**
- **View**
- **Layout**
SymFONY WORKFLOW

Client

Internet

Front Controller

Action

Layout

View Logic

Template

Data Access

Database Abstraction

Database

Server

http://www.symfony-project.org/book/1_0/02-Exploring-Symfony-s-Code
Script example

A Flat Script

- Very Easy to write
- Very difficult to maintain.

```php
<?php
// Connecting, selecting database
$link = mysql_connect('localhost', 'myuser', 'mypassword') or die('Could not connect to database');
mysql_select_db('blog_db', $link);

// Performing SQL query
$result = mysql_query('SELECT date, title FROM post', $link) or die('Could not execute query');

<html>
<head>
    <title>List of Posts</title>
</head>
<body>
    <h1>List of Posts</h1>
    <table>
        <tr><th>Date</th><th>Title</th></tr>
        <?php
        // Printing results in HTML
        while ($row = mysql_fetch_array($result, MYSQL_ASSOC)) {
            echo "\t<tr><td>\n;
            printf("\t<td> %s </td><\n", $row['date']);
            printf("\t<td> %s </td><\n", $row['title']);
            echo "\t</tr><\n";
        }
        ?>
        </table>
    </body>
</html>

<?php
// Closing connection
mysql_close($link);
?>
```
In Symphony, the entire code can be listed using just three files.

1. List Action

```php
<?php
class weblogActions extends sfActions
{
    public function executeList()
    {
        $this->posts = PostPeer::doSelect(new Criteria());
    }
}
?>
```
2. List Template

```php
<?php foreach ($posts as $post): ?>
  <tr>
    <td><?php echo $post->getDate(); ?></td>
  </tr>
<?php endforeach; ?>
```

3. Layout

```html
<html>
  <head>
    <?php echo include_title(); ?>
  </head>
  <body>
    <?php echo $sf_data->getRaw('sf_content'); ?>
  </body>
</html>
```
Symfony libraries

- PEAR: This package has all the symphony libraries. Recommended for most users.
- Subversion(SVN): Recommended for experienced or advanced PHP developers.
- Pake: A CLI(Command line Interface) utility
- Lime: A unit testing utility
- Creole: A Database abstraction engine
- Propel: ORM tool. Object persistence and query service
- Phing: Build system used by model.
Page Creation in Symfony

- Symfony groups pages into modules.
- Before creating a page an empty module has to be created
- A default index action is created for each module. It has an action method and template file.
- The logic behind page is stored in action and the presentation is stored in template
- Pages without a logic would still require an action method that is empty
- Similarly if we execute a URL without a template method, symfony will give an error
Page Creation in Symfony(2)

Action example

- Adding an action means adding `executeAction` method to the Action class.
- The name of an action is always `executeXxx` where `Xxx` is the name of action.

```php
<?php

class mymoduleActions extends sfActions {
    public function executeMyAction() {
    }
}
```
Page Creation in Symfony(3)

Template example

Normal PHP syntax to print time

```php
<p>Hello, world!</p>
<?php
    if ($test)
    {
        echo "<p>".time()."</p>";
    }
?>
```

Alternative PHP syntax used for templates. It is more readable to the user

```php
<p>Hello, world!</p>
<?php if ($test): ?>
<p><?php echo time(); ?></p>
<?php endif; ?>
```
Passing Information

Passing information from action to template

- Action does all the complicated calculation, data retrieval and tests.
- Symphony makes the attributes of action class directly accessible to template in the global namespace.
- The following figure shows that the template has direct access to action attributes

```php
<p>Hello, world!</p>
<?php if ($hour >= 18): ?>
<p>Or should I say good evening? It is already <?php echo $hour ?>.</p>
<?php endif; ?>
```
Templates can use the traditional HTML forms as below

```php
<p>Hello, world!</p>
<?php if ($hour >= 18): ?>
<p>Or should I say good evening? It is already <?php echo $hour ?>.</p>
<?php endif; ?>
<form method="post" action="/myapp_dev.php/mymodule/anotherAction">
    <label for="name">What is your name?</label>
    <input type="text" name="name" id="name" value="" />
    <input type="submit" value="Ok" />
</form>
```

However, Symfony has helper functions defined which makes writing forms faster and easier

```php
<p>Hello, world!</p>
<?php if ($hour >= 18): ?>
<p>Or should I say good evening? It is already <?php echo $hour ?>.</p>
<?php endif; ?>
<?php echo form_tag('mymodule/anotherAction') ?><br>
    <?php echo label_for('name', 'What is your name?') ?><br>
    <?php echo input_tag('name') ?><br>
    <?php echo submit_tag('Ok') ?></form>
```
URL and Action Names

- Symfony’s routing system allows complete separation between actual action name and the form of URL needed to call it
- This allows custom formatting of URL
- Example:
  - A call to index of an action module could look like this
    - http://localhost/myapp_dev.php/article/index?id=123
  - But the URL can be written in a completely different way with a simple change in routing.yml configuration
  - The routing system automatically peels the request parameters from the smart URL
Linking Action to other action

- There is a total decoupling between the action name and its corresponding URL.
- Changing the way URLs look at later stages might become cumbersome since it will only work with default routing.
- To avoid this use the link_to() helper to create hyperlinks for the actions.
- The resulting HTML might look the same as previous however when we change the routing rules, all the templates reformats the URLs correctly.
Symphony Configuration

- Symphony configuration system uses YAML language to be simple and readable
- We can deal with multiple environments and also set parameters through definition cascade
- This provides versatility to the developer
- sfConfig is the object using which some configuration files can be accessed
- Symfony has a lot of configuration files and they are useful when high level of customization is required
Inside Control Layer

- In Symfony, control layer the code linking the business logic and presentation is split into several components for different purposes
- 1. Front Controller: It is the unique entry point to the application it loads configuration and determines the actions to execute
- Actions: They contain the logic. They check the integrity of a request and gets the presentation layer ready
- Request, response and session: These objects give access to request parameters, the response headers and the persistent user data.
- Filters: Portions of code executed for every request either before or after the action.
Inside the View Layer

- Templates: It contains HTML codes and some basic PHP codes. Usually calls to variables are declared in action and helper class
  - Helpers are PHP codes that return HTML code

- Code fragments: Often it might be required to insert HTML or PHP codes in several places. To avoid repeating these codes, PHP include() statement is used

- View Configuration: In Symfony, view consists of 2 parts
  - The HTML presentation of action result
  - Metadata, page title, File inclusions, Layout
Inside the Model layer

- Business logic of a web application mostly lies on its data model.
- Symphony’s default model component is based on an object or relational mapping layer.
- Symphony comes with the two most popular ORMs Propel and doctrine.
- In symphony, any data element is accessed using objects this provides high level of abstraction and portability.

Object Relational mapping

- PHP5 and Symfony are Object oriented. When accessing the database an interface is required to translate Object logic to Relational logic.
- ORM also provides encapsulation for data logic.
- They allow us to add accessors in an object that don’t necessarily match a column in a table.
In order to create the data object model that Symfony will use, whatever relational model the database has should be converted into data model.

Schema is the description of relational model which facilitates this translation.

**Schema example**

Consider a blog database table structure:

- **Blog Table**:
  - id
  - Title()
  - Content()
  - Created_at()

- **Article Table**:
  - id
  - Article_id()
  - Author()
  - Content()
  - Created_at()
Database Schema(2)

The schema.yml file related to the blog database looks like shown:

```
Article:
  actAs: [Timestampable]
  tableName: blog_article
  columns:
    id:
      type: integer
      primary: true
      autoincrement: true
    title: string(255)
    content: clob

Comment:
  actAs: [Timestampable]
  tableName: blog_comment
  columns:
    id:
      type: integer
      primary: true
      autoincrement: true
    article_id: integer
    author: string(255)
    content: clob
  relations:
    Article:
      onDelete: CASCADE
      foreignAlias: Comments
```

http://www.symfony-project.org/get/pdf/gentle-introduction-1.4-en.pdf
Symfony is simple to use irrespective of the expertise a person has in PHP programming.

The deciding factor is the size of the project.

If the requirement is just a simple web application with 5-10 pages then it would be better to stick with php.

For more complex application with heavy PHP is not enough. Symfony provides good maintenance and extension support.
References

http://www.symfony-project.org
http://www.symfony-project.org/book/1_0/o2-Exploring-Symfony-s-Code
http://www.symfony-project.org/get/pdf/gentle-introduction-1.4-en.pdf
http://forum.symfony-project.org/
http://groups.google.com/group/symfony-users
http://en.wikipedia.org/wiki/Model-view-controller