

The symfony platform

Create your very own framework

Fabien Potencier / Sensio Labs

Sensio / Me

- Founder of Sensio
 - Web Agency
 - Founded in 1998
 - 45 people
 - Open-Source Specialists
- Creator of symfony
 - PHP Web framework
 - Based on
 - 10 years of Sensio experience
 - Existing Open-Source projects

Web Framework

« Whatever the application, a framework is build to ease development by providing tools for recurrent and boring tasks. »

- Generic components
 - Built-in
 - Well integrated
 - To solve web problems
- Professionalize web development

The symfony Platform

- symfony is made of decoupled classes based on a small number of core classes
 - Event Dispatcher
 - Parameter Holder

- Classes with no dependency

cache, command, database, form, i18n, log, request, response, routing, storage, user, validator, widget

The symfony Platform

You can use all of those classes by themselves

... to create your own framework

Let's do it

The Goals

- We won't create a full stack framework
- We will create a framework customized for YOUR needs
- The code we will write today can be used as a bootstrap for your own framework

The Web Workflow

The **User** asks a **Resource** in a Browser

The Browser sends a **Request** to the Server

The Server sends back a **Response**

The Browser displays the **Resource** to the **User**

```
session_start();
```

```
if (isset($_GET['name']))  
{  
    $name = $_GET['name'];  
}  
else if (isset($_SESSION['name']))  
{  
    $name = $_SESSION['name'];  
}  
else  
{  
    $name = 'World';  
}
```

```
$_SESSION['name'] = $name;
```

```
echo 'Hello ' . $name;
```

PHP Global arrays
Built-in PHP functions

User



Resource



Request

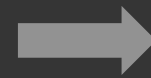


Response

Move to OOP

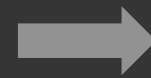
- Use objects instead of global arrays and functions

- `$_GET`, `$_POST`, `getcookie()`



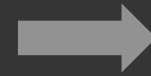
Request

- `echo`, `header()`, `setcookie()`



Response

- `$_SESSION`



User

- Why ?

- To add behaviors to those objects

- To have several requests, users, responses in one PHP process (functional testing)

- To be able to mock those objects to ease testing

sfWebRequest

PHP

```
$_SERVER['REQUEST_METHOD']  
$_GET['name']
```

```
get_magic_quotes_gpc() ?  
    stripslashes($_COOKIE[$name]) : $_COOKIE[$name];  
  
{  
    isset($_SERVER['HTTPS'])  
    && {  
        strtolower($_SERVER ['HTTPS']) == 'on'  
        ||  
        strtolower($_SERVER ['HTTPS']) == 1  
    }  
    || {  
        isset($_SERVER ['HTTP_X_FORWARDED_PROTO'])  
        &&  
        strtolower($_SERVER ['HTTP_X_FORWARDED_PROTO']) == 'https'  
    }  
}
```

Object

```
→getMethod()  
→getParameter('name')
```

```
→getCookie('name')
```

```
→isSecure()
```

Abstract
Parameters
and Headers

sfWebResponse

PHP

```
echo 'Hello World!'
```

```
header('HTTP/1.0 404 Not Found')
```

```
setcookie('name', 'value')
```

Object

```
→setContent('Hello World')
```

```
→setStatusCode(404)
```

```
→setCookie('name', 'value')
```

Abstract
Headers
and Cookies

sfUser / sfStorage

PHP

```
$_SESSION['name'] = 'value'
```

Object

```
→setAttribute('name', 'value')
```

```
→setCulture('fr')
```

```
→setAuthenticated(true)
```

Native session storage +
MySQL, PostgreSQL, PDO, ...

Abstract
\$_SESSION
Add features

sfEventDispatcher

- Allow decoupled objects to communicate

```
// sfUser  
$event = new sfEvent($this, 'user.change_culture', array('culture' => $culture));  
$dispatcher->notify($event);
```

```
// sfI18N  
$callback = array($this, 'listenToChangeCultureEvent');  
$dispatcher->connect('user.change_culture', $callback);
```

Based on
Cocoa Notification Center

- sfI18N and sfUser are decoupled
- « Anybody » can listen to any event
- You can notify existing events or create new ones

```
session_start();
```

```
if (isset($_GET['name']))
```

```
{
```

```
    $name = $_GET['name'];
```

```
}
```

```
else if (isset($_SESSION['name']))
```

```
{
```

```
    $name = $_SESSION['name'];
```

```
}
```

```
else
```

```
{
```

```
    $name = 'World';
```

```
}
```

```
$_SESSION['name'] = $name;
```

```
echo 'Hello '.$name;
```

← sfWebRequest

← sfUser

← sfWebResponse

User



Resource



Request



Response

Install symfony

- Install symfony 1.1 (via PEAR or Subversion)

```
$ svn co http://svn.symfony-project.com/branches/1.1 symfony
```

- Core classes are autoloaded

```
require dirname(__FILE__).'/../symfony/lib/autoload/sfCoreAutoload.class.php';  
sfCoreAutoload::register();
```

```
require dirname(__FILE__).'/../symfony/lib/autoload/sfCoreAutoload.class.php';
sfCoreAutoload::register();
```

```
$dispatcher = new sfEventDispatcher();
$request = new sfWebRequest($dispatcher);
```

```
session_start();
```

```
if ($request->hasParameter('name'))
{
    $name = $request->getParameter('name');
}
else if (isset($_SESSION['name']))
{
    $name = $_SESSION['name'];
}
else
{
    $name = 'World';
}
```

```
$_SESSION['name'] = $name;
```

```
echo 'Hello '.$name;
```



```
if (isset($_GET['name']))
{
    $name = $_GET['name'];
}
```

User



Resource



Request



Response

```
require dirname(__FILE__).'/../symfony/lib/autoload/sfCoreAutoload.class.php';
sfCoreAutoload::register();

$dispatcher = new sfEventDispatcher();
$request = new sfWebRequest($dispatcher);

session_start();

if (!$name = $request->getParameter('name'))
{
    if (isset($_SESSION['name']))
    {
        $name = $_SESSION['name'];
    }
    else
    {
        $name = 'World';
    }
}

$_SESSION['name'] = $name;

echo 'Hello '.$name;
```

→getParameter() returns null if the parameter is not defined

User



Resource



Request



Response

```
require dirname(__FILE__).'/../symfony/lib/autoload/sfCoreAutoload.class.php';
sfCoreAutoload::register();
```

```
$dispatcher = new sfEventDispatcher();
$request = new sfWebRequest($dispatcher);
session_start();
```

```
if (!$name = $request->getParameter('name'))
{
    if (isset($_SESSION['name']))
    {
        $name = $_SESSION['name'];
    }
    else
    {
        $name = 'World';
    }
}
```

```
$_SESSION['name'] = $name;
```

```
$response = new sfWebResponse($dispatcher);
$response->setContent('Hello '.$name);
$response->send();
```

```
echo 'Hello '.$name;
```



User



Resource



Request



Response

```
require dirname(__FILE__). '/../symfony/lib/autoload/sfCoreAutoload.class.php';
sfCoreAutoload::register();
```

```
$dispatcher = new sfEventDispatcher();
```

```
$storage = new sfSessionStorage();
```

```
$user = new sfUser($dispatcher, $storage);
```

```
$request = new sfWebRequest($dispatcher);
```

```
if (!$name = $request->getParameter('name'))
```

```
{
```

```
    if (!$name = $user->getAttribute('name'))
```

```
    {
```

```
        $name = 'World';
```

```
    }
```

```
}
```

```
$user->setAttribute('name', $name);
```

```
$response = new sfWebResponse($dispatcher);
```

```
$response->setContent('Hello '.$name);
```

```
$response->send();
```

```
session_start();
```

```
else if (isset($_SESSION['name']))
```

```
{
```

```
    $name = $_SESSION['name'];
```

```
}
```

```
$_SESSION['name'] = $name;
```

User



Resource



Request



Response

```
require dirname(__FILE__).'/../symfony/lib/autoload/sfCoreAutoload.class.php';  
sfCoreAutoload::register();
```

```
$dispatcher = new sfEventDispatcher();
```

```
$storage = new sfSessionStorage();
```

```
$user = new sfUser($dispatcher, $storage);
```

```
$request = new sfWebRequest($dispatcher);
```

```
$name = $request->getParameter('name', $user->getAttribute('name', 'World'));
```

```
$user->setAttribute('name', $name);
```

```
$response = new sfWebResponse($dispatcher);
```

```
$response->setContent('Hello '.$name);
```

```
$response->send();
```

User



Resource



Request



Response

sfRouting

- Clean URLs \leftrightarrow Resources
- Parses `PATH_INFO` to inject parameters in the `sfRequest` object
- Several strategies: `PathInfo`, `Pattern`
- Decouples Request and Controller

```
require dirname(__FILE__).'/../symfony/lib/autoload/sfCoreAutoload.class.php';
sfCoreAutoload::register();
```

```
$dispatcher = new sfEventDispatcher();
$storage = new sfSessionStorage();
$user = new sfUser($dispatcher, $storage);
$routing = new sfPatternRouting($dispatcher);
$routing->connect('hello', '/hello/:name');
$request = new sfWebRequest($dispatcher);
```

/step.php?name=Fabien



/step.php/hello/Fabien

```
$name = $request->getParameter('name', $user->getAttribute('name', 'World'));
```

```
$user->setAttribute('name', $name);
```

```
$response = new sfWebResponse($dispatcher);
$response->setContent('Hello '.$name);
$response->send();
```

User



Resource



Request



Response

The Web Workflow

The **User** asks a **Resource** in a Browser

The Browser sends a **Request** to the Server

The Server sends back a **Response**

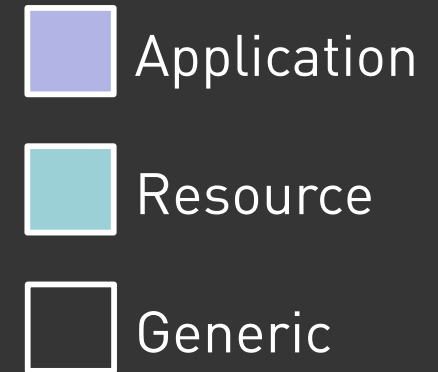
The Browser displays the **Resource** to the **User**

Let's create
a new framework

Code name: fp

```
require dirname(__FILE__).'/../symfony/lib/autoload/sfCoreAutoload.class.php';
sfCoreAutoload::register();
```

```
$dispatcher = new sfEventDispatcher();
$storage = new sfSessionStorage();
$user = new sfUser($dispatcher, $storage);
$routing = new sfPatternRouting($dispatcher);
$routing->connect('hello', '/hello/:name');
$request = new sfWebRequest($dispatcher);
```



```
$name = $request->getParameter('name', $user->getAttribute('name', 'World'));
$user->setAttribute('name', $name);
```

```
$response = new sfWebResponse($dispatcher);
$response->setContent('Hello '.$name);
$response->send();
```

User



Resource

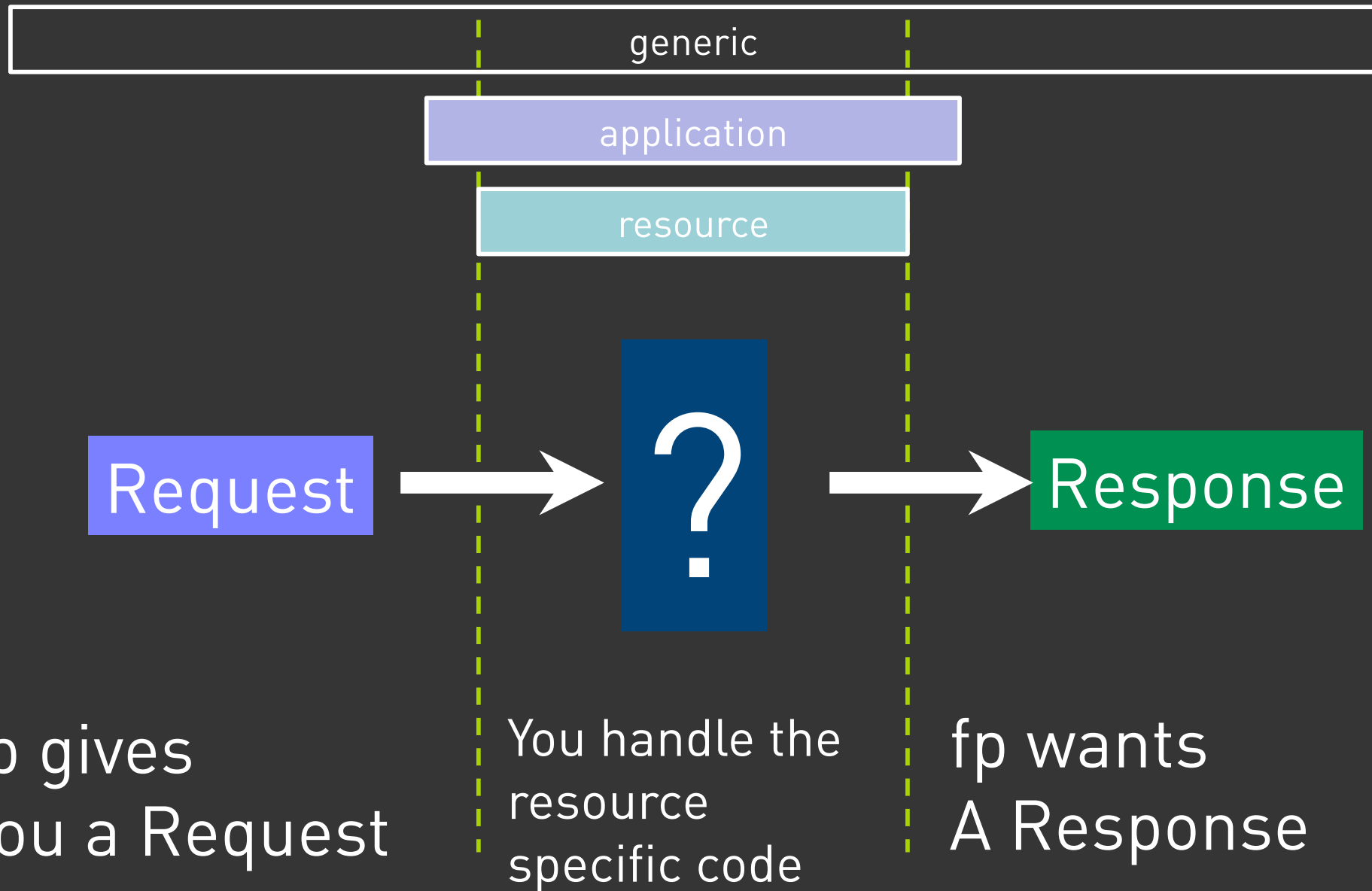


Request



Response

Resource specific code



- The dispatching logic is the same for every resource
- The business logic depends on the resource and is managed by the controller
- The controller responsibility is to « convert » the request to a response

```
require dirname(__FILE__).'/../symfony/lib/autoload/sfCoreAutoload.class.php';
sfCoreAutoload::register();
```

```
$dispatcher = new sfEventDispatcher();
$storage = new sfSessionStorage();
$user = new sfUser($dispatcher, $storage);
$routing = new sfPatternRouting($dispatcher);
$routing->connect('hello', '/hello/:name');
$request = new sfWebRequest($dispatcher);
```

```
$controller = new helloController();
```

```
$response = $controller->indexAction($dispatcher, $request, $user);
```

```
$response->send();
```

```
class helloController
{
    function indexAction($dispatcher, $request, $user)
    {
        $name = $request->getParameter('name', $user->getAttribute('name', 'World'));

        $user->setAttribute('name', $name);

        $response = new sfWebResponse($dispatcher);
        $response->setContent('Hello '.$name);

        return $response;
    }
}
```

The framework creation process

- We write code that just works
- We abstract the code to make it generic

```
require dirname(__FILE__).'/../symfony/lib/autoload/sfCoreAutoload.class.php';
sfCoreAutoload::register();
```

```
$dispatcher = new sfEventDispatcher();
$storage = new sfSessionStorage();
$user = new sfUser($dispatcher, $storage);
$routing = new sfPatternRouting($dispatcher);
$routing->connect('hello', '/hello/:name',
    array('controller' => 'hello', 'action' => 'index'));
$request = new sfWebRequest($dispatcher);
```

```
$class = $request->getParameter('controller').'Controller';
$method = $request->getParameter('action').'Action';
$controller = new $class();
```

```
$response = $controller->$method($dispatcher, $request, $user);
```

```
$response->send();
```

sfPatternRouting accepts default parameter values

```
require dirname(__FILE__).'/../symfony/lib/autoload/sfCoreAutoload.class.php';
sfCoreAutoload::register();
```

 Application

```
$dispatcher = new sfEventDispatcher();
```

 Resource

```
$storage = new sfSessionStorage();
```

```
$user = new sfUser($dispatcher, $storage);
```

 Generic

```
$routing = new sfPatternRouting($dispatcher);
```

```
$routing->connect('hello', '/hello/:name',
    array('controller' => 'hello', 'action' => 'index'));
```

```
$request = new sfWebRequest($dispatcher);
```

```
$class = $request->getParameter('controller').'Controller';
```

```
$method = $request->getParameter('action').'Action';
```

```
$controller = new $class();
```

```
$response = $controller->$method($dispatcher, $request, $user);
```

```
$response->send();
```

The Request Handler

- Handles the dispatching of the request
- Calls the Controller
- Has the responsibility to return a `sfResponse`

```
$dispatcher = new sfEventDispatcher();
$storage = new sfSessionStorage();
$user = new sfUser($dispatcher, $storage);
$routing = new sfPatternRouting($dispatcher);
$routing->connect('hello', '/hello/:name',
    array('controller' => 'hello', 'action' => 'index'));
$request = new sfWebRequest($dispatcher);
```

```
$response = fpRequestHandler::handle($dispatcher, $request, $user);
```

```
$response->send();
```



```
$class = $request->getParameter('controller').'Controller';
$method = $request->getParameter('action').'Action';
$controller = new $class();

$response = $controller->$method($dispatcher, $request, $user);
```

```
class fpRequestHandler
{
    static function handle($dispatcher, $request, $user)
    {
        $class = $request->getParameter('controller').'Controller';
        $method = $request->getParameter('action').'Action';

        $controller = new $class();
        $response = $controller->$method($dispatcher, $request, $user);

        return $response;
    }
}
```

```
require dirname(__FILE__).'/../symfony/lib/autoload/sfCoreAutoload.class.php';
sfCoreAutoload::register();
```

 Application

```
$dispatcher = new sfEventDispatcher();
```

 Resource

```
$storage = new sfSessionStorage();
```

```
$user = new sfUser($dispatcher, $storage);
```

 Generic

```
$routing = new sfPatternRouting($dispatcher);
```

```
$routing->connect('hello', '/hello/:name',
    array('controller' => 'hello', 'action' => 'index'));
```

```
$request = new sfWebRequest($dispatcher);
```

```
$response = sfRequestHandler::handle($dispatcher, $request, $user);
```

```
$response->send();
```

Abstract object management

- I need a container for my application objects
 - The dispatcher
 - The user
 - The routing
 - The i18n
 - The database
 - ...
- These objects are specific to my Application and do not depend on the Request

```
$dispatcher = new sfEventDispatcher();  
$application = new helloApplication($dispatcher);  
  
$request = new sfWebRequest($dispatcher);  
  
$response = $application->handle($request);  
  
$response->send();
```

```

class helloApplication
{
    public $dispatcher, $user, $routing;

    function __construct($dispatcher)
    {
        $this->dispatcher = $dispatcher;

        $storage = new sfSessionStorage();
        $this->user = new sfUser($this->dispatcher, $storage);

        $this->routing = new sfPatternRouting($this->dispatcher);
        $this->routing->connect('hello', '/hello/:name',
            array('controller' => 'hello', 'action' => 'index'));
    }

    function handle($request)
    {
        return fpRequestHandler::handle($this->dispatcher, $request, $this->user);
    }
}

```

Instead of passing a dispatcher around,
pass the application object

```

class helloApplication
{
    // ...

    function handle($request)
    {
        return fpRequestHandler::handle($this, $request);
    }
}

class fpRequestHandler
{
    static function handle($application, $request)
    {
        $class = $request->getParameter('controller').'Controller';
        $method = $request->getParameter('action').'Action';

        $controller = new $class();
        $response = $controller->$method($application, $request);

        return $response;
    }
}

class helloController
{
    function indexAction($application, $request)
    {
        $name = $request->getParameter('name', $application->user->getAttribute('name', 'World'));

        $application->user->setAttribute('name', $name);

        $response = new sfWebResponse($application->dispatcher);
        $response->setContent('Hello '.$name);

        return $response;
    }
}

```

fpRequestHandler
is now generic

```
class helloApplication
```

```
{  
    public $dispatcher, $user, $routing;
```

```
    function __construct($dispatcher)
```

```
    {  
        $this->dispatcher = $dispatcher;
```

```
        $storage = new sfSessionStorage();
```

```
        $this->user = new sfUser($this->dispatcher, $storage);
```

```
        $this->routing = new sfPatternRouting($this->dispatcher);
```

```
        $this->routing->connect('hello', '/hello/:name',  
            array('controller' => 'hello', 'action' => 'index'));
```

```
    }
```

```
    function handle($request)
```

```
    {
```

```
        return fpRequestHandler::handle($this->dispatcher, $request, $this->user);
```

```
    }
```

```
}
```

 Application

 Generic

Create a `fpApplication` class

```

abstract class fpApplication
{
    public $dispatcher, $user, $routing;

    function __construct($dispatcher)
    {
        $this->dispatcher = $dispatcher;
        $this->user = new sfUser($this->dispatcher, new sfSessionStorage());
        $this->routing = new sfPatternRouting($this->dispatcher);

        $this->configure();
    }

    abstract function configure();

    function handle($request)
    {
        return fpRequestHandler::handle($this, $request);
    }
}

class helloApplication extends fpApplication
{
    function configure()
    {
        $this->routing->connect('hello', '/hello/:name',
            array('controller' => 'hello', 'action' => 'index'));
    }
}

```

Move the public properties to accessor methods

```

abstract class fpApplication
{
    protected $dispatcher, $storage, $user, $routing;
    function __construct($dispatcher)
    {
        $this->dispatcher = $dispatcher;
        $this->configure();
    }

    function getDispatcher()
    {
        return $this->dispatcher;
    }

    function getStorage()
    {
        if (is_null($this->storage))
        {
            $this->storage = new sfSessionStorage();
        }

        return $this->storage;
    }

    function getUser()
    {
        if (is_null($this->user))
        {
            $this->user = new sfUser($this->dispatcher, $this->getStorage());
        }

        return $this->user;
    }

    function getRouting()
    {
        if (is_null($this->routing))
        {
            $this->routing = new sfPatternRouting($this->dispatcher);
        }

        return $this->routing;
    }

    // ...
}

```

Sensible defaults

- Most of the time
 - The dispatcher is a `sfEventDispatcher`
 - The request is a `sfWebRequest` object
- Let's change the Application to take defaults

```
abstract class fpApplication
{
    function __construct($dispatcher = null)
    {
        $this->dispatcher = is_null($dispatcher) ? new sfEventDispatcher() : $dispatcher;

        // ...
    }

    function handle($request = null)
    {
        $request = is_null($request) ? new sfWebRequest($this->dispatcher) : $request;

        return fpRequestHandler::handle($this, $request);
    }
}

$application = new helloApplication();

$response = $application->handle();

$response->send();
```

More sensible defaults

- Most of the time
 - The controller creates a `sfWebResponse` object
 - ... with some content

- Let's introduce a new `Controller` abstract class

```
class fpController
{
    function __construct($application)
    {
        $this->application = $application;
    }
}
```

```
function render($content)
{
    $response = new sfWebResponse($this->application->dispatcher);
    $response->setContent($content);

    return $response;
}
```

```
class helloController extends fpController
{
    function indexAction($request)
    {
        $name = $request->getParameter('name', $this->application->getUser()->getAttribute('name', 'World'));

        $this->application->getUser()->setAttribute('name', $name);

        return $this->render('Hello '.$name);
    }
}
```

Move the framework

- Move the framework code to its own directory structure

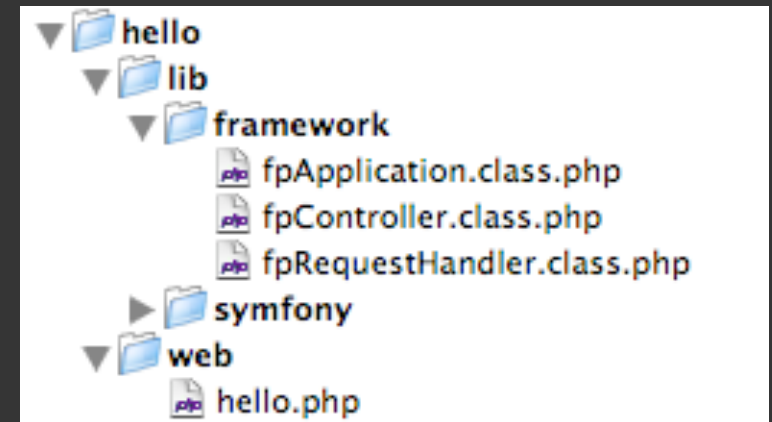
```
require dirname(__FILE__).'../../lib/symfony/lib/autoload/sfCoreAutoload.class.php';  
sfCoreAutoload::register();
```

```
require dirname(__FILE__).'../../lib/framework/fpApplication.class.php';  
require dirname(__FILE__).'../../lib/framework/fpController.class.php';  
require dirname(__FILE__).'../../lib/framework/fpRequestHandler.class.php';
```

```
class helloApplication extends fpApplication  
{  
    // ...  
}
```

```
class helloController extends fpController  
{  
    // ...  
}
```

```
$application = new helloApplication();  
$application->handle()->send();
```



Autoload our framework

```
require dirname(__FILE__).'/../lib/symfony/lib/autoload/sfCoreAutoload.class.php';  
sfCoreAutoload::register();
```

```
$autoload = sfSimpleAutoload::getInstance();  
$autoload->addDirectory(dirname(__FILE__).'/../lib/framework');  
$autoload->register();
```

```
class helloApplication extends fpApplication  
{  
    // ...  
}
```

```
class helloController extends fpController  
{  
    // ...  
}
```

```
$application = new helloApplication();  
$application->handle()->send();
```

Create a bootstrap file

```
require dirname(__FILE__).'../lib/framework/bootstrap.php';
```

```
class helloApplication extends fpApplication  
{  
    // ...  
}
```

```
class helloController extends fpController  
{  
    // ...  
}
```

```
$application = new helloApplication();  
$application->handle()->send();
```

Move classes

- Move the application and controller classes to their own directory structure

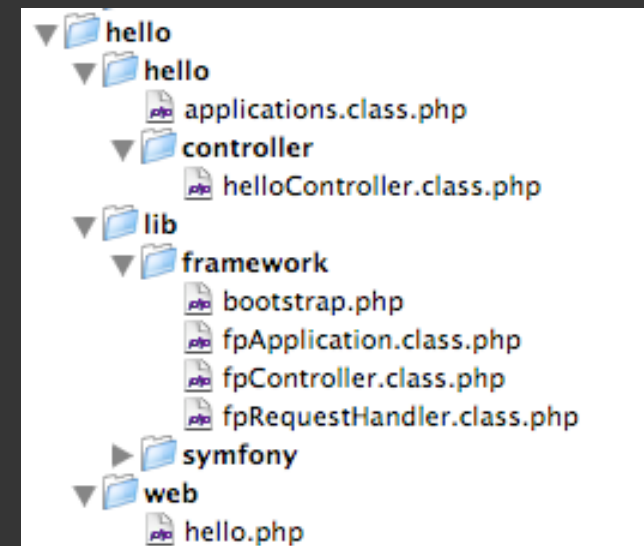
```
require dirname(__FILE__). '/../lib/framework/bootstrap.php';
```

```
require dirname(__FILE__). '/../hello/application.class.php';
```

```
require dirname(__FILE__). '/../hello/controller/helloController.class.php';
```

```
$application = new helloApplication();
```

```
$application->handle()->send();
```



Summary

- 3 generic classes
 - fpApplication
 - fpController
 - fpRequestHandler
- 2 specific classes
 - helloApplication
 - helloController
- A small bootstrap code

```
$application = new helloApplication();  
$application->handle()->send();
```

Conventions

- We already have some conventions
 - Controller class names
 - Action method names
- Let's add some directory name conventions
 - Controllers are in the controller directory in the same directory as `applications.class.php`
 - The controller file is the controller class name suffixed by `.class.php`

```

abstract class fpApplication
{
    function __construct($dispatcher = null)
    {
        $this->dispatcher = is_null($dispatcher) ? new sfEventDispatcher() : $dispatcher;

        $r = new ReflectionObject($this);
        $this->root = realpath(dirname($r->getFileName()));

        // ...
    }

    // ...
}

class fpRequestHandler
{
    static function handle($application, $request)
    {
        $class = $request->getParameter('controller').'Controller';
        $method = $request->getParameter('action').'Action';

        require_once $application->root.'/controller/'.$class.'.class.php';

        $controller = new $class();
        $response = $controller->$method($application, $request);

        return $response;
    }
}

```

Let's add a simple templating
system based on PHP

```

abstract class fpApplication
{
    protected $dispatcher, $storage, $user, $routing, $template;

    function getTemplate()
    {
        if (is_null($this->template))
        {
            $this->template = new fpTemplate();
        }

        return $this->template;
    }

    // ...
}

class fpTemplate
{
    function render($template, $parameters = array())
    {
        extract($parameters);

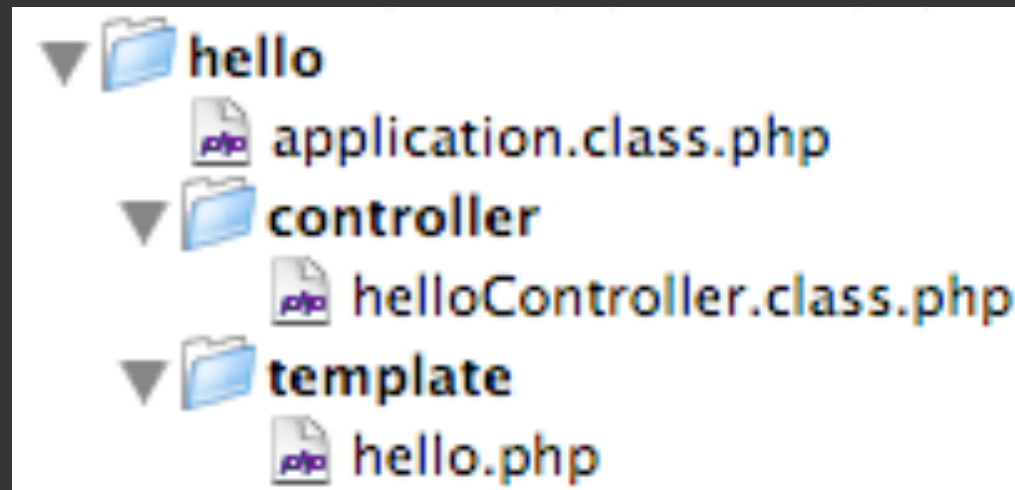
        ob_start();
        require $template;

        return ob_get_clean();
    }
}

```

The directory structure

- Add a template directory



```

class fpController
{
    // ...

    function render($template, $parameters = array())
    {
        $template = $this->application->root.'/template/'.$template;
        $content = $this->application->getTemplate()->render($template, $parameters);

        $response = new sfWebResponse($this->application->getDispatcher());
        $response->setContent($content);

        return $response;
    }
}

class helloController extends fpController
{
    function indexAction($request)
    {
        $name = $request->getParameter('name', $this->application->getUser()->getAttribute('name', 'World'));

        $this->application->getUser()->setAttribute('name', $name);

        return $this->render('hello.php', array('name' => $name));
    }
}

```

Write some tests

- Create a test/ directory to host test classes
- Use PHPUnit to test our controllers
- Change the session storage object

```
<?php
```

```
require_once 'PHPUnit/Framework.php';  
require dirname(__FILE__).'/../lib/framework/bootstrap.php';  
require dirname(__FILE__).'/../application.class.php';
```

```
class helloApplicationTest extends helloApplication  
{  
    function __construct($dispatcher = null)  
    {  
        parent::__construct($dispatcher);  
        $this->root = dirname(__FILE__).'/..';  
    }  
  
    function getStorage()  
    {  
        return new sfSessionTestStorage(  
            array('session_path' => '/tmp/quebec_demo', 'session_id' => '123')  
        );  
    }  
}
```

```

class helloControllerTest extends PHPUnit_Framework_TestCase
{
    protected function setUp()
    {
        $this->application = new helloApplicationTest();
    }

    public function testWithRequestParameter()
    {
        $_SERVER['PATH_INFO'] = '/hello/Fabien';
        $request = new sfWebRequest($this->application->getDispatcher());
        $response = $this->application->handle($request);

        $this->assertEquals('Hello Fabien', $response->getContent());
    }

    public function testWithSession()
    {
        $_SERVER['PATH_INFO'] = '/hello';
        $request = new sfWebRequest($this->application->getDispatcher());
        $response = $this->application->handle($request);

        $this->assertEquals('Hello Fabien', $response->getContent());
    }
}

```

Refactor the code to create
a `fpControllerTest` class

```

abstract class fpControllerTest extends PHPUnit_Framework_TestCase
{
    protected function setUp()
    {
        $r = new ReflectionObject($this);
        $root = realpath(dirname($r->getFileName()).'/.');

        require_once $root.'/application.class.php';

        $this->application = $this->getMock(
            basename($root).'Application', array('getStorage')
        );
        $this->application->root = $root;

        $storage = new sfSessionTestStorage(
            array('session_path' => '/tmp/quebec_demo', 'session_id' => '123')
        );

        $this->application->
            expects($this->any())->
            method('getStorage')->
            will($this->returnValue($storage))
        ;
    }

    // ...
}

```

Add XSS protection

- Add XSS protection by escaping all template parameters
- Use `sfOutputEscaper` symfony classes to do the job
- Update the tests

```

abstract class fpApplication
{
    // ...

    function getTemplate()
    {
        if (is_null($this->template))
        {
            $this->template = new fpTemplate($this->dispatcher);
        }

        return $this->template;
    }

    // ...
}

class fpTemplate
{
    function __construct($dispatcher)
    {
        $this->dispatcher = $dispatcher;
    }

    function render($template, $parameters = array())
    {
        $event = $this->dispatcher->filter(new sfEvent($this, 'template.filter_parameters'), $parameters);
        $parameters = $event->getReturnValue();

        // ...
    }
}

```

```

class helloApplication extends fpApplication
{
    function configure()
    {
        $this->getRouting()->connect('hello', '/hello/:name',
            array('controller' => 'hello', 'action' => 'index'));

        $this->getDispatcher()->connect('template.filter_parameters',
            array($this, 'escapeTemplateParameters'));
    }

    function escapeTemplateParameters(sfEvent $event, $parameters)
    {
        $parameters['sf_data'] = sfOutputEscaper::escape(array($this, 'htmlspecialchars'), $parameters);
        foreach ($parameters['sf_data'] as $key => $value)
        {
            $parameters[$key] = $value;
        }

        return $parameters;
    }

    function htmlspecialchars($value)
    {
        return htmlspecialchars($value, ENT_QUOTES, 'UTF-8');
    }
}

```

Move the generic code to `fpApplication`

```

class helloApplication extends fpApplication
{
    function configure()
    {
        $this->getRouting()->connect('hello', '/hello/:name',
            array('controller' => 'hello', 'action' => 'index'));

        $this->enableOutputEscaping();
    }
}

class fpApplication
{
    // ...

    function enableOutputEscaping()
    {
        $this->dispatcher->connect('template.filter.parameters',
            array($this, 'escapeTemplateParameters'));
    }

    function escapeTemplateParameters(sfEvent $event, $parameters)
    {
        $parameters['sf_data'] = sfOutputEscaper::escape(array($this, 'htmlspecialchars'), $parameters);
        foreach ($parameters['sf_data'] as $key => $value)
        {
            $parameters[$key] = $value;
        }

        return $parameters;
    }

    function htmlspecialchars($value)
    {
        return htmlspecialchars($value, ENT_QUOTES, 'UTF-8');
    }
}

```

Add custom 404, 500

- Allow custom 404 and 500 pages
 - 404 pages → `sfError404Exception`
 - 500 pages → `Exception`
- Change `fpRequestHandler`

Customization

- Add some events in the RequestHandler
 - application.request
 - application.controller
 - application.response
 - application.exception
- They can return a sfResponse and stop the RequestHandler flow

What for?

- Page caching
 - application.request: If I have the page in cache, unserialize the response from the cache and returns it
 - application.response : Serialize the response to the cache
- Security
 - application.controller: Check security and change the controller if needed

If we have time...

- Add CSS selector support to the test class
- Implement a page cache system
- Implement a security mechanism for controllers
- Improve performance with caching (routing, framework « compilation », ...)
- Add a CLI
- Implement a layout system
- Use symfony Forms
- Use a database/model

SENSIO S.A.

26, rue Salomon de Rothschild
92 286 Suresnes Cedex
FRANCE

Tél. : +33 1 40 99 80 80

Fax : +33 1 40 99 83 34

Contact

Fabien Potencier
fabien.potencier@sensio.com

<http://www.sensiolabs.com/>

<http://www.symfony-project.com/>