# Symfony

## Mastering book

**Монгол хэлнээ хөрвүүлсэн М.Мөнхбат**

https://www.facebook.com/munkhbat.mygmarsuren

# Агуулга

1-р бүлэг

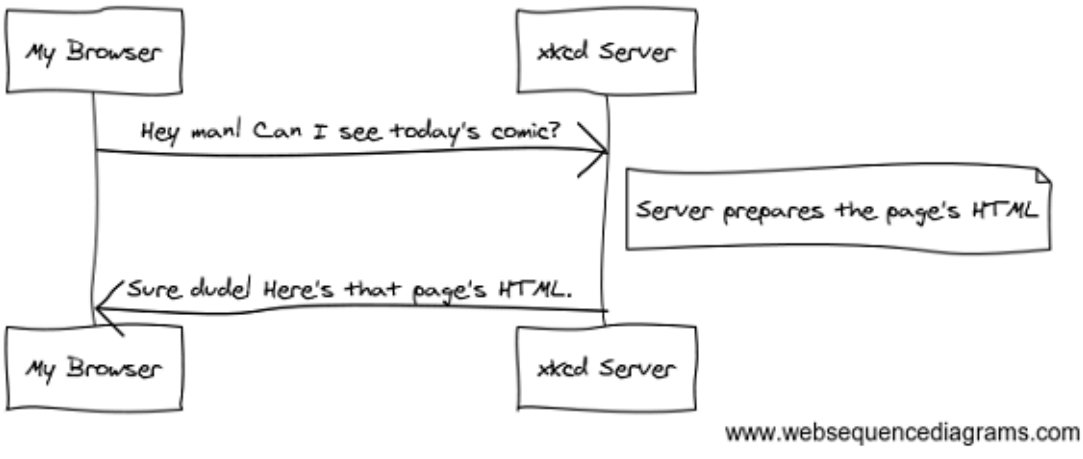
# Symfony2 болон HTTP-н тухай үндсэн ойлголт

Symfony2 фрэймворкыг судалсанаар та илүү бүтээлч, туршлагатай вэб хөгжүүлэгч болох болно.

Symfony2 бол энгийнээс эхлээд илүү цогц вэб програмыг хурдан хугацаанд бүтээх, хөгжүүлэх боломжыг танд олгож буй хүчирхэг хэрэгсэл юм. Symfony нь маш олон технологуудын шилдэг санаан дээр бүтээгдсэн бөгөөд олон жилийн турш, мянга мянган хүний хичээл зүтгэлээр таны судлах гэж буй өнөөгийн түвшинд хүрч ирсэн юм. Өөрөөр хэлбэл та Symfony –г судалсанаар вэбийн ажиллах үндсэн ойлголт, хөгжүүлэлтийн шилдэг арга барилуудыг судалж, олон олон гайхамшигтай PHP сангуудыг хэрхэн ашиглахыг мэдэж авах болно.

## HTTP гэж юу вэ?

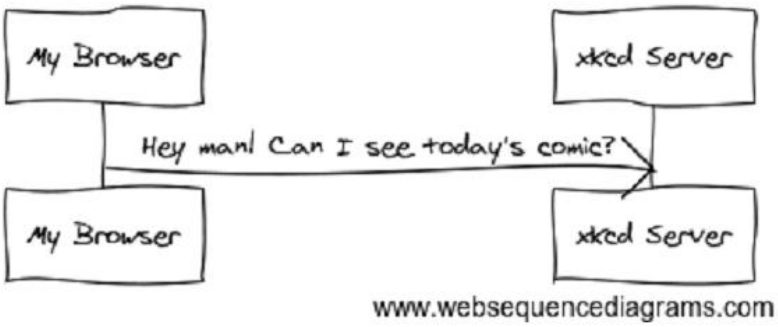
HTTP (Hypertext Transfer Protocol) нь 2 машин өөр хоорондоо харилцах боломжыг олгодог энгийн текст хэл юм. Жишээ нь: Хэрэглэгч хамгийн сүүлд гарсан комикс номын тухай мэдээллийг шалгах үед дараах харилцаа явагдана.



### Алхам 1. Клейнт –ээс Request (хүсэлт) илгээх

Вэб дээрх харилцаа бүр **request** (хүсэлт илгээх)– ээр эхэлнэ. **Request** гэдэг бол клейнт (вэб хөтөч, iPhone app гэх мэт)-ийн HTTP протоколын тусгай форматад оруулан үүсгэсэн текст мессеж юм. Клейнт нь тухайн request-ийг сервер лүү илгээгээд, хариу ирэхийг хүлээнэ.

Дээр үзүүлсэн вэб хөтөч болон **xkcd** [1] сервер 2-ийн хоорон явагдах харилцааны эхний хэсгийг харая.




Дээрх HTTP Request ерөнхийдөө дараах байдалтай байна.

```
1  GET / HTTP/1.1
2  Host: xkcd.com
3  Accept: text/html
4  User-Agent: Mozilla/5.0 (Macintosh)
```

Энэ энгийн текст мессеж клейнтийн юу хүсч байгаа тухай мэдээллийг агуулж байдаг.

Дээрх HTTP request-ийн эхний мөр нь **URI** болон **HTTP** method гэсэн 2 хэсгээс бүрдэнэ.

**URI** ( **/, /contact**, гэх мэт) нь клейнтийн хүсч байгаа мэдээллийн хаяг буюу байрлалыг заана. **HTTP method** (жишээ нь **GET**) нь тухайн мэдээлэл дээр ямар үйлдэл хийхийг тодорхойлно. HTTP method бол request-ийн үйл хөдлөл бөгөөд дараах хэдэн төрлийн method –ууд байдаг.

| | |
|---|---|
| GET | Серверээс мэдээлэл авна. |
| POST | Сервер дээр мэдээлэл үүсгэнэ. |
| PUT | Сервер дээрх мэдээллийг засна. |
| DELETE | Сервер дээрх мэдээллийг устгана. |

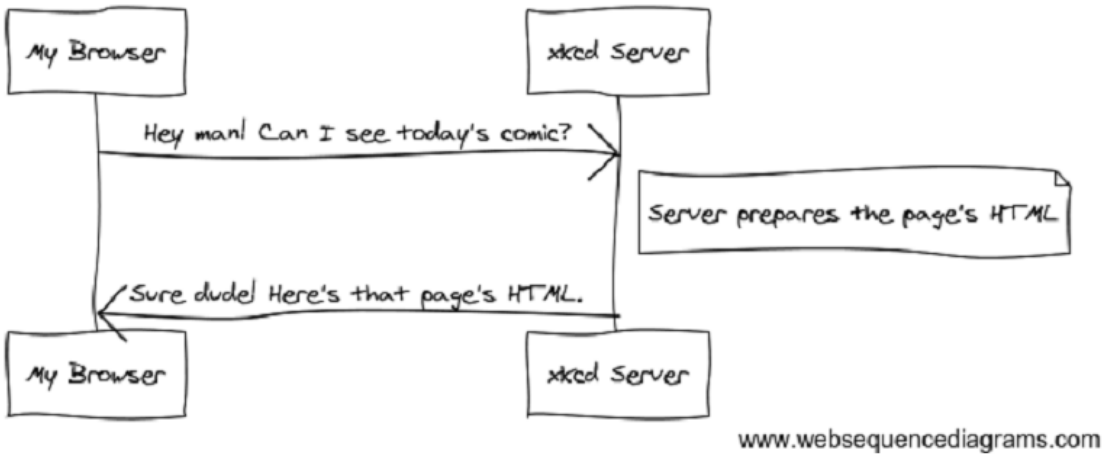
Ерөнхийдөө 9 төрлийн HTTP method байдаг боловч тэдгээрийн ихэнх нь төдийлөн ашиглагддаггүй, тэр бүү хэл зарим вэб хөтчүүдэд дэмжигддэггүй. Сүүлийн үеийн вэб хөтчүүд PUT, DELETE method-уудыг дэмждэггүй болсон.

Эхний мөрөөс гадна HTTP request нь **Request header** хэмээх мэдээллийн хэд хэдэн мөрийг агуулдаг. **Header** нь хүсэлт илгээж буй **Host**, клейнтийн хүлээн авах боломжтой **response** (хариу) -ийн формат (**Accept**), клейнтийн request илгээхэд ашиглаж байгаа програм (**User-Agent**) гэх зэрэг өргөн хэмжээний мэдээлэл агуулах боломжтой байдаг. Бусад Header-ийн тухай мэдээллийг Wikipedia сайтын **List of HTTP header fields**[2] сэдвээс харна уу.

[1] Xkcd бол жишээ болгон авсан вэб сервер юм.
[2] http://en.wikipedia.org/wiki/List_of_HTTP_header_fields

### Алхам 2: Серверээс Response (хариу) буцаах

Вэб сервер клейнтээс ирсэн request-ийг хүлээж аваад, клейнт ямар мэдээлэл хүсч байгаа тухай (URI) болон тухайн мэдээлэл дээр ямар үйлдэл (method) хийх гэж байгаа тухай мэдэж авна. Жишээ нь , GET хүсэлт байгаа тохиолдолд вбэ сервер мэдээллийг бэлтгээд **HTTP response** буцаана. Доорх зурган дээр Xkcd серверээс буцааж байгаа **Response**-ийг харуулав.




HTTP-рүү хөрвүүлэгдэн вэб хөтөч рүү буцааж байгаа Response нь дараах байдалтай байна.

```
HTTP/1.1 200 OK
Date: Sat, 02 Apr 2011 21:05:05 GMT
Server: lighttpd/1.4.19
Content-Type: text/html

<html>
<!-- ... xkcd comic-ийн HTML код энд байна -->
</html>
```

HTTP response нь клейнтийн хүссэн мэдээлэл (дээрх тохиолдолд HTML агуулга) болон Response –ийн тухай нэмэлт мэдээллийг агуулна. Эхний мөр нь HTTP response –ийн **status code** (энэ тохиолдолд 200) –ийг агуулдаг. Статус код нь клейнт рүү Request-ийн ерөнхий үр дүнг дамжуулдаг. Тухайлбал Request амжилттай болсон уу? Алдаа гарсан уу? Эсвэл өөр хуудасруу шилжүүлэх үү? гэх мэт. Wikipedia сайтын **List of HTTP status codes**[3] гэсэн сэдэвээс бүх статус кодыг хараарай.

Request-ийн нэгэн адил Response нь **HTTP header** гэгддэг мэдээллийн нэмэлт хэсгүүдийг агуулдаг.

Жишээ нь, нэг чухал HTTP response header бол **Content-Type** юм. Серверээс илгээгдэж байгаа тухайн нэг мэдээллийн үндсэн хэсэг нь HTML, XML, JSON зэрэг олон төрлийн форматтай байж болох ба Content-Type header нь тухайн мэдээллийг клейнт рүү ямар форматаар буцаахыг заана. Клейнт рүү буцаах боломжтой файлын төрлүүдийг Wikipedia-н **List of common media types**[4] сэдэвээс хараарай.

[3] http://en.wikipedia.org/wiki/List_of_HTTP_status_codes
[4] http://en.wikipedia.org/wiki/Internet_media_type#List_of_common_media_types

### Request, Response ба вэб хөгжүүлэлт

Энэ **request – response** гэсэн харилцаа нь вэб дээрх бүхий л харилцааг зохицуулж байдаг үндсэн процесс юм. Таны ашигладаг програмчлалын хэл, бүтээж байгаа програмын төрөл (web, mobile, JSON API), таны програм хөгжүүлэлтийн арга барилаас үл хамааран програмын эцсийн зорилго бол **Request** –ийг ойлгон түүнд тохирох **Response**-ийг үүсгэн буцаах явдал юм.

HTTP протоколын тодорхойлолтын тухай дэлгэрэнгүй харах бол **HTTP 1.1 RFC**[5] эсвэл **HTTP Bis**[6] –ийг хараарай. Хэрэв та ямар нэгэн вэб дээр явгдаж байгаа request, response –ийг шууд харахыг хүсвэл Firefox хөтчид зорилусан **Live HTTP Headers**[7] өргөтгөлийг ашиглаж болно.

## PHP дэх request болон response

Тэгэхээр PHP ашиглан хэрхэн request болон response үүсгэх вэ?

```php
$uri = $_SERVER['REQUEST_URI'];
$foo = $_GET['foo'];
header( 'Content-type: text/html');
echo 'URI: '.$uri;
echo '"foo" параметерийн утга нь: '.$foo;
```

Энэ код request-ээс мэдээллийг авч, түүнийгээ ашиглан response үүсгэнэ. HTTP request мессежийг задлан шинжлэхийн оронд PHP –д байдаг **$_SERVER** болон **$_GET** гэх зэрэг request-ийн бүх мэдээллийг агуулж байдаг супер глобал хувисагчуудыг ашигладаг. Үүний адил HTTP response буцаахын оронд **header()** функц ашиглан Response header үүсгэн response мессежийн бодит агуулгыг хэвлэн гаргаж болно. Ингэж PHP нь тохирох HTTP response-ийг үүсгэн клейнт рүү буцаадаг.

```
HTTP/1.1 200 OK
Date: Sat, 03 Apr 2011 02:14:33 GMT
Server: Apache/2.2.17 (Unix)
Content-Type: text/html
The URI requested is: /testing?foo=symfony
The value of the "foo" parameter is: symfony
```

[5] http://www.w3.org/Protocols/rfc2616/rfc2616.html
[6] http://datatracker.ietf.org/wg/httpbis/
[7] https://addons.mozilla.org/en-US/firefox/addon/live-http-headers/

## Symfony дахь Request болон Response

Symfony нь HTTP request болон response –ийг хялбар үүсгэх боломжтой 2 ширхэг PHP классыг агуулдаг. **Request**[8] классаар request-ийн бүх л мэдээлэлтэй ажиллах боломжтой.

```php
use Symfony\Component\HttpFoundation\Request

$request = Request::createFromGlobals();

// зорьж буй URI (/about гэх мэт). Ямар нэг параметергүй
$request->getPathInfo();

// GET болон POST хувьсагчуудыг авна
request->query->get('foo');
request->request->get('bar', 'default value if bar does not exist');

// SERVER хувьсагчийг авна.
$request->server->get('HTTP_HOST');

// foo Утгаар нь ялгаж UploadFile-ийн обектийг авна.
$request->files->get('foo');

// COOKIE утгыг авна.
$request->cookies->get('PHPSESSID');

// HTTP request header-ийг авна.
$request->headers->get('host');
$request->headers->get('content_type');

$request->getMethod();  // GET, POST, PUT, DELETE, HEAD
$request->getLanguages();      // Клейнт хүлээн авах боломжтой хэлнүүдийг
агуулсан массив.
```

Request класс нь цааная маш их үйлдэлийг биелүүлж байдаг.  Тухайлбал, **isSecure()** method нь тухайн хэрэглэгч хамгаалагдсан холболтоор (HTTPS гэх мэт) холбогдож байгаа эсэхийг шалгадаг гэх мэт.

**ParameterBags болон Request аттрибутууд**

Дээр үзсэнээр **$_GET, $_POST** хувьсагчууд нь public хандалттай query болон request-ийн property-уудыг ашиглах боломжтой байна. Эдгээр обект бүр нь **ParameterBag**[9] обект бөгөөд **get()**[10]**, has()**[11]**, all()**[12] гэх мэт олон method-уудтай.

Request класс нь мөн тухайн програм дотороо хэрхэн ажиллах тухай тусгай мэдээллийг агуулж байдаг public хандалттай **attributes** property-тай байдаг.

[8] http://api.symfony.com/master/Symfony/Component/HttpFoundation/Request.html
[9] http://api.symfony.com/master/Symfony/Component/HttpFoundation/ParameterBag.html
[10] http://api.symfony.com/master/Symfony/Component/HttpFoundation/ParameterBag.html#get()
[11] http://api.symfony.com/master/Symfony/Component/HttpFoundation/ParameterBag.html#has()
[12] http://api.symfony.com/master/Symfony/Component/HttpFoundation/ParameterBag.html#all()

Мөн Symfony нь **Response** классыг агуулдаг. Энэ нь таны програмд обект хандалтад interface –ийг ашиглан клейнт рүү буцаах Response –ийг үүсгэхэд тань туслах болно.

```
use Symfony\Component\HttpFoundation\Response
$response = new Response();

$response->setContent('<html><body><h1>Hello world!</h1></body></html>');
$response->setStatusCode(Response::HTTP_OK);
$response->headers->set('Content-Type', 'text/html');

//HTTP headers-ийг хэвлэнэ.
$response->send();
```

Request болон Response классууд нь Symfony фраймворкын HttpFoundation хэмээх компонентийн бүрэлдэхүүн хэсэгт багтана.

## Request –ээс Response руу

HTTP –тэй адил Request болон Response обектууд нь маш энгийн зүйл юм. Харин програмчлалын хэцүү хэсэг бол тэдгээрийн хооронд юу хийгдэхийг бичих явдал билээ. Өөрөөр хэлбэл бодит байдал дээр request-ийн мэдээллийг хөрвүүлэн response үүсгэх кодыг л бид бүхэн байнга хийдэг гэсэн үг юм.

Магадгүй таны програм мэйл илгээх, форм бөглөх, өгөгдийн санд мэдээлэл хадгалах, HTML хуудас үзүүлэх, өгөгдлийг нууцлах зэрэг олон үйлдэл гүйцэтгэдэг байж болно. Харин та хэрхэн энэ бүгдийг зохион байгуулж, анх үүсгэсэн кодынхоо бүтцийг өөрчлөлгүйгээр вэб програмаа цаашид сайжруулах вэ?

### Front controller

Уламжлалт програмчлалд вэб сайтын хуудас бүр өөрийн гэсэн физик файлтай байдаг.

```
index.php
contact.php
blog.php
```

Энэ нь хэд хэдэн асуудлыг үүсгэнэ. URL-ийн уян хатан биш байдал (хэрэв та сайтынхаа link-үүдийг эвдлэхгүйгээр **blog.php** файлыг **news.php** болгон өөрчилөх бол яах вэ?) нь файл бүрт хамгаалалтын болон өгөгдлийн сангийн холболт хийх файлуудыг заавал гараар оруулж өгөх шаардлагыг бий болгоно.

Энэ асуудлыг ***front controller*** ашиглан шийднэ. Энэ нь нэг php файлд таны програмд ирж байгаа бүх request-ийг хүлээж авна гэсэн үг юм.

| | |
|---|---|
| /index.php | executes index.php |
| /index.php/contact | executes index.php |
| /index.php/blog | executes index.php |

Ингэснээр бүх request нэг л замаар ирнэ. Өөр өөр PHP файлаар URL-ийг биелүүлхийн оронд front controller байнга биелэгдэх бөгөөд вэб програмын өөр өөр хэсгүүд рүү URL-үүдийг чиглүүлж өгнө.

### Кодын зохион байгуулалтаа хадгалах

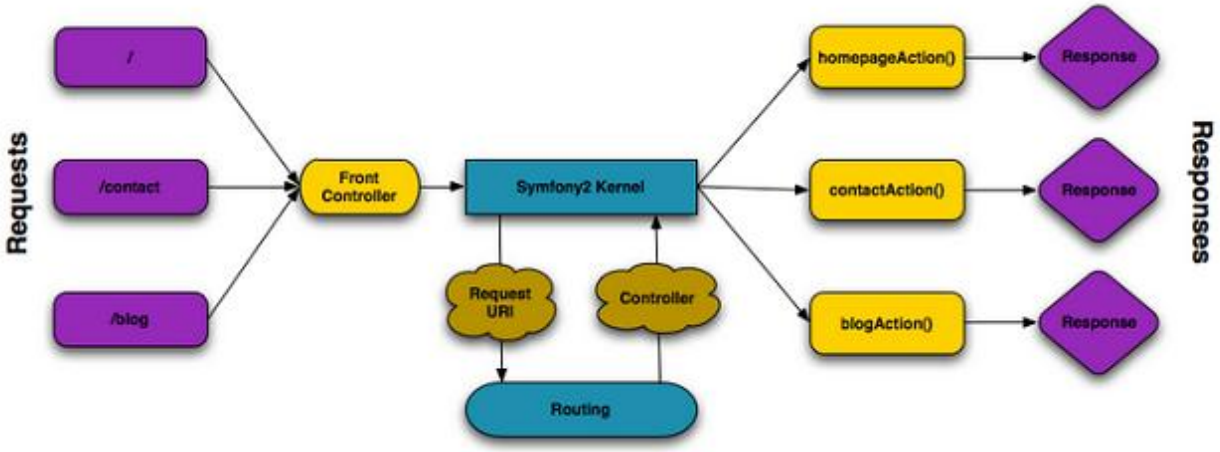
Та **front controller** дотороо аль код биелэгдэж ямар утга буцаах ёстойг зааж өгөх хэрэгтэй.  Энэ аргаар орж ирж байгаа URI-г шалгах ба тухайн утгаас хамаараад кодын ялгаатай хэсгүүдийг биелүүлдэг.

```
1  //index.php
2  use Symfony\Component\HttpFoundation\Request;
3  use Symfony\Component\HttpFoundation\Response;
4
5  $request = Request::createFromGlobals();
6  $path = $request->getPathInfo();
7
8  If(in_array($path, array(' ', '/'))) {
9     $response = new Response('Нүүр хуудсанд тавтай морил');
10 } elseif ($path == '/contact') {
11    $response = new Response('Холбоо барих');
12 } else {
13    $response = new Response('Хуудас олдсонгүй.', Response::HTTP_NOT_FOUND);
14 }
15 $response->send();
```

Асуудлыг ингэж шийдэх нь төвөгтэй байдаг. Тэгвэл Symfony энэ асуудлыг хэрхэн шийдвэрлэдэг тухай үзье.

### Symfony програмын бүтэц

Symfony -д request-ийг боловсруулах нь маш хялбар байдаг бөгөөд энэ нь дараах энгийн загварыг ашиглана.




Орж ирж байгаа request нь route –н тусламжтайгаар хөрвүүлэгдэн, **controller** функц руу дамжигдах ба энэ функц нь response обект буцаадаг.

URL –уудыг тохирох PHP функц руу чиглүүлж өгөх үүрэгтэй Routing файлд сайтын “хуудас” бүрийг тодорхойлж өгдөг. Front controller хэмээх PHP функцын үүрэг бол орж ирж байгаа request-ийн мэдээллийг ашиглан response обектыг үүсгэн буцаана. Өөрөөр хэлбэл controller нь request-ийг боловсруулаад, response обектийг буцаах үүрэгтэй юм. Энэ нь дараах алхамуудаар хийгдэнэ.

- Request орж ирэх бүрт Front controller файл ажиллана.
- Routing систем нь request ээс ирж байгаа мэдээлэл дээр үндэслэн аль PHP функц ажиллах ёстойг шийднэ.
- Тохирох PHP функц биелэгдсэнээр таны код тохирох Response обектийг үүсгэн буцаана.

### Action дахь Symfony Request

Та Symfony програмд **/contact** хуудас нэмэх хэрэгтэй байлаа гэж бодоё. Ингэхийн тулд эхлээд routing тохиргооны файлд **/contact** гэсэн хэсэг нэмнэ.

```
# app/config/routing.yml
contact:
    path:     /contact
    defaults: { _controller: AcmeDemoBundle:Main:contact }
```

Дээрх жишээнд routing тохиргооны файлыг *YAML* хэмээх файлд тодорхойлсон бөгөөд үүнийг мөн XML, PHP зэрэг өөр файлд үүсгэж бас болно.

Хэрэглэгч **/contact** хуудсанд зочилох үед энэ route нь ажиллах бөгөөд заасан controller биелэгдэнэ.

**AcmeDemoBundle:Main:contact** гэсэн мөр бол **MainController** хэмээх класс дотор байгаа **contactAction** гэсэн method-ийг зааж байна.

```
// src/Acme/DemoBundle/Controller/MainController.php
namespace Acme\DemoBundle\Controller;

use Symfony\Component\HttpFoundation\Response;

class MainController
{
      public function contactAction()
      {
            return new Response('<h1>Холбоо барих!</h1>');
      }
}
```

Дээрх Controller-ийн жишээнд **<h1>Холбоо барих!</h1>** гэсэн HTML мөрийг **Response**[13] обект болгон үүсгээд, буцааж байна.

[13] http://api.symfony.com/master/Symfony/Component/HttpFoundation/Response.html

# Symfony2: Өөрийн програмыг бүтээх.

Одоо та, ямар нэг вэб програмын зорилго бол орж ирж байгаа request-ийг хүлээн авч, тохирох response-ийг үүсгэх явдал гэдэгийг мэддэг боллоо. Програм цаашид томроод ирэхээр урьд нь үүсгэсэн кодынхоо зохион байгуулалтыг тэр хэвээр нь хадгалж, програмаа сайжруулахад хэцүү болдог.

### Symfony2 компонентууд

Тэгэхээр Symfony2 гэж юу вэ? Юун түрүүнд Symfony2 бол ямар ч PHP project-од ашиглаж болох 20 гаруй биеэ даасан library (сан) -уудын цуглуулга юм. Эдгээр сангуудыг **Symfony2 Components** гэж нэрлэдэг. Эдгээрээс заримыг нь дурдвал:

- ***HttpFoundation*** – Request, Response классууд болон session –ийг удирдах, файл upload хийхэд ашиглагддаг бусад классуудыг агуулдаг.
- ***Routing*** – Routing систем нь заасан URI-руу (жишээ нь /contact) тухайн Request-ийг хэрхэн боловсруулах тухай мэдээллийг (жишээ нь contactAction() method-ийг биелүүлэх) чиглүүлэх боломжыг олгоно.
- ***Form***[14] – Форм үүсгэх, формын мэдээллийг боловсруулахад зориулагдсан framework юм.
- ***Validator***[15] – Өгөгдлийн тухай дүрмийг тодорхойлох систем бөгөөд хэрэглэгчийн илгээж байгаа мэдээлэл тухайн дүрмийн дагуу байгаа эсэхийг шалгана.
- ***ClassLoader*** – Шаардлагатай классуудыг гар аргаар дуудахгүйгээр ашиглах боломжтой
- ***Templating*** – Template дээр хийгддэг нийтлэг үйлдлүүдийг биелүүлэхэд ашиглагдах хэрэгсэл юм.
- ***Security***[16] – Програм дахь хамгаалалтын бүх л төрлүүдийг хэрэгжүүлж байдаг хүчирхэг сан юм.
- ***Translating***[17] – Таны програм дахь текстүүдийг орчуулахад хэрэглэгддэг framework юм.

[14] https://github.com/symfony/Form
[15] https://github.com/symfony/Validator
[16] https://github.com/symfony/Security
[17] https://github.com/symfony/Translation

2-р бүлэг

# Symfony2 vs цэвэр PHP

Хэрэв та өмнө нь ямар нэг PHP framework ашиглаж байгаагүй бол MVC –ийн филисофийн тухай сайн мэдэхгүй байж магадгүй юм.

Энэ бүлэгт цэвэр PHP дээр энгийн програм бичиж үзэх бөгөөд энэ кодоо цааш нь илүү сайн зохион байгуулалтай болгож явах болно.

Дараа нь Symfony2 ашиглан олон дахин хийгддэг улиг болсон үйлдлүүдээс таныг хэрхэн чөлөөлөхийг үзэх болно.

## Цэвэр PHP ашиглан энгийн блог хийцгээе.

Энд бүгдээрэ цэвэр PHP ашиглан блог хийж үзье. Тэгэхээр эхлээд өгөгдлийн санд хадгалагдаж байгаа блогуудыг хэрэглэгчид харуулах нэг хуудас үүсгэх хэрэгтэй. Ингэж шууд PHP код бичих нь  хурдан ч их бохир ажил байдаг.

```php
<?php
// index.php
$link = mysql_connect('localhost', 'myuser', 'mypassword');
mysql_select_db('blog_db', $link);
$result = mysql_query('SELECT id, title FROM post', $link);
?>

<!DOCTYPE html>
<html>
    <head>
        <title>List of Posts</title>
    </head>
    <body>
        <h1>List of Posts</h1>
        <ul>
            <?php while ($row = mysql_fetch_assoc($result)): ?>
            <li>
                <a href="/show.php?id=<?php echo $row['id'] ?>">
```

```
19                     <?php echo $row['title'] ?>
20                 </a>
21             </li>
22             <?php endwhile; ?>
23         </ul>
24     </body>
25 </html>
26
27 <?php
28 mysql_close($link);
29 ?>
```

Энэ нь бичхэд хурдан, ажиллахад түргэн ч таны програм өсч томрох, сайжруулалт хийх боломжгүй болно. Энд хэд хэдэн асуудал байна.

- Алдаа шалгахгүй: Хэрэв өгөгдлийн сангийн холболтонд алдаа гарвал яах вэ?
- Зохион байгуулалт муу: Хэрэв програм томорвол энэ ганц файлыг сайжруулахад улам л хүндрэлтэй болно. Формын мэдээлэл боловсруулах кодоо хаана тавих вэ? Өгөгдлийг хэрхэн шалгах вэ? Мэйл илгээх кодоо хаана тавих вэ? Зэрэг асуудал гарна.
- Кодыг дахин ашиглахад хэцүү: Нэг файлд бүхнийг хийснээр програмын аль нэг хэсэгт тухайн кодыг дахин ашиглах аргагүй болно.

Дээр дурдаагүй өөр нэг асуудал бол MYSQL өгөгдлийн сан руу шууд холбогдож байгаа явдал юм. Үүний талаар дараа үзэх болно.

### Харагдах хэсгийг тусгаарлах

Дараах код нь HTML хуудсыг програмын логик хэсгээс тусдаа үүсгэх боломжыг олгоно.

```
1  <?php
2  // index.php
3  $link = mysql_connect('localhost', 'myuser', 'mypassword');
4  mysql_select_db('blog_db', $link);
5
6  $result = mysql_query('SELECT id, title FROM post', $link);
7
8  $posts = array();
9  while ($row = mysql_fetch_assoc($result)) {
10     $posts[] = $row;
11 }
12
13 mysql_close($link);
14
15 // include the HTML presentation code
16 require 'templates/list.php';
```

HTML код нь тусдаа файлд (templates/list.php) байрлана.

```
<!DOCTYPE html>
<html>
    <head>
        <title>List of Posts</title>
    </head>
    <body>
        <h1>List of Posts</h1>
        <ul>
            <?php foreach ($posts as $post): ?>
            <li>
                <a href="/read?id=<?php echo $post['id'] ?>">
                    <?php echo $post['title'] ?>
                </a>
            </li>
            <?php endforeach; ?>
        </ul>
    </body>
</html>
```

Програмын бүх логик хэсгийг агуулж байгаа **index.php** файлыг **Controller** гэж нэрлэнэ. Энэ нь хэрэглэгчийн илгээж байгаа мэдээллийг боловсруулах болон response –ийг бэлтгэх кодын хэсэгт хамаарна. Дээрх жишээн дээр Controller маань өгөгдлийн сангаас мэдээлэл авч, тухайн мэдээллийг харуулах загвар темплэйтийг оруулж ирж байна. Ингэж controller-ийг тусд нь хийснээр темплэйт файлыг хялбархан өөрчлөх боломжтой болгож байгаа юм.

### Програмын логик хэсгийг тусгаарлах

Одоогоор манай програм нэг л хуудас агуулж байна. Гэвч нэг ижил өгөгдлийн сангийн холболтыг ашиглах шаардлагатай өөр нэг хуудас үүсгэх хэрэгтэй бол яах вэ? Тэгвэл програмын өгөгдөлд хандах хэсгийг **model.php** гэсэн шинэ файлд тусгаарлан кодоо өөрчилөе.

```
<?php
// model.php
function open_database_connection()
{
    $link = mysql_connect('localhost', 'myuser', 'mypassword');
    mysql_select_db('blog_db', $link);

    return $link;
}

function close_database_connection($link)
{
    mysql_close($link);
}

function get_all_posts()
{
    $link = open_database_connection();

    $result = mysql_query('SELECT id, title FROM post', $link);
    $posts = array();
    while ($row = mysql_fetch_assoc($result)) {
```

```
        $posts[] = $row;
    }

    close_database_connection($link);

    return $posts;
}
```

Model.php гэсэн файлын нэр ашигласны учир нь уламжлалт програмын логик болон өгөгдлийн хэсэгт хандах хэсгийг “model” гэж нэрлэдэгтэй холбоотой.

Ингэснээр **Controller** (index.php) нь одоо маш хялбар болж байна.

```
<?php
require_once 'model.php';

$posts = get_all_posts();

require 'templates/list.php';
```

Одоо энэ controller програмын **model layer**-аас өгөгдлийг авч, тухайн өгөгдлийг хэрэглэгч рүү харуулах темплэйт-ийг дуудах гэсэн цөөхөн үйлдэлтэй боллоо. Энэ бол **model-view-controller** хэмээх загварыг харуулсан маш энгийн жишээ юм.

### Хуудасны ерөнхий бүтцийг тусгаарлах

Ингэж програмыг 3 тусдаа хэсэг болгосоноор олон давуу талыг бий болгох ба бараг ямар ч хуудсан дээр бүгдийг дахин ашиглах боломжтой болдог.

Харин одоо page layout (хуудасны ерөнхий бүтэц)-ийг хэрхэн тусгаарлахыг үзье. Үүнийг **laypout.php** гэсэн шинэ файл үүсгэн хийнэ.

```
<!-- templates/layout.php -->
<!DOCTYPE html>
<html>
    <head>
        <title><?php echo $title ?></title>
    </head>
    <body>
        <?php echo $content ?>
    </body>
</html>
```

Одоо темплэйт (**templates/list.php**) -ийг layout-аас удамшуулах боломжтой боллоо.

```
<?php $title = 'List of Posts' ?>

<?php ob_start() ?>
    <h1>List of Posts</h1>
    <ul>
        <?php foreach ($posts as $post): ?>
        <li>
            <a href="/read?id=<?php echo $post['id'] ?>">
```

```
                <?php echo $post['title'] ?>
            </a>
        </li>
            <?php endforeach; ?>
        </ul>
<?php $content = ob_get_clean() ?>

<?php include 'layout.php' ?>
```

Та одоо layout-ийг дахин дахин ашиглах боломжтой болсон. Гэхдээ үүнийг илүү боловсронгуй болгохын тулд темплэйт дахь хэдэн PHP функцүүд (**ob_start(), ob_get_clean()**) -ийг ашиглах шаардлагатай. Харин Symfony2-т ашиглагддаг **Templating component** нь энэ сайжруулалтыг цэвэрхэн, хялбар аргаар шийдвэрлэх боломж олгодог.

## Блогтоо "show" гэсэн хуудас шинээр нэмэх

Манай блогын **list** хуудасны кодын зохион байгуулалт сайжирч, тухайн кодыг програмын өөр өөр хэсгүүдэд дахин дахин ашиглах боломжтой болсон билээ. Одоо үүнийг шалгах зорилгоор блогын постуудыг **id** параметерээр нь ялган харуулах **show** нэртэй шинэ хуудас нэмж үзье.

Эхлээд **model.php** файлд шинэ функц үүсгэе. Энэ нь тухайн блогуудын бичлэгийг **id**-гаар нь ялган авах зорилготой.

```
// model.php
function get_post_by_id($id)
{
    $link = open_database_connection();

    $id = intval($id);
    $query = 'SELECT date, title, body FROM post WHERE id = '.$id;
    $result = mysql_query($query);
    $row = mysql_fetch_assoc($result);

    close_database_connection($link);

    return $row;
}
```

Үүний дараа **show.php** нэртэй шинэ файл үүсгэе. Энэ шинэ файл бол **controller** юм.

```
require_once 'model.php';

$post = get_post_by_id($_GET['id']);

require 'templates/show.php';
```

Эцэст нь тухайн блог постыг харуулах **templates/show.php** гэсэн темплэйт файл үүсгэнэ.

```
<?php $title = $post['title'] ?>

<?php ob_start() ?>
    <h1><?php echo $post['title'] ?></h1>

    <div class="date"><?php echo $post['date'] ?></div>
    <div class="body">
```

```
8           <?php echo $post['body'] ?>
9       </div>
10 <?php $content = ob_get_clean() ?>
11
12 <?php include 'layout.php' ?>
```

Ингэж 2 дахь хуудасыг үүсгэх нь хялбар бөгөөд код дахин давхардаагүй байна. Гэсэн хэдий ч энэ хуудсан дээр хэд хэдэн асуудал гарч ирж байна. Тухайлбал, буруу эсвэл байхгүй **id** дамжих үед хуудас дээр алдаа гарч эвдрэх болно. Энэ тохиолдолд 404 хуудсыг харуулах нь зөв юм. Гэхдээ энэ нь тийм хялбар хийх зүйл бас биш юм. Мууграар бодвол, та **intval()** функцээр **id** параметерээ цэвэрлэхээ мартсан тохиолдолд таны өгөгдлийн сан **SQL injection** халдлагад өртөж болзошгүй болно.

Өөр нэг гол асуудал бол **controller** файл бүрт **model.php** файлыг оруулж ирэх ёстой. Хэрэв controller файлыг өөр нэмэлт файлд оруулах эсвэл өөр бусад үйлдлүүдийг (жишээ нь enforce security) биелүүлэх хэрэгтэй болвол яах вэ?

## "Front Controller" ашиглах

Дээрх асуудлыг front controller ашиглан шийднэ. Үүнийг ашигласнаар бүх request нэг л php файлаар дамжин биелэгддэг.

```
1  front controller ашиглаагүй тохиолдолд:
2  /index.php         => list хуудас (index.php биелэнэ)
3  /show.php          => show хуудас (show.php биелэнэ)
4
5  front controller ашигласан тохиолдолд:
6  /index.php => list хуудас (index.php биелэнэ)
7  /index.php/show => show хуудас (index.php биелэнэ)
```

Front controller ашиглаж байгаа тохиолдолд нэг PHP файлд (манай жишээнд **index.php** байгаа) бүх request-ийг боловсруулна. Манай блогын **show** хуудас /**index.php**/show нь **index.php** файлд биелэгдэнэ.

### Front Controller үүсгэх

Нэг файлд бүх request-ийг боловсруулсанаар та **хамгаалалт, тохиргоо, routing** зэрэг зүйлсийг нэг дор төвлөрүүлэх болно. Одоо манай програмын index.php нь тухайн ирж байгаа хүсэлтийн URI дээр үндэслээд блогын **list** болон **show** хуудсын аль нэгийг харуулах болно.

```
1  // index.php
2
3  // load and initialize any global libraries
4  require_once 'model.php';
5  require_once 'controllers.php';
6
7  // route the request internally
8  $uri = $_SERVER['REQUEST_URI'];
9  if ('/index.php' == $uri) {
10 list_action();
11 } elseif ('/index.php/show' == $uri && isset($_GET['id'])) {
12 show_action($_GET['id']);
13 } else {
14 header('Status: 404 Not Found');
```

```
15 echo '<html><body><h1>Хуудас олдсонгүй</h1></body></html>';
16 }
```

Одоо controller-ууд (өмнө нь байсан index.php, show.php) маань энгийн PHP функц болсон бөгөөд хоёуланг нь **controllers.php** гэсэн тусдаа файлд оруулсан байна.

```
1  function list_action()
2  {
3      $posts = get_all_posts();
4      require 'templates/list.php';
5  }
6
7  function show_action($id)
8  {
9      $post = get_post_by_id($id);
10     require 'templates/show.php';
11 }
```

Front controller болох index.php нь core сангуудыг оруулан ирж, controller (**list_action()** болон **show_action()** функцүүд) –руу чиглүүлэн өгч байна.

Ингээд програм маань нэг PHP файлаас хөгжсөөр илүү зохион байгуулалт сайтай, кодыг дахин дахин ашиглах боломжтой болж байна. Гэсэнч блогоо хөгжүүлэхийн оронд маш их цагийг кодын бүтэц (routing, controller, templates зэрэг) дээр ажиллахад зарцуулж байна. Мөн бас формын мэдээлэл боловсруулах, хэрэглэгчээс ирж байгаа мэдээллийг шалгах, лог файл, хамгаалалт зэрэгт их цаг зарцуулах хэрэгтэй.

### Symfony2 ашиглах

Тэгвэл танд Symfony2 туслах болно. Symfony2-ийг ашиглахын тулд эхлээд үүнийг татан авах хэрэгтэй. Үүнийг **Composer** ашиглан хийх ба энэ нь тохирох хувилбар болон түүнд хамааралтай бүх хэсгийг татан авч, autoloader-ийг бэлтгэнэ. Autoloader бол класс агуулж байгаа файлыг шууд оруулж ирэхгүйгээр PHP классуудыг ашиглах боломж олгодог хэрэгсэл юм.

Эхлээд root буюу сайтын үндсэн хавтсанд дараах **composer.json** файлыг үүсгэнэ.

```
1  {
2      "require": {
3          "symfony/symfony": "2.4.*"
4      },
5      "autoload": {
6          "files": ["model.php","controllers.php"]
7      }
8  }
```

Дараа нь **Composer**[18]-ийг татан авааД, дараах коммандыг ажиллуулна. Энэ команд нь **vendor/** хавтас руу Symfony-г татах болно.

```
1  $ php composer.phar install
```

Энэ командыг ашиглан файлуудыг татахаас гадна, Composer нь дээрх **composer.json** файлын **autoload** хэсэгт тодорхойлсон файлуудыг татан авах зорилгоор **vendor/autoload.php** файлыг үүсгэнэ. Аливаа нэг вэб програмын үндсэн үүрэг бол request-ийг хөрвүүлэн, response буцаах юм гэж дээр үзсэн. Иймдээ ч Symfony2 нь Request болон Response классуудыг агуулдаг. Эдгээр классууд нь HTTP request-ийг

[18] http://getcomposer.org/download/

боловсруулан, HTTP response-ийг буцаах үүрэгтэй билээ. Одоо эдгээр классуудыг ашиглаад бидний жишээ болгон хийж байгаа блог сайтаа илүү болосронгуй болгоё.

```php
<?php
// index.php
require_once 'vendor/autoload.php';

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

$request = Request::createFromGlobals();

$uri = $request->getPathInfo();
if ('/' == $uri) {
    $response = list_action();
} elseif ('/show' == $uri && $request->query->has('id')) {
    $response = show_action($request->query->get('id'));
} else {
    $html = '<html><body><h1>Хуудас олдсонгүй</h1></body></html>';
    $response = new Response($html, Response::HTTP_NOT_FOUND);
}
// echo the headers and send the response
$response->send();
```

Одоо controller -оос Response обектийг буцаах хэрэгтэй. Үүнийг хийхийн тулд **render_template()** хэмээх функцийг ашиглах хэрэгтэй.

```php
// controllers.php
use Symfony\Component\HttpFoundation\Response;

function list_action()
{
    $posts = get_all_posts();
    $html = render_template('templates/list.php', array('posts' => $posts));

    return new Response($html);
}

function show_action($id)
{
    $post = get_post_by_id($id);
    $html = render_template('templates/show.php', array('post' => $post));

    return new Response($html);
}

// helper function to render templates
function render_template($path, array $args)
{
    extract($args);
    ob_start();
    require $path;
    $html = ob_get_clean();

    return $html;
}
```

Symfony2-ийн багахан хэсгийг ашиглахад л програм маань илүү уян хатан, найдвартай болж байна. **Request** ашиглан HTTP request-ийн тухай мэдээллийг авах нь найдвартай байдаг. Ялангуяа, **getPathInfo()** method нь URI-г цэвэрлээд (үргэлж **/show** гэж буцаах ба хэзээ ч **/index.php/show** гэж буцаахгүй) буцаадаг.

### Symfony2 дахь энгийн програм

Манай блог багагүй замыг туулж энд ирлээ. Хэдийгээр програм маань энгийн ч гэсэн бас л их хэмжээний кодыг агуулсан хэвээрээ байна. Үүний зэрэгцээ, энгийн **routing system** үүсгэж, **ob_start()** болон **ob_get_clean()** method-уудыг ашиглан темплэйтийг боловсруулсан билээ. Харин Symfony фрэймфоркт байдаг **Routing** болон **Templating** компонентуудыг ашиглан дээрх асуудлыг шийдвэрлэнэ.

Одоо энд Symfony2 ашиглан нэгэн энгийн жижиг програмыг байгуулая.

```
1  // src/Acme/BlogBundle/Controller/BlogController.php
2  namespace Acme\BlogBundle\Controller;
3
4  use Symfony\Bundle\FrameworkBundle\Controller\Controller;
5  class BlogController extends Controller
6  {
7      public function listAction()
8      {
9          $posts = $this->get('doctrine')
10             ->getManager()
11             ->createQuery('SELECT p FROM AcmeBlogBundle:Post p')
12             ->execute();
13
14         return $this->render(
15             'AcmeBlogBundle:Blog:list.html.php',
16             array('posts' => $posts)
17         );
18     }
19
20     public function showAction($id)
21     {
22         $post = $this->get('doctrine')
23             ->getManager()
24             ->getRepository('AcmeBlogBundle:Post')
25             ->find($id);
26
27         if (!$post) {
28             // cause the 404 page not found to be displayed
29             throw $this->createNotFoundException();
30     }
31
32     return $this->render(
33         'AcmeBlogBundle:Blog:show.html.php',
34         array('post' => $post)
35         );
36     }
37 }
```

Дээрх 2 controller-ийн аль аль нь **Doctrine ORM library**-г ашиглан өгөгдлийн сангаас обектийг авч, **Templating** компонент нь темплэйтийг боловсруулаад Response обектийг буцааж байна. Одоо list темплэйт нь илүү энгийн болох болно.

```
<!-- src/Acme/BlogBundle/Resources/views/Blog/list.html.php -->
<?php $view->extend('::layout.html.php') ?>

<?php $view['slots']->set('title', 'List of Posts') ?>

<h1>List of Posts</h1>
<ul>
    <?php foreach ($posts as $post): ?>
    <li>
        <a href="<?php echo $view['router']->generate(
            'blog_show',
            array('id' => $post->getId())
) ?>">
<?php echo $post->getTitle() ?>
</a>
</li>
<?php endforeach; ?>
</ul>
```

Layout нь дээр хийсэнтэй бараг адил байна.

```
<!-- app/Resources/views/layout.html.php -->
<!DOCTYPE html>
<html>
    <head>
        <title><?php echo $view['slots']->output(
            'title',
            'Default title'
        ) ?></title>
    </head>
    <body>
        <?php echo $view['slots']->output('_content') ?>
    </body>
</html>
```

Symfony2-ийн хөдөлгүүр (Kernel) ажиллаж эхлэхдээ тухайн request-ийн мэдээлэл дээр үндэслээд аль controller-ийг биелүүлэхээ мэддэг байх хэрэгтэй. Иймээс Routing тохиргооны файл нь энэ мэдээллийг агуулдаг.

```
# app/config/routing.yml
blog_list:
    path: /blog
    defaults: { _controller: AcmeBlogBundle:Blog:list }

blog_show:
    path: /blog/show/{id}
    defaults: { _controller: AcmeBlogBundle:Blog:show }
```

Symfony2 нь улиг болсон үйлдлүүдийг өөрөө гүйцэтгэх бөгөөд front controller нь маш энгийн байх болно. Front controller-ийг нэг удаа үүсгэсэн бол түүнтэй дахин ажиллах шаардлагагүй.

```php
// web/app.php
require_once __DIR__.'/../app/bootstrap.php';
require_once __DIR__.'/../app/AppKernel.php';

use Symfony\Component\HttpFoundation\Request;

$kernel = new AppKernel('prod', false);
$kernel->handle(Request::createFromGlobals())->send();
```

Тэгэхээр Front controller-ийн үндсэн ажил бол Symfony2-ийн хөдөлгүүр болох **Kernel**-ийг эхлүүлж, түүн рүү **Request** обектийг дамжуулна. Тэгээд Symfony2-ийн **Kernel** нь routing тохиргооны мэдээллийг ашиглаад аль controller-ийг дуудах эсэхээ шийднэ. Харин өмнөхтэй адилаар controller method нь эцсийн Response обектийг буцаана.

## Илүү сайжруулсан темплэйт

Symfony2 нь **Twig**[19] хэмээх темплэйт engine-ийг агуулдаг бөгөөд энэ нь бичихэд хурдан, уншихад хялбар байдаг ба код бичих ажлыг ихээхэн багасгадаг. Доорх жишээнд Twig ашиглан list хэмээх темплэйтийг хийсэн байна.

```twig
{# src/Acme/BlogBundle/Resources/views/Blog/list.html.twig #}
{% extends "::layout.html.twig" %}

{% block title %}List of Posts{% endblock %}

{% block body %}
    <h1>List of Posts</h1>
    <ul>
        {% for post in posts %}
        <li>
            <a href="{{ path('blog_show', {'id': post.id}) }}">
                {{ post.title }}
            </a>
        </li>
        {% endfor %}
    </ul>
{% endblock %}
```

Мөн **layout.html.twig** темплэйт нь доох байдлаар бичигдэнэ.

```twig
{# app/Resources/views/layout.html.twig #}
<!DOCTYPE html>
<html>
    <head>
        <title>{% block title %}Default title{% endblock %}</title>
    </head>
    <body>
        {% block body %}{% endblock %}
    </body>
</html>
```

[19] http://twig.sensiolabs.org

3-р бүлэг

# Symfony фрэймворкыг суулгах, тохируулах

Symfony –ийн хэд хэдэн тархацуудаас сонгон, өөрийн прожектдоо тохирох тархацыг татан авч програмаа шууд хөгжүүлж эхлэх боломжтой.

## Symfony2 тархацыг суулгах

Юун түрүүнд та PHP вэб серверийг компьютертаа суулгаж, тохируулсан эсэхийг шалгах хэрэгтэй гэдэгийг анхааараарай.

Symfony2 багцын “тархац” бол Symfony2-ийн цөм сангууд, олон хэрэгцээтэй bundle-ууд, зарим тохиргооны файлууд зэрэгийг багтаасан бүрэн ажиллагаатай програм юм. http://symfony.com/download сайтаас Symfony2-ийг татан авна. Энэ хуудаснаас Symfony2-ийн гол тархац болох Symfony Standard Edition-ийг татан авахыг зөвлөж байна. Үүнийг татан авах 2 арга байдаг.

**1-р арга: Composer**

**Composer**[20] бол PHP –д зориулсан удирдлагын сан бөгөөд **Symfony2 Standard Edition**-ийг татахад ашиглагдана.

Эхлээд та Composer-ийг өөрийн компьютер дээрээ татан авах хэрэгтэй. Хэрэв та **Curl** суулгасан бол дараах командаар composer-ийг татан авч болно.

```
1  $ curl -s https://getcomposer.org/installer | php
```

Хэрэв таны компьютер Composer –ийг ашиглахад бэлэн биш байгаа бол энэ командыг ажиллуулах үед танд хэдэн заавар өгөх болно. Тэдгээр зааварыг даган хийснээр Composer ямар нэг асуудалгүй ажиллах болно.

[20] http://getcomposer.org/

Энэ командыг ажиллуулсанаар Composer нь Standard тархацыг татахад ашиглагдах PHAR файлыг ажиллуулна.

```
$ php composer.phar create-project symfony/framework-standard-edition
/path/to/webroot/Symfony dev-master
```

Энэ команд Standard тархацыг татаж дуустал хэдэн минут ажиллана. Үүнийг дууссаны дараа татан авсан файл дараах бүтэцтэй харагдана.

```
path/to/webroot/              <- энэ таны вэб серверийн хавтас (заримдаа
htdocs эсвэл public нэртэй байна)
    Symfony/                  <- шинэ хавтас
        app/
            cache/
            config/
            logs/
        src/
            ...
        vendor/
            ...
        web/
            app.php
            ...
```

**2-р арга: Архив файлаар татаж авах**

Та мөн Standard Edition-ийг архив файл хэлбэрээр нь татан авч болно. Ингэхдаа дараах 2-н аль нэгээс сонгох хэрэгтэй.

- .tgz эсвэл .zip файлын аль нэгээр нь татан авах.
- Тухайн тархацыг vender-той, эсвэл vendor-гүйгээр татан авах. Хэрэв та олон thirdparty сангууд, bundle-уудыг ашиглах бөгөөд эдгээрийг Composer ашиглан удирдахаар төлөвлөсөн бол vendor-гүйгээр татан авах хэрэгтэй.

Архив файлыг татаж аваад, вэб серверийнхээ үндсэн хавтсанд хуулаад, файлаа задлах хэрэгтэй. UNIX command line ашиглаж хийх бол дараах командыг ажиллуулна. (### -ийн оронд тохирох файлын нэр байна.)

```
# for .tgz file
$ tar zxvf Symfony_Standard_Vendors_2.4.###.tgz

# for a .zip file
$ unzip Symfony_Standard_Vendors_2.4.###.zip
```

Хэрэв та vendor-гүйгээр татаж авсан бол дараагийн сэдэвийг сайтар унших хэрэгтэй.

Бүх public файлууд болон Symfony2 програмд орж ирж байгаа хүсэлтийг дамжуулах front contrøller зэрэг нь **Symfony/web/** хавтсанд байрлана. Тэгэхээр та веб серверийнхээ үндсэн хавтсанд татаж авсан файлаа задалсан бол таны програмыг эхлүүлэх URL нь http://localhost/Symfony/web/ байна.

### Vendor фхйлуудыг шинэчлэх

Symfony прожект нь олон тооны гадаад сангуудтай холбоотой байдаг. Эдгээр нь таны прожектийн vendor/ хавтас руу татагдсан байдаг.

Vender файлуудыг шинэчлэх нь танд хэрэгтэй vendor сангуудыг авах боломжыг олгоно.

*Алхам 1: Composer-ийг татах*

```
1  $ curl -s http://getcomposer.org/installer | php
```

Ингэснээр **composer.phar** файлыг **composer.json** файл байрлаж байгаа хавтас руу (энэ нь таны Symfony прожектийн үндсэн хавтас байна) татан авна.

*Алхам 2: Vendor-ийг суулгах*

```
1  $ php composer.phar install
```

Энэ командыг ажиллуулсанаар vendor/ хавтас руу бүх Vendor сангууд татагдах болно.

Хэрэв та curl суулгаагүй бол installer файлыг http://getcomposer.org/installer сайтаас өөрөө татан авах боломжтой. Энэ файлаа прожект дотороо байрлуулаад ажиллуулна.

```
1  $ php installer
2  $ php composer.phar install
```

**php composer.phar install** эсвэл **php composer.phar update** командыг ажиллуулах үед, Composer нь **install/update** командуудыг биелүүлж, кэйшийг цэвэрлэн, asset (css, js image гэх мэт файлууд)-ыг суулгана. Default-аар asset-ууд нь **web** хавтас руу хуулагдсан байна.

Symfony asset-ийг хуулахын оронд, хэрэв таны үйлдлийн систем дэмжих бол symlink-ийг үүсгэж болно. Symlink үүсгэхдээ, **composer.json** файлын extra гэсэн хэсэгт **symfony-assets-install** түлхүүр үгээр symlink гэсэн утгыг нэмнэ.

Listing 3-8

```
"extra": {
    "symfony-app-dir": "app",
    "symfony-web-dir": "web",
    "symfony-assets-install": "symlink"
}
```

### Тохиргоо

Ингээд **vendor/** хавтсанд шаардлагатай бүх third-party сангууд байрласан байна. Мөн програмын default буюу анхны тохиргоо нь **app/** хавтсанд, жишээ кодууд нь **src/** хавтсанд байрласан байна.

Symfony2 –т өөрт нь визуал сервер тохиргоо шалгагч байна. Та дараах хаягаар тохиргоогоо шалгаж болно.

```
1  http://localhost/config.php
```

Хэрэв ямар нэг алдаа илэрвэл тэдгээрийг засах хэрэгтэй.

**Permission тохируулах**

**app/cache** болон **app/logs** хавтасууд руу веб сервер болон command line-аас бичилт хийх боломжтой байх хэрэгтэй. UNIX системд, хэрэв веб сервер хэрэглэгч нь command line хэрэглэгчээс өөр байгаа тохиолдолд та дараах командыг ажиллуулан өөрийн прожектдоо тохирох permission-ийг зааж өгнө.

**1. chmod +a дэмждэг систем дээр ACL ашиглах**

Ихэнх системүүдэд chmod +a командыг ашиглах боломжтой байдаг. Энэ нь вэб сервер хэрэглэгчийг тодорхойлоход ашиглагддаг команд бөгөөд HTTPDUSER болгон тохируулна.

```
1 $ rm -rf app/cache/*
2 $ rm -rf app/logs/*
3
4 $ HTTPDUSER=`ps aux | grep -E '[a]pache|[h]ttpd|[_]www|[w]ww-
  data|[n]ginx' | grep -v
5 root | head -1 | cut -d\ -f1`
6 $ sudo chmod +a "$HTTPDUSER allow
  delete,write,append,file_inherit,directory_inherit"
7 app/cache app/logs
8 $ sudo chmod +a "`whoami` allow
  delete,write,append,file_inherit,directory_inherit"
9 app/cache app/logs
```

**2. chmod +a дэмждэггүй систем дээр ACL ашиглах**

Зарим системүүд chmod +a –г дэмждэггүй боловч **setfacl** хэмээх хэрэгсэл ашиглаж болно Ингэхдээ эхлээд өөрийн хард диск дээрээ ACL support[21]-ийг идэвхтэй болгох бөгөөд setfacl-ийг суулгах хэрэгтэй. Энэ нь бас вэб сервер хэрэглэгчийг тодорхойлоход ашиглагддаг команд бөгөөд HTTPDUSER болгон тохируулна.

```
1 $ HTTPDUSER=`ps aux | grep -E '[a]pache|[h]ttpd|[_]www|[w]ww-
  data|[n]ginx' | grep -v
2 root | head -1 | cut -d\ -f1`
3 $ sudo setfacl -R -m u:"$HTTPDUSER":rwX -m u:`whoami`:rwX app/cache
  app/logs
4 $ sudo setfacl -dR -m u:"$HTTPDUSER":rwX -m u:`whoami`:rwX app/cache
  app/logs
```

Хэрэв энэ команд ажиллахгүй бол –n сонголтыг нэмээд үзээрэй.

**3. ACL ашиглахгүйгээр permission зааx**

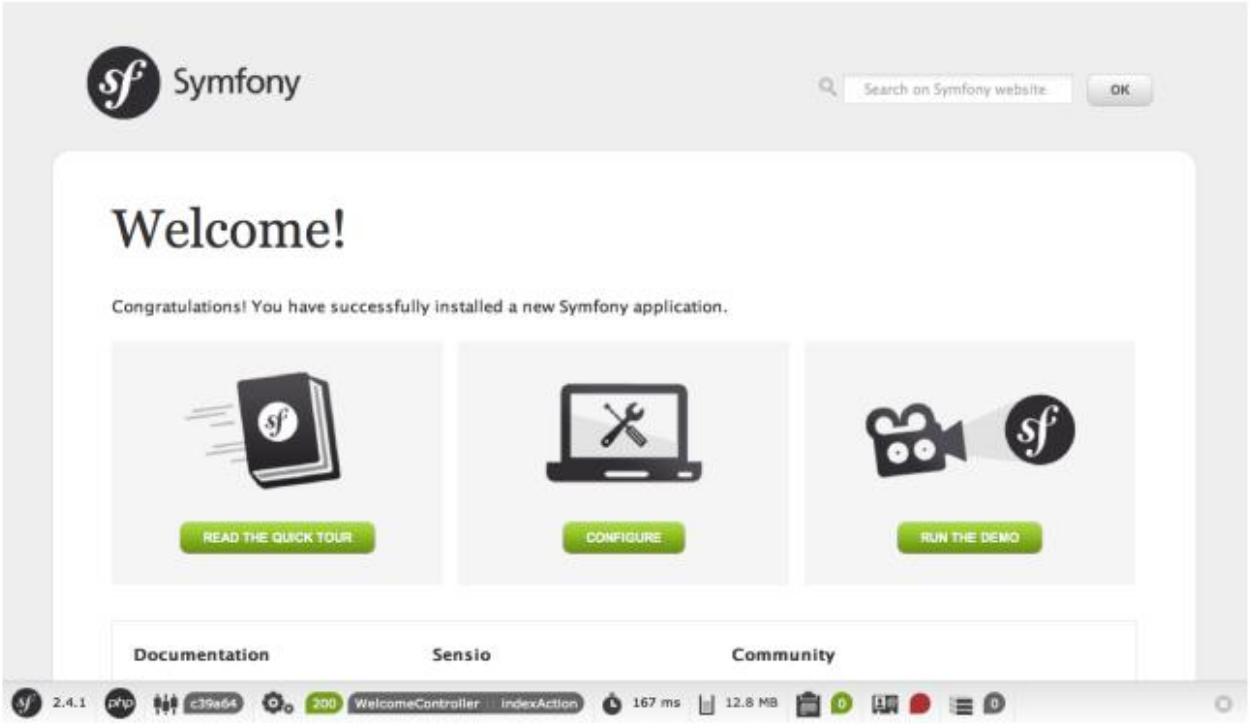
[21] https://help.ubuntu.com/community/FilePermissionsACLs

Хэрэв та хавтасны ACL-ийг өөрчлөх боломжгүй бол cache болон log хавтасуудыг group-writable эсвэл world-writable болгохдоо umask –ийг өөрчлөх хэрэгтэй. Энэ үйлдлийг хийхдээ **app/console**, **web/app.php** болон **web/app_dev.php** файлуудын эхний мөрөнд дараах мөрийг нэмэх хэрэгтэй.

```
umask(0002); // permission-ийг 0775 болгоно.

// эсвэл

umask(0000); // permission-ийг 0777 болгоно.
```

Бүх зүйл амжилттай болсон бол "**Go to the Welcome page**" дээр дарж, Symfony2 вэб хуудасыг дуудна.

```
http://localhost/app_dev.php/
```




## Хөгжүүлэлтийг эхлэх

Ингээд та одоо програмаа хөгжүүлж эхлэхэд бэлэн боллоо. Таны татан авсан Symfony тархац нь дотроо судлах боломжтой зарим жишээ кодыг агуулж байдаг ба үүнийг тухайн тархацад ямар жишээ код байгаа тухай мэдээллийг **README.md** файлаас үзээрээ.

Хэрэв та өөрийн тархацаас жишээ кодыг устгах бол энэ номны “AcmeDemoBundle –ийг хэрхэн утгах вэ?” гэсэн сэдэвийг уншаарай.

4-р бүлэг

# Symfony2-т хуудас үүсгэх

Symfony2 –т шинэ хуудасыг дараах 2 алхамаар үүсгэнэ.

- **Route үүсгэх:** Route нь хуудасны URL –ийг тодорхойлох ба controller – ийг зааж өгнө.
- **Controller үүсгэх:** Controller нь орж ирж байгаа Request-ийг хүлээн авч, Response обект болгон хэрэглэгч рүү буцаадаг PHP функц юм.

Энэ бол их амархан арга бөгөөд ямар ч вэб програм ажиллах үндсэн аргатай нийцэж байгаа юм. Вэб програм дахь харилцаа бүр HTTP request-ээр эхэлдэг билээ. Бидний дээр хийсэн жишээ програм request-ийг хөрвүүлэн тохирох HTTP response болгон буцааж байсан.

## Environment /орчин/ & front controller

Symfony програм бүр тухайн нэг environment буюу орчин дотор ажилладаг. Environment нь тохиргоонууд, ачаалагдсан bundle-ууд, string –үүд юм. Symfony2 нь **–dev, -test, -prod** гэсэн 3 төрлийн Environment –той байна. Гэхдээ хэрэглэгч өөрийн Environment –ийг бас үүсгэх боломжтой байдаг.

Symfony2-т вэб рүү хандах боломжтой 2 өөр төрлийн Front controller байдаг. **app_dev.php** нь **dev** environment-ийг, **app.php** нь prod environment-ийг бий болгоно. Symfony2-т вэбийн бүх хандалтууд энэ 2 controller-ын аль нэгээр дамжин явагдана. (**Test** environment зөвхөн unit test хийх үед ашиглагдах бөгөөд front controller-т зориулагдаагүй юм. Мөн console хэрэгсэл нь ямар ч environment –д ашиглагдаж болдог front controller-ийг агуулж байдаг.)

Front controller нь kernel-ийг эхлүүлэхдээ environment болон kernel нь debug горимд ажиллах ёстой эсэхийг шийдэх 2 параметерийг агуулдаг. Програмыг хурдан ажиллагаатай болгохын тулд, Symfony2 нь **app/cache/** хавтсан дахь кэйшийг сайжруулж байдаг. Debug горим идэвхтэй байгаа үед та ямар нэг код эсвэл тохиргоог өөрчлөхөд энэ кэйшийг автоматаар цэвэрлэж байдаг. Энэ горимд ажиллаж байгаа үед Symfony2 нь удаан ажилладаг боловч кэйшийг өөрийн гараар цэвэрлэхгүй байх давуу талтай.

### “Hello Symfony” хуудас

Одоо энд "Hello World!" програмыг бүтээе. Үүнийг хийж дууссаны дараа, хэрэглэгч дараах URL-ийн дагуу өөрийн мэндчилгээг гаргах боломжтой.

```
1    http://localhost/app_dev.php/hello/Symfony
```

Symfony гэдэгийн оронд ямар ч нэр байж болно.

## Эхлэхийн өмнө: Bundle үүсгэх

Та эхлэхийн өмнө bundle үүсгэх хэрэгтэй. Symfony2-т bundle-ийг plugin –тай адил гэж ойлгож болно.

Bundle бол PHP классууд, тохиргооны файлууд, stylesheet, javascript файлуудыг агуулж байдаг энгийн л хавтас юм.

За ингээд одоо **AcmeHelloBundle** нэртэй bundle үүсгэе. Ингэхийн тулд дараах командыг ажиллуулна.

```
1  $ php app/console generate:bundle --namespace=Acme/HelloBundle --format=yml
```

Игэснээр **src/Acme/HelloBundle** гэсэн bundle хавтас үүснэ. Мөн **app/AppKernel.php** файлд нэг мөр код автоматаар нэмэгдэнэ.  Энэ мөр код нь bundle –г кернелд бүртгэж байгаа хэрэг юм.

```
1  // app/AppKernel.php
2  public function registerBundles()
3  {
4      $bundles = array(
5          ...,
6          new Acme\HelloBundle\AcmeHelloBundle(),
7      );
8      // ...
9
10     return $bundles;
11 }
```

Ингээд bundle –ийг үүсгэлээ. Одоо bundle дотороо програмаа бүтээхэд бэлэн боллоо.

## 1-р алхам: Route үүсгэх

Symfony2-ийн Routing тохиргооны файл нь **app/config/routing.yml** файлд хадгалагдаж байдаг. Та бусад тохиргооны файлуудтай адил  routing тохиргоонд XML, PHP –ийн алийг ч ашиглаж болно.

Хэрэв та үндсэн routing тохиргооны файлыг нээж үзвэл сая AcmeHelloBundle –ийг үүсгэх үед түүнийг бүртгэн авсан байх болно.

```
1  # app/config/routing.yml
2  acme_hello:
3      resource: "@AcmeHelloBundle/Resources/config/routing.yml"
4      prefix: /
```

Энэ нь **AcmeHelloBundle** –д байгаа **Resources/config/routing.yml** файлаас routing тохиргооны файлыг уншина гэдгийг зааж байгаа ба **app/config/routing.yml** –д шууд routing тохиргоог байрлуулж байгаа юм. Эндээс програмынхаа тохиргоонуудыг дуудна.

Одоо bundle-аас дуудагдаж байгаа **routing.yml** файлдаа шинээр үүсгэх гэж байгаа хуудасныхаа URL-ийг тодорхойлох шинэ route нэмэе.

```yaml
# src/Acme/HelloBundle/Resources/config/routing.yml
hello:
    path: /hello/{name}
    defaults: { _controller: AcmeHelloBundle:Hello:index }
```

Routing нь үндсэн 2 хэсгээс бүтнэ.

- **Path:** Route –г заасан зам.
- **Defaults:** Ажиллуулах ёстой controller

Path-д байгаа placeholder ({name}) –т орлуулах утга ирэх бөгөөд энэ утга нь controller луу дамжигдана. Жишээ нь: /hello/Ryan, /hello/Fabien гэх мэт.

## 2-р алхам: Controller үүсгэх

**/hello/Ryan** гэх мэт URL-ийг програмд дамжуулах үед **hello** гэсэн route тохирч, AcmeHelloBundle:Hello:index гэсэн controller биелэгдэнэ. Хуудас үүсгэх дараагийн алхам бол controller үүсгэх байна.

**AcmeHelloBundle:Hello:index** бол Controller-ийн логик нэр бөгөөд **Acme\HelloBundle\Controller\HelloController** хэмээх PHP класын **indexAction** функц рүү чиглүүлнэ.

Ингээд controller-оо үүсгэе. Ингэхдээ эхлээд, **AcmeHelloBundle** дотороо дараах файлыг үүсгэнэ.

```php
// src/Acme/HelloBundle/Controller/HelloController.php
namespace Acme\HelloBundle\Controller;
class HelloController
{
}
```

Чухамдаа controller бол таны үүсгэсэн энгийн PHP функц юм. Энэ нь request-ээс ирж байгаа мэдээллийг ашиглан хүссэн мэдээллийг нь бэлтгэж өгдөг.

**Hello** route тохирох үед ажиллах indexAction функцийг үүсгэе.

```php
// src/Acme/HelloBundle/Controller/HelloController.php
namespace Acme\HelloBundle\Controller;

use Symfony\Component\HttpFoundation\Response;

class HelloController
{
    public function indexAction($name)
    {
        return new Response('<html><body>Сайн уу '.$name.'!</body></html>');
    }
}
```

Дээрх controller шинээр **Response** обект (энэ жишээнд жижигхэн HTML хуудас) үүсгэж байна. За ингээд Route болон Controller үүсгэснээр та бүрэн ажиллагаатай хуудастай боллоо. Хэрэв та бүгдийг зөв хийсэн бол програм тань тантай мэндчилэх болно.

```
1  http://localhost/app_dev.php/hello/Ryan
```

Мөн та програмаа prod enveronment дотор харж болно.

```
1  http://localhost/app.php/hello/Ryan
```

Хэрэв ямар нэгэн алдаа гарвал Cache-ээ цэвэрлэх хэрэгтэй.

```
1  $ php app/console cache:clear --env=prod --no-debug
```

## Нэмэлт алхам 3: Темплэйт үүсгэх

Темплэйт нь вэбийн бүх харагдах хэсгийг (HTML код) тусдаа нэг файлд хийх боломжыг олгох ба хуудасны ерөнхий бүтцийн өөр өөр хэсгүүдийг дахин ашиглах боломж олгоно. Controller дотор HTML код бичихийн оронд темплэйт ашиглана.

```
1  // src/Acme/HelloBundle/Controller/HelloController.php
2  namespace Acme\HelloBundle\Controller;
3
4  use Symfony\Bundle\FrameworkBundle\Controller\Controller;
5
6  class HelloController extends Controller
7  {
8      public function indexAction($name)
9      {
10         return $this->render(
11             'AcmeHelloBundle:Hello:index.html.twig',
12             array('name' => $name)
13         );
14         // render a PHP template instead
15         // return $this->render(
16         // 'AcmeHelloBundle:Hello:index.html.php',
17         // array('name' => $name)
18         // );
19     }
20 }
```

**Render()** method нь темплэйтэд өгөгдсөн агуулгыг багтаасан Response обектийг үүсгэнэ. Энд темплэйтийг үзүүлэх 2 төрлийн жишээ байна. Symfony2 уламжлалт **PHP** болон **Twig** гэсэн 2 өөр төрлийн темплэйтийг ашиглах боломжтой. Гэсэн ч та нэг прожект дотор хоёуланг нь ашигласан ч болно, эсвэл нэгийг нь ашигласан ч болно.

Controller дараах загварын дагуу **AcmeHelloBundle:Hello:index.html.twig** темплэйтийг үзүүлнэ.

> **BundleName:ControllerName:TemplateName**

Энэ бол темплэйтийн логик нэр бөгөөд дараах зарчмаар физик зам руу нь чиглүүлдэг.

> **/path/to/BundleName/Resources/views/ControllerName/TemplateName**

Дээрх тохиолдолд **AcmeHelloBundle** гэдэг нь bundle-ийн нэр, **Hello** нь controller бөгөөд **index.html.twig** нь темплэйт юм.

```twig
{# src/Acme/HelloBundle/Resources/views/Hello/index.html.twig #}
{% extends '::base.html.twig' %}

{% block body %}
    Сайн уу {{ name }}!
{% endblock %}
```

Дээрх темплэйтийн кодыг тайлбарлая.

- 2-р мөр: **extends** гэсэн түлхүүр үгээр эцэг темплэйтийг тодорхойлж өгнө.
- 4-р мөр: **body** гэсэн блокын дотор бүх агуулга байрлана гэдэгийг зааж байна. Мөн эцэг темплэйт (**base.html.twig**) нь body блокыг уншина.

Эцэг темплэйт болох **::base.html.twig** -ээс **BundleName** болон **ControllerName** -ийг хассан байна (давхар :: тэмдгээр эхэлж байна). Үүний учир нь тухайн темплэйт нь bundle-ийн гадна талд **app** хавтсан дотор байгааг илэрийлж байгаа юм.

```twig
{# app/Resources/views/base.html.twig #}
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>{% block title %}Welcome!{% endblock %}</title>
        {% block stylesheets %}{% endblock %}
        <link rel="shortcut icon" href="{{ asset('favicon.ico') }}" />
    </head>
    <body>
        {% block body %}{% endblock %}
        {% block javascripts %}{% endblock %}
    </body>
</html>
```

Үндсэн темплэйтийн файл нь HTML хуудасны ерөнхий зохион байгуулалтыг тодорхойлох бөгөөд **index.html.twig** –т тодорхойлсон **body** блокыг уншин гаргана. Иймд хүүхэд темплэйтэд **title** блокыг тодорхойлж өгөөгүй болно. Энэ нь анхандаа “Welcome” гэсэн title-тай байдаг.

Темплэйт нь хуудасны агуулгыг зохион байгуулж, харуулах маш сайн арга бөгөөд энэ нь HTML, CSS кодоос гадна controller-ийн бүцааж болох бүх л зүйлсийг харуулах боломжтой.

### Хавтасны бүтэц, зохион байгуулалт

Ингээд та Symfony2-т хуудсыг хэрхэн үүсгэж, харуулах талаар ойлголттой болсон байх. Дараах бүлэгт Symfony програмын бүтцийн талаар судлах болно.

Symfony програм нь ерөнхийдөө дараах бүтэц, зохион байгаалалттай байдаг.

- **app/:** Энэ хавтсанд програмын тохиргооны файлууд байдаг.
- **src/:** Прожектийн бүх PHP код энэ хавтсанд хадгалагдана.
- **vendor/:** vendor сангууд энд байрлана.
- **web/:** Энэ бол вэбийн үндсэн /root/ хавтас бөгөөд public хандалттай файлуудыг агуулна.

## “Web” хавтас

Вэбийн үндсэн хавтас болох “Web”-д зурган файлууд, хуудасны хэлбэр тодорхойлох файлууд /css, less гэх мэт/ болон javascript файлууд зэрэг public хандалттай болон статик файлуудыг агуулахаас гадна мөн энд front controller байрлана.

```
// web/app.php
require_once __DIR__.'/../app/bootstrap.php.cache';
require_once __DIR__.'/../app/AppKernel.php';

use Symfony\Component\HttpFoundation\Request;

$kernel = new AppKernel('prod', false);
$kernel->loadClassCache();
$kernel->handle(Request::createFromGlobals())->send();
```

Front controller (энэ жишээнд **app.php**) бол PHP файл бөгөөд Symfony2 програм ажиллах үед биелэгддэг ба энэ нь Kernel клас болох **AppKernel –**ийг ашиглан програмыг ажиллагаанд оруулна.

front controller нь энгийн PHP ашигласнаас илүү URL-ийг уян хатан болгоно. Front controller ашиглаж байгаа тохиолдолд URL нь дараах хэлбэртэй байна.

```
http://localhost/app.php/hello/Ryan
```

Front controller болох **app.php** нь routing тохиргоог ашиглан URL болох /hello/Ryan-ийг чиглүүлнэ. Apache mod_rewrite дүрмийг ашигласнаар тухайн URL-д app.php –ийг зааж өгөлгүйгээр ажиллуулах боломжтой.

```
http://localhost/hello/Ryan
```

## Application (app) хавтас

**AppKernel** класс бол програмын гол хэсэг бөгөөд бүх тохиргооны файлуудтай ажиллах үүрэгтэй. Энэ нь **app/** хавтсанд байрлана.

Энэ класс нь таны програмын тухай бүх мэдээллийг Symfony-д зааж өгөхөд шаардлагатай 2 төрлийн method-ийг хангаж өгдөг.

- **registerBundles():** Програм ажиллахад шаардагдах бүх bundle-уудыг агуулсан массивийг буцаана.
- **registerContainerConfiguration():** Програмын үндсэн тохиргооны файлыг дуудна.

Програм хөгжүүлж байх явцад **app/** хавтас руу байнга хандан **app/config/** хавтсан дахь тохиргооны болон routing файлуудыг засах хэрэгтэй болно. Мөн програмын кэйшийг агуулдаг **app/cache** хавтас, лог файлыг агуулж байдаг **app/logs** хавтас, темплэйт зэрэг файлуудыг агуулах **app/Resources** хавтас зэргийг **app/** хавтас агуулдаг.

### Autoloading

Symfony-г эхлүүлэх үед vendor/autoload.php гэх тусгай файл орж ирсэн байна. Энэ файлыг Composer үүсгэсэн бөгөөд **src/** хавтсан дахь програмын бүх файлууд, мөн composer.json файлд заасан third-party сангуудыг автоматаар дуудна.

Ингэснээр та include эсвэл require –ийг ашиглах шаардлагагүй болно. Оронд нь Composer классын namespace-ийг ашиглан танд шаардлагатэй классыг таны өмнөөс автоматаар оруулж ирэх болно.

Autoloader нь src/ хавтсанд байх таны PHP классуудыг аль хэдийн бүртгээд авчихсан байдаг. Энэ нь дараах загварын дагуу классын нэр болон замыг авч ажиллана.

```
Class Name:
    Acme\HelloBundle\Controller\HelloController
Path:
    src/Acme/HelloBundle/Controller/HelloController.php
```

## Source (src) хавтас

**src/** хавтас нь таны програмд ашиглагдах жинхэнэ код (PHP классууд, Темплэйтүүд, Stylesheet гэх мэт)-уудыг агуулна. Програм хөгжүүлж байх явцад, энэ хавтсанд нэг эсвэл хэд хэдэн bundle үүсгэн ажиллах болно.

Ингэхэд Bundle гэж яг юу вэ?

### Bundle систем

Bundle гэдэг бол бусад програмд байх plugin –тэй ижил зүйл юм. Гэхдээ plugin-аас хамаагүй илүү. Гол ялгаа нь гэвэл Symfony2 дахь **bundle** нь фрэймворкын цөм хэсэгүүд болон таны бичсэн бүх кодуудыг агуулж байдаг систем юм.

Symfony2 дахь хамгийн анхны класс бол **bundle**. Энэ нь урьдчилан бэлтгэсэн third-party bundle, эсвэл өөрийн bundle-ийг үүсгэн ашиглах боломжыг танд олгоно.

Bundle нь нэг л үйлдлийг биелүүлэхээр зохион байгуулагдсан байна. Жишээ нь **BlogBundle**, **ForumBundle**, хэрэглэгчийг удирдах зэрэг bundle-уудыг үүсгэж болно. Ийм төрлийн олон bundle –ийг нээлттэйгээр хөгжүүлж байдаг тул та өөрт тохирохыг сонгон авч ажиллах боломж бас бий. **Bundle** хавтас болгон тухайн нэг зорилгыг биелүүлэхэд ашиглагдах PHP файлууд, темплэйтүүд, stylesheet, javascript, Test гэх зэрэг бүх л зүйл байна.

Програмд хэрэглэгдэх бүх bundle-уудыг **AppKernel** классын **registerBundles()** method –д тодорхойлсон байдаг.

```
// app/AppKernel.php
public function registerBundles()
{
    $bundles = array(
      new Symfony\Bundle\FrameworkBundle\FrameworkBundle(),
      new Symfony\Bundle\SecurityBundle\SecurityBundle(),
      new Symfony\Bundle\TwigBundle\TwigBundle(),
      new Symfony\Bundle\MonologBundle\MonologBundle(),
      new Symfony\Bundle\SwiftmailerBundle\SwiftmailerBundle(),
      new Symfony\Bundle\DoctrineBundle\DoctrineBundle(),
```

```
            new Symfony\Bundle\AsseticBundle\AsseticBundle(),
            new Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtraBundle(),
        );
    if (in_array($this->getEnvironment(), array('dev', 'test'))) {
        $bundles[] = new Acme\DemoBundle\AcmeDemoBundle();
        $bundles[] = new Symfony\Bundle\WebProfilerBundle\WebProfilerBundle();
        $bundles[] = new
    Sensio\Bundle\DistributionBundle\SensioDistributionBundle();
        $bundles[] = new Sensio\Bundle\GeneratorBundle\SensioGeneratorBundle();
     }

     return $bundles;
    }
```

## Bundle үүсгэх

**Bundle** системийг хэрхэн энгийн аргаар үүсгэхийг харуулах зорилгоор **AcmeTestBundle** хэмээх bundle үүсгэж, түүнийгээ идэвхжүүлэе.

**Acme** гэдэг бол зүгээр л жишээ нэр юм. Үүний оронд та өөрөө нэр өгч болно.

Эхлээд **src/Acme/TestBundle/** хавтасыг үүсгээд түүндээ **AcmeTestBundle.php** нэртэй шинэ файл нэмнэ.

```
// src/Acme/TestBundle/AcmeTestBundle.php
namespace Acme\TestBundle;

use Symfony\Component\HttpKernel\Bundle\Bundle;

class AcmeTestBundle extends Bundle
{
}
```

Энэ хоосон класс шинэ bundle үүсгэхэд л хэрэглэгдэнэ. Хэдийгээр хоосон класс байгаа ч гэсэн bundle – ийг зохион байгуулахад хэрэглэгдэнэ.

Ингээд bundle-аа үүсгэсэн болохоор түүнийгээ **AppKernel** классд бүртгүүлэнэ.

```
// app/AppKernel.php
public function registerBundles()
{
    $bundles = array(
        ...,
        // таны үүсгэсэн bundle-ийг бүртгэнэ
        new Acme\TestBundle\AcmeTestBundle(),
    );
    // ...

    return $bundles;
}
```

Энэ нь одоогоор ямар нэг үйлдэл хийхгүй ч **AcmeTestBundle** ашиглахад бэлэн боллоо.

Symfony нь өөртөө bundle-ийн үндсэн араг ясыг үүсгэх command-line интерфэйсийг агуулдаг.

```
$ php app/console generate:bundle --namespace=Acme/TestBundle
```

Bundle-ийн араг яс үүсгэхэд тухайн bundle дотор түүний үндсэн controller, темплэйт, routing файлууд хамт үүснэ.

Шинээр bundle үүсгэх, эсвэл third-party bundle ашиглахдаа тухайн bundle-ийг **registerBundles()** –т бүртгүүлэх хэрэгтэй. Харин **generate:bundle** комманд ашиглаж байгаа үед бол энэ үйлдэл автоматаар хийгдэнэ.

## Bundle хавтасны бүтэц

Bundle хавтасны бүтэц нь энгийн, үян хатан байдаг. Доор bundle-ийн ерөнхий хэрэглэгддэг элементүүдийг агуулсан жишээ болгон **AcmeHelloBundle** –ийг харууллаа.

- **Controller/** хавтас нь bundle-д ашиглагдах controller-уудыг агуулна. (жнь: HelloController.php)
- **Resources/config/** нь routing тохиргоо зэрэг тохиргооны файлуудыг агуулна.
- **Resources/views/** нь controller-ийн нэрийн дагуу үүсгэгдсэн темплэйтийг агуулна. (жнь: Hello/index.html.twig)
- **Resources/public/** нь public хандалттай вэбийн файлуудыг (зураг, stylesheet гэх мэт) агуулна.
- **Tests/** нь bundle-ийг шалгах файлуудыг агуулна.

Bundle нь гүйцэтгэж байгаа үйлдлээсээ хамааралд жижиг ч байж болно, том ч байж болно. Энэ нь зөвхөн танд хэрэгцээтэй файлуудыг л агуулна.

Энэ номыг уншсанаар та өгөгдлийн санд мэдээлэл хадгалах, бүцаан унших, формын мэдээллийг баталгаажуулах, тест хийх гэх зэрэг олон үйлдлийг гүйцэтгэх болно. Энэ бүгд тухайн bundle дотороо л хийгдэнэ.

### Програмын тохиргоо

Аливаа нэг програм тодорхой үйлдлийг биелүүлэх үүрэг бүхий bundle-уудын цуглуулгуудаас бүрдэнэ. Bundle бүр YAML, XML, эсвэл PHP форматаар бичигдсэн тохиргооны файлаар тохируулагдсан байна. Default-аар бол програмын үндсэн тохиргооны файл нь **app/config/** хавтсанд байрлаж байдаг.

```
# app/config/config.yml
imports:
    - { resource: parameters.yml }
    - { resource: security.yml }
framework:
    secret: "%secret%"
    router: { resource: "%kernel.root_dir%/config/routing.yml" }
    # ...
# Twig Configuration
twig:
    debug:                "%kernel.debug%"
    strict_variables:     "%kernel.debug%"
# ...
```

Тохиргоог YAML, XML, PHP гэсэн 3 төрлийн файлаар үүсгэж болно. Эдгээр нь өөрийн давуу болон сул талтай.

- YAML: Энгийн, ойлгомжтой, цэвэр
- XML: Зарим тохиолдолд YAML-аас илүү хүчирхэг
- PHP: Маш хүчирхэг боловч, уншихад төвөгтэй.

## Default буюу анхны тохиргоог салгах

YAML файлд тодорхойлсон bundle-ийн default тохиргоог салгах боломжтой. Ингэхийн тулд **config:dumpreference** командыг ашиглана. Энд FrameworkBundle –ийн default тохиргоог салгах жишээг харууллаа.

```
1  $ app/console config:dump-reference FrameworkBundle
```

### Environment буюу орчин

Програм олон төрлийн environment-д ажиллах боломжтой байдаг. Өөр өөр environment нь адил PHP кодуудыг ашиглах боловч өөр өөр тохиргоог ашиглана. Жишээ нь, **dev** environment нь анхааруулга болон алдааны тухай мэдээллийг бүртгэж байдаг бол **prod** environment нь зөвхөн алдааны тухай мэдээллийг бүртгэнэ. Зарим файлууд **dev** environment –д дахин үүсдэг ч, **prod** environment –д бол нөөцлөгдсөн байна. Бүх environment –ууд нэг машинд байрлаж, нэг програмд ажиллана.

Symfony2 прожект ерөнхийдөө 3 төрлийн (dev, test, prod) environment –д ажиллана. Мөн түүнчлэн шинэ environment үүсгэх боломжтой. Програмаа эдгээр environment –уудад ажиллуулж үзэхийн тулд вэб хөтчид front controller-оо өөрчилнө. Тухайлбал **dev** environment –д харах бол:

```
1  http://localhost/app_dev.php/hello/Ryan
```

Хэрэв програмаа production environment-д харахыг хүсвэл

```
1  http://localhost/app.php/hello/Ryan
```

**prod** environment-д тохиргоо, routing, Twig темплэйтүүд нь PHP класс руу хөрвүүлэгдэн, кэйшлэгдсэн байдаг. prod environment-д хийгдсэн өөрчлөлтийг харахын тулд эдгээр кэйш файлуудыг цэвэрлэж, тэдгээрийг файлуудыг дахин үүсгэх хэрэгтэй.

```
1  $ php app/console cache:clear --env=prod --no-debug
```

**web/app.php** файлыг нээж үзвэл prod env –ийн тохиргоог харах болно.

```
1    $kernel = new AppKernel('prod', false);
```

Хэрэв та шинэ env-д үүсгэн түүндээ шинэ front controller үүсгэх бол энэ файлыг тухайн env-доо хуулаад, **prod** гэсэн нэрийг өөрийн хүссэн нэрээрээ солих хэрэгтэй.

Listing

Автомат тест ажиллаж байгаа үед **test** environment-ийг ашиглах бөгөөд түүн рүү вэб хөтчөөр шууд хандах боломжгүй.

## Environment тохиргоо

**AppKernel** класс нь таны сонгосон форматтай тохиргооны файлыг ашиглан ажиллана.

```php
// app/AppKernel.php
public function registerContainerConfiguration(LoaderInterface $loader)
{
    $loader->load(
        __DIR__.'/config/config_'.$this->getEnvironment().'.yml'
    );
}
```

Та **.yml** өргөтгөлийг мэддэг болсон бөгөөд **.xml** эсвэл **.php** –ийн алийг ч ашиглах боломжтой. Environment бүр нь өөрт зориулсан тохиргооны файлыг л ачааллана. Энд **dev** environment-д зориулсан тохиргооны файлыг харууллаа.

```yaml
# app/config/config_dev.yml
imports:
    - { resource: config.yml }

framework:
    router: { resource: "%kernel.root_dir%/config/routing_dev.yml" }
    profiler: { only_exceptions: false }
# ...
```

**Imports** түлхүүр үг нь PHP include оператортай адилхан бөгөөд үндсэн тохиргооны файлыг (config.yml) эхлээд дуудна гэдэгийг зааж байгаа юм.

5-р бүлэг

# Controller

Controller бол HTTP request-ийг хүлээн авч, HTTP response (Symfony2 Response обект) буцаадаг PHP функц юм. Response нь HTML хуудас, XML, serialized JSON массив, зураг, redirect, 404 алдаа, зэрэг ямар ч зүйл байж болно. Дараах controller –ийн жишээнд **Hello World** гэсэн үгийг хэвлэн харуулах хуудсыг харууллаа.

```
1  use Symfony\Component\HttpFoundation\Response;
2
3  public function helloAction()
4  {
5      return new Response('Hello world!');
6  }
```

Өөрөөр хэлбэл Controller –ийн үндсэн үүрэг бол Response обект үүсгэн, буцаах юм. Энэ үүргийнхээ дагуу request-ээс мэдээлэл унших, өгөгдлийн сангаас мэдээлэл авах, мэйл илгээх, хэрэглэгчийн session –д мэдээлэл хадгалах гэх мэт олон үйлдлийг гүйцэтгэх боломжтой. Энэ бүх тохиолдолд controller нь клейнт рүү Response обектыг буцаадаг.

Энд хэдэн жишээ controller орууллаа.

- *Controller A* нь сайтын нүүр хуудасны агуулгыг харуулах Response обектийг бэлтгэнэ.
- *Controller B* өгөгдлийн сангаас тухайн блог постыг унших параметерийг request-ээс унших ба тухайн блогыг үзүүлэх Response обектийг үүсгэнэ. Хэрэв өгөгдлийн сангаас тохирох параметер олдоогүй бол 404 статус кодтой Response обект үүсгэн буцаана.
- *Controller C* Холбоо барих формтой ажиллана. Энэ нь Request-ээс формын мэдээллийг уншиж, өгөгдлийн сан руу формын мэдээллийг хадгалж, вэбийн админ руу тухайн мэдээллийг мэйлээр илгээнэ.

## Requests, Controller, Response – амьдралын цикл

Symfony прожектийн request боловсруулах амьдралын цикл нь ерөнхийдөө дараах алхамаар явагдана.

1. Request бүр нэг л front controller (жнь: **app.php** эсвэл **app_dev.php**) – файлаар дамжин биелэгдэнэ.
2. **Router** нь request-ээс мэдээлэл уншиж, тухайн мэдээлэлтэй тохирох route-ийг олоод, тухайн route-ээс **_controller** параметерийг уншина.
3. Controller нь **Response** обектийг үүсгээд, буцаана.
4. HTTP headers болон Response обектийн агуулгыг клейнт рүү буцаана.

## Энгийн Controller-ийн жишээ

Controller нь дуудан ажилуулах боломжтой ямар ч PHP функц байж болох ч, Symfony2 –т controller нь ихэнхдээ controller обект доторх ганц method байдаг. Controller-ийг мөн өөрөөр ***action*** гэж нэрлэдэг.

```
1  // src/Acme/HelloBundle/Controller/HelloController.php
2  namespace Acme\HelloBundle\Controller;
3
4  use Symfony\Component\HttpFoundation\Response;
5
6  class HelloController
7  {
8      public function indexAction($name)
9      {
10         return new Response('<html><body>Сайн уу '.$name.'!</body></html>');
11     }
12 }
```

Controller гэдэг бол *controller* класс (HelloController) дотор байх **indexAction** method гэдэгийг санаарай. Ерөнхийдөө бол controller класс нь хэд хэдэн controller/action (**updateAction**, **deleteAction**, гэх мэт)–уудыг агуулж байдаг.

Дээрх controller маш энгийн харагдаж байна.

- 4-р мөр: **Use** түлхүүр үг нь **Response** класыг оруулж ирнэ.
- 6-р мөр: класын нэр (жнь: Hello)-ийг Controller гэдэг үгтэй нийлүүлсэн байна. Энэ нь controller агуулсан класс гэдэгийг зааж байгаа бөгөөд тохиргооны файлд нэрийн эхний хэсгийг (Hello) л зааж өгөхөд болно.
- 8-р мөр: Controller класс дахь method бүр нь **Action** үгээр төгссөн байх бөгөөд тохиргооны файлд action-ий нэр (index)-ийг заана.
- 10-р мөр: Controller нь Response обектийг үүсгэн буцаана.

## Controller руу URL-ийг чиглүүлэх

Controller нь энгийн HTML хуудас буцааж болно. Вэб хөтчид энэ хуудсыг дуудахын тулд, Controller- руу URL замыг заах route үүсгэх хэрэгтэй.

```
# app/config/routing.yml
hello:
    path: /hello/{name}
    defaults: { _controller: AcmeHelloBundle:Hello:index }
```

Одоо ингээд **/hello/ryan** гэж ажиллуулхад **HelloController::indexAction()** controller биелэгдэн, **$name** хувьсагч руу **ryan** гэсэн утга дамжина. Тэгэхээр “хуудас” үүсгэнэ гэдэг нь ерөнхийдөө Controller үүсгээд, түүнийгээ route-тэй холбох л ажил юм.

Synfony2 өөр өөр controller-руу хандахдаа **String notation/тэмдэглэгээ/** ашигладаг. Энэ арга нь их түгээмэл ашиглагддаг бөгөөд энд **AcmeHelloBundle** нэртэй bundle дотор **HelloController** хэмээх controller класс байна гэдэгийг зааж байгаа хэрэг юм. Тэгээд **indexAction()** method биелэгдэнэ.

### Controller аргумент маягийн Route параметер

**_controller** параметер нь **AcmeHelloBundle** хэмээх bundle –ийн **HelloController::indexAction()** method руу **AcmeHelloBundle:Hello:index** –ийг заадаг. Тухайн method руу ямар аргумент дамжигдаж байгаа нь их чухал байдаг.

```
// src/Acme/HelloBundle/Controller/HelloController.php
namespace Acme\HelloBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class HelloController extends Controller
{
    public function indexAction($name)
    {
        // ...
    }
}
```

Дээрх controller нь **$name** хэмээх нэг аргумент авч байгаа бөгөөд энэ нь route-ээс тохирох {name} параметерийг авна. Энэ жишээнд бол **ryan** байна. Өөрөөр хэлбэл таны controller биелэгдэхэд route-ээс ирж байгаа параметертэй controller-ийн аргументийг харьцуулдаг. Дараах жишээг харая:

```
# app/config/routing.yml
hello:
    path: /hello/{firstName}/{lastName}
    defaults: { _controller: AcmeHelloBundle:Hello:index, color: green }
```

Controller нь хэд хэдэн аргументтэй байж болно.

```
public function indexAction($firstName, $lastName, $color)
{
    // ...
}

```

Дээрх утга авах **{firstName}, {lastName}** хувьсагчууд болон анхнаасаа тодорхойлж өгсөн **color** хувьсагчууд нь controller-ийн авах аргементүүд юм. Тохирсон route орж ирэхэд **{firstName}, {lastName}** хувьсагчууд **color** хувьсагчтай нийлэн нэг массив болж таны controller руу дамжигдана.

Route-ийн параметерүүдийг controller руу чиглүүлэх нь их амархан байдаг.

Та хөгжүүлэлт хийж байхдаа дараах зааврыг санаж байх хэрэгтэй.

- **Controller-ийн аргументүүдийн дараалал чухал биш**
  Symfony-д controller method-ийн хувьсагчийн нэр, route-ийн параметерийн нэртэй адил байж болно. Өөрөөр хэлбэл {**lastName**} параметер нь **$lastName** аргументтэй адил байна. Ингэхэд controller-ийн аргументүүд ямар ч дараалалгүй байсан ч зөв ажилласаар байх болно.
  ```
  1  public function indexAction($lastName, $color, $firstName)
  2  {
  3      // ...
  4  }
  ```

- **Controller-ийн аргументүүд болон route –ийн параметүүдийн тоо тэнцүү байх ёстой.**
  Дараах тохиолдолд **RuntimeException** алдаа заана. Учир нь **foo** параметерийг route-д тодорхойлж өгөөгүй.
  ```
  1  public function indexAction($firstName, $lastName, $color, $foo)
  2  {
  3      // ...
  4  }
  ```

  Нэмэлт аргумент хийж болно. Дараах тохиолдолд алдаа заахгүй.

  ```
  1  public function indexAction($firstName, $lastName, $color, $foo = 'bar')
  2  {
  3      // ...
  4  }
  ```
- **Route-ийн бүх параметерүүд controller-ийн аргументэд байх шаардлагагүй.**
  Жишээ нь, хэрэв таны үүсгэсэн Controller-т lastName шаардлагүй бол үүнийг орхиж болно.

  ```
  1  public function indexAction($firstName, $color)
  2  {
  3      // ...
  4  }
  ```

### Controller аргумент маягийн Request

Мөн та controller-тоо Request обектийг аргумент болгон дамжуулж болно. Ялангуяа формтой ажиллаж байх үед энэ нь маш тохиромжтой байдаг. Жишээ нь:

```
1  use Symfony\Component\HttpFoundation\Request;
2
3  public function updateAction(Request $request)
4  {
5      $form = $this->createForm(...);
6      $form->handleRequest($request);
7  // ...
8  }
```

## Статик хуудас үүсгэх

Та controller ашиглахгүйгээр статик хуудас үүсгэх боломжтой. Ингэхийн тулд зөвхөн route болон темплэйт л хэрэгтэй. Үүнийг “Хэрэглэгчийн үүсгэсэн Controller ашиглахгүйгээр темплэйтийг боловсруулах” гэсэн сэдэвээс үзээрэй.

## Үндсэн Controller класс

Symfony2-ийн үндсэн **Controller**[22] клас нь controller дээр хийгддэг нийтлэг үйлдлүүдийг биелүүлхэд туслалдаг. Энэ класыг та өөрийн controller-тоо оруулж ирсэнээр олон тооны туслах method-уудыг ашиглах боломжтой болно. **Use** түлхүүр үгээр энэ үндсэн Controller –ийг оруулж ирнэ.

```php
// src/Acme/HelloBundle/Controller/HelloController.php
namespace Acme\HelloBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Response;

class HelloController extends Controller
{
    public function indexAction($name)
    {
        return new Response('<html><body>Hello '.$name.'!</body></html>');
    }
}
```

Symfony-д энэ үндсэн классыг заавал оруулж ирэх шаардлагагүй. Өөрөөр хэлбэл та **ContainerAware**[23] –ийг, эсвэл **Symfony\Component\DependencyInjection\ContainerAwareTrait** (хэрэв PHP 5.4 ашиглаж байгаа бол) классыг оруулан ирж болно.

ContainerAwareTrait-ийг Symfony 2.4-т анх танилцуулсан.

## Controller дээр хийгддэг нийтлэг үйлдлүүд

Онолын хувьд controller ямар ч үйлдлийг хийж болох боловч ихэнхдээ controller-ууд нэг ижил үйлдлүүдийг дахин дахин хийдэг. Эдгээрт redirecting буюу үйлдлийг өөр хуудас руу шилжүүлэх, forwarding, темплэйт боловсруулах, цөм сервес –үүд рүү хандах зэрэг орох бөгөөд эдгээрийг Symfony-д хялбархан зохион байгуулж болно.

[22] http://api.symfony.com/master/Symfony/Bundle/FrameworkBundle/Controller/Controller.html
[23] http://api.symfony.com/master/Symfony/Component/DependencyInjection/ContainerAware.html

### Redirecting

Хэрэв та хэрэглэгчийг өөр хуудас руу шилжүүлэх бол **redirect()** method ашиглана.

```
public function indexAction()
{
    return $this->redirect($this->generateUrl('homepage'));
}
```

**generateUrl()** method нь route-т өгсөн URL-ийг үүсгэдэг функц юм.

Default-аар **redirect()** method нь 302 redirect-ийг биелүүлдэг. 301 redirect-ийг биелүүлэхдээ 2 дахь аргументийг ашиглана.

```
public function indexAction()
{
    return $this->redirect($this->generateUrl('homepage'), 301);
}
```

## Forwarding

Мөн та **forward()** method ашиглан өөр controller руу шилжиж болно. Хэрэглэгчийг өөр хуудас руу чиглүүлэхийн оронд өөр нэг request-ийг биелүүлж, заасан controller-ийг дуудна.

```
public function indexAction($name)
{
    $response = $this->forward('AcmeHelloBundle:Hello:fancy', array(
        'name' => $name,
        'color' => 'green',
    ));
    // ... response-ийг өөрчлөх эсвэл түүнийг шууд буцаана

return $response;
}
```

Энэ **forward()** method нь routing тохиргоонд ашигласан controller-ийн string зааврыг ашигладаг. Дээрх тохиолдолд зорьсон controller нь AcmeHelloBundle доторх HelloController байна. Массив нь controller—ийн method руу аргумент болж очино. Зорьж байгаа controller нь дараах байдалтай байж болно.

```
public function fancyAction($name, $color)
{
    // ... Response обект үүсгэн буцаана
}
```

### Темплэйт боловсруулах

**renderView()** method ашиглан темплэйтийг боловсруулж, түүнийгээ буцаана. Темплэйтийн агуулгыг Response обект үүсгэхэд ашиглаж болно.

```
use Symfony\Component\HttpFoundation\Response;
$content = $this->renderView(
    'AcmeHelloBundle:Hello:index.html.twig',
    array('name' => $name)
```

```
);
return new Response($content);
```

Үүнийг render() method ашиглан нэг л алхамаар хийж болно.

```
return $this->render(
    'AcmeHelloBundle:Hello:index.html.twig',
    array('name' => $name)
);
```

Дээрх 2 тохиолдолд **AcmeHelloBundle** дахь **Resources/views/Hello/index.html.twig** тэмплэйт уншигдах болно.

### Бусад сервес рүү хандах

Үндсэн controller класыг оруулж ирэн, Symfony2 –ийн аль нэг сервес рүү **get()** method-оор хандаж болно. Энд танд хэрэг болж магадгүй хэдэн сервесийг харууллаа.

```
$templating = $this->get('templating');

$router = $this->get('router');

$mailer = $this->get('mailer');
```

Энд мөн бусад сервесүүд байх бөгөөд та ч бас өөрийн сервесийг тодорхойлж болно. Доорх командаар бүх сервесийн жагсаалтыг харах боломжтой.

```
$ php app/console container:debug
```

## Алдааны тухай мэдээлэл болон 404 хуудсыг удирдах

Ямар нэг зүйл олдоогүй тохиолдолд HTTP протокол ашиглан 404 response буцаана. Үүнийг хийхийн тулд тусгай төрлийн **exception** ашиглана. Хэрэв үндсэн controller класыг оруулж ирсэн бол дараах аргаар хийж болно.

```
public function indexAction()
{
    // өгөгдлийн сангаас мэдээлэл авна
    $product = ...;
    if (!$product) {
        throw $this->createNotFoundException('Энэ бүтээгдэхүүн байхгүй
байна');
    }
    return $this->render(...);
}
```

**createNotFoundException()** method нь Symfony дотор HTTP 404 response-ийг үүсгэх **NotFoundHttpException** обектийг үүсгэдэг.

Мэдээж та өөрийн controller-тоо Exception класыг чөлөөтэй ашиглаж болох ба Symfony2 нь 500 HTTP response обектийг буцаана.

```
throw new \Exception('Энд алдааны тухай мессеж байна!');
```

Энэ бүрт алдааны загварчилсан хуудас хэрэглэгчид харагдах бол харин тухайн алдааг илрүүлэх хуудас хөгжүүлэгчид харагдана (debug mode-д хуудсыг харах үед). Дээрх алдааны 2 хуудсыг хөгжүүлэгч өөрөө зохион байгуулах боломжтой.

## Session удирдах

Symfony2-т request-үүдийн хооронд хэрэглэгчийн тухай мэдээллийг хадгалхад ашиглагддаг **Session** обект байдаг ба default-аар Symfony2 нь цэвэр PHP session ашиглан мэдээллийг **cookie**-д хадгалдаг.

Controller ашиглан session-д мэдээлэл хадгалах, мэдээллийг буцаан унших нь маш хялбар байдаг.

```
1  use Symfony\Component\HttpFoundation\Request;
2
3  public function indexAction(Request $request)
4  {
5      $session = $request->getSession();
6
7      // request илгээсний дараа тухайн аттрибутыг дахин ашиглах зорилгоор
   хадгална.
8      $session->set('foo', 'bar');
9
10     // өөр controller-т өөр request-ээр тохируулагдсан аттрибутыг авна.
11     $foobar = $session->get('foobar');
12
13     // хэрэв тухайн аттрибут үүсээгүй байвал default утгыг авна.
14     $filters = $session->get('filters', array());
15 }
```

Эдгээр аттрибутууд хэрэглэгчийн Session-д хадгалагдана.

### Flash мессеж

Хэрэглэгчийн Session-д хадгалагдаж байгаа жижиг хэмжээний мессежийг мөн хадгалах боломжтой. Энэ нь формтой ажиллаж байх үед маш хэрэгтэй байдаг. Ийм төрлийн мессежүүдийг flash мессеж гэж нэрлэдэг.

Жишээ нь формоос мэдээлэл илгээж байна гэж төсөөлөе.

```
1  use Symfony\Component\HttpFoundation\Request;
2  public function updateAction(Request $request)
3  {
4      $form = $this->createForm(...);
5      $form->handleRequest($request);
6      if ($form->isValid()) {
7      // do some sort of processing
8      $this->get('session')->getFlashBag()->add(
9          'notice',
10         'Амжилттай хадгалагдлаа!'
11     );
12     return $this->redirect($this->generateUrl(...));
13     }
14     return $this->render(...);
15 }
```

Request-ийг биелүүлсний дараа controller нь “**notice**” гэсэн flash мессежийг үүсгээд өөр хуудас руу шилжүүлнэ. “Notice” гэсэн нэрийн оронд та өөрийн flash мессеждээ тохирсон нэрийг ашиглаж болно.

Дараа нь темплэйтэдээ дараах кодыг ашиглан notice мессежийг харуулна.

```
{% for flashMessage in app.session.flashbag.get('notice') %}
    <div class="flash-notice">
        {{ flashMessage }}
    </div>
{% endfor %}
```

## Response обект

Controller-ийн үүрэг бол Response обект буцаах. Response бол PHP Abstract класс юм.

```
use Symfony\Component\HttpFoundation\Response;

// 200 статус кодтой энгийн Response үүсгэнэ(default)
$response = new Response('Сайн уу '.$name, Response::HTTP_OK);

// 200 статус кодтой JSON-response
$response = new Response(json_encode(array('name' => $name)));
$response->headers->set('Content-Type', 'application/json');
```

**Headers** property нь Response header-ийг унших, өөрчлөхөд ашиглагдах хэд хэдэн method-уудыг агуулдаг **HeaderBag**[24] юм.

### Request обект

Controller нь Request обект руу ханддаг.

```
use Symfony\Component\HttpFoundation\Request;

public function indexAction(Request $request)
{
    $request->isXmlHttpRequest();

    $request->getPreferredLanguage(array('en', 'fr'));

    $request->query->get('page'); // $_GET параметерийг авна

    $request->request->get('page'); // $_POST параметерийг авна
}
```

Response обекттой адил Request header нь HeaderBag –д хадгалагддаг ба хандахад хялбар байдаг.

[24] http://api.symfony.com/master/Symfony/Component/HttpFoundation/HeaderBag.html

6-р бүлэг

# Routing

Ямар ч вэб програмын хувьд URL энгийн байх нь маш чухал зүйл. Энэ нь **index.php?article_id=57** гэсэн төвөгтэй URL-ийг **/read/intro-to-symfony** гэх мэтээр хялбархан болгоно гэсэн үг юм.

Symfony2 router нь програмын өөр өөр хэсгүүд рүү хандах URL-ийг тодорхойлоход ашиглагдана.

## Action-тай route холбох

Route нь URL замыг controller руу чиглүүлж өгдөг. Тухайлбал: та **/blog/my-post**, **/blog/all-about-symfony** гэх зэргээр URL-ийг зааж, controller руу тухайн URL –аа илгээнэ. Дараах энгийн route жишээг харая.

```
# app/config/routing.yml
blog_show:
    path: /blog/{slug}
    defaults: { _controller: AcmeBlogBundle:Blog:show }
```

**blog_show** гэсэн route-д тодорхойлсон /blog/* -ийн ард байгаа **slug** гэсэн хэсгийг **wildcard** гэж нэрлэнэ. Тухайлбал, **/blog/my-blog-post** гэж өгсөн тохиолдолд slug хувьсагч нь тухайн controller –руу **my-blog-post**-ийн утгыг илгээнэ.

**_controller** параметер бол тусгай түлхүүр үг бөгөөд URL энэ Route –тэй тохирч байгаа тохиолдолд тухайн controller биелэгдэнэ гэдэгийг Symfony-д зааж өгч байгаа юм. Энэ _controller –ийг логик нэр хэмээн нэрлэдэг.

```
// src/Acme/BlogBundle/Controller/BlogController.php
namespace Acme\BlogBundle\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
class BlogController extends Controller
{
    public function showAction($slug)
    {
        // use the $slug variable to query the database
        $blog = ...;
```

```
10          return $this->render('AcmeBlogBundle:Blog:show.html.twig', array(
11              'blog' => $blog,
12          ));
13      }
14  }
```

За ингээд та анхны route-ээ үүсгэн үүнийгээ controller-тойгоо холболоо. Одоо та **/blog/my-post** руу зочилход **showAction** controller биелэгдэх ба **$slug** хувьсагч my-post той тэнцүү байх болно.

Энэ бол Symfony2 router-ийн гол зарчим юм. ӨХ, request-ийн URL-ийг controller руу чиглүүлнэ гэсэн үг.

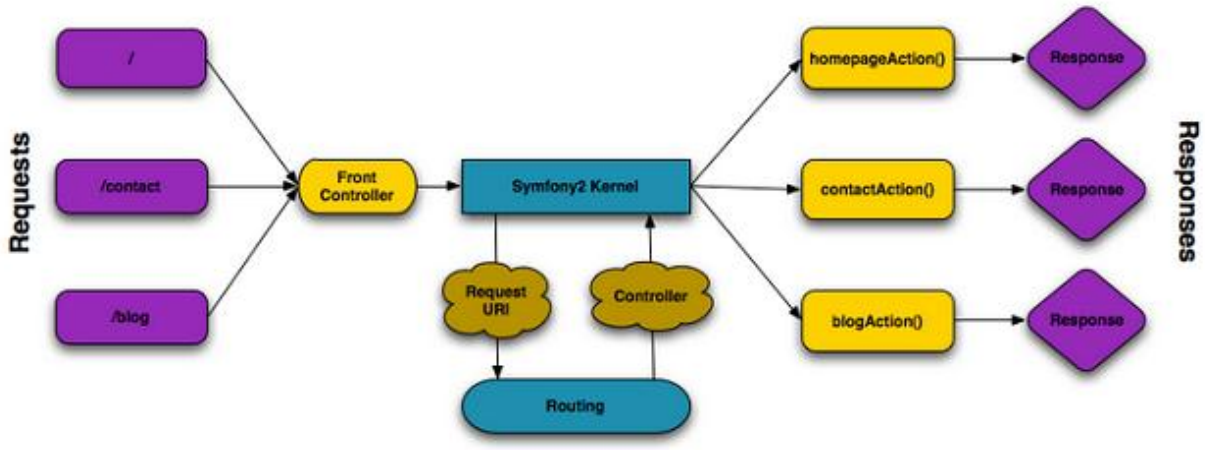
## Routing: Under the Hood

Таны програмдаа үүсгэж байгаа request нь тухайн клейнтийн хүсч байгаа шаардлагатай мэдээллийг авах хаягийг агуулж байдаг. Энэ хаягийг URL (мөн бас URI) хэмээн нэрлэдэг. Дараах HTTP request-ийн жишээг харна уу.

```
1   GET /blog/my-blog-post
```

Symfony2 routing системийн гол зорилго бол URL-ийг хөрвүүлэн аль controller ажиллах ёстойг шийднэ. Route нь дараах процессийн дагуу ажиллана.

1. Front controller (app.php гэх мэт) нь request-ийг хүлээн авна.
2. Symfony2 core (kernel) нь router –т request-ийг дамжуулна.
3. Router нь заасан route-ийг орж ирж байгаа URL-тэй тохируулан үзээд, route-ийн тухай мэдээллийг буцаана.
4. kernel нь тохирох Response обектийг буцаах controller-ийг биелүүлнэ.




Rooting давхарга бол заасан controller руу орж ирж байгаа URL-ийг хөрвүүлдэг хэрэгсэл юм.

## Route үүсгэх

Symfony –д, програмынхаа бүх route-ийг нэг л routing тохиргооны файлаас дууддаг. Энэ файл нь ихэнхдээ **app/config/routing.yml** байдаг ба XML, PHP зэрэг өөр өргөтгөлтэй файлаар бас үүсгэж болно.

```
# app/config/config.yml
framework:
    # ...
    router: { resource: "%kernel.root_dir%/config/routing.yml" }
```

### Үндсэн тохиргооны файл

Route үүсгэх нь маш энгийн ба ихэнх програмууд маш олон route-ээс бүтнэ. Үндсэн route нь path болон default array гэсэн 2 хэсгээс бүрдэнэ.

```
_welcome:
    path: /
    defaults: { _controller: AcmeDemoBundle:Main:homepage }
```

Дээрх route нь нүүр хуудсыг заах ба **AcmeDemoBundle:Main:homepage** гэсэн controller руу зааж байна. **_controller** гэсэн үг нь PHP функц руу хөрвүүлэгдэн биелэнэ. Үүний тухай Controller Naming Pattern хэсэгт тайлбарлах болно.

### Placeholder –той routing

Route нь мөн нэг эсвэл хэд хэдэн placeholder-той хэсгийг агуулж болдог.

```
blog_show:
    path: /blog/{slug}
    defaults: { _controller: AcmeBlogBundle:Blog:show }
```

Энд байгаа **slug** нь таны програмын параметерийн утгыг авна. Өөрөөр хэлбэл: URL нь **/blog/hello-world** байлаа гэж бодвол **$slug** хувьсагч нь **hello-world** утгыг авна. Энэ нь тухайн String (hello-world)-тэй тохирч байгаа блогын бичлэгийг уншина гэсэн үг юм.

### Заавал шаардлагатай болон шаардлагатай бус placeholder

Бидний жишээ болгон авч байгаа блог програмдаа бүх блогын жагсаалтыг харуулдаг нэг route нэмэе.

```
blog:
    path: /blog
    defaults: { _controller: AcmeBlogBundle:Blog:index }
```

Энэ route нь ямар нэг placeholder агуулаагүй ба зөвхөн **/blog** гэсэн URL заасан байна. Харин та route –дээ блогын 2 дахь хуудсыг харуулах хэрэгтэй бол яах вэ? Игэхийн тулд **{page}** гэсэн шинэ параметр route-дээ нэмэх хэрэгтэй.

```
blog:
    path: /blog/{page}
    defaults: { _controller: AcmeBlogBundle:Blog:index }
```

Өмнөх {slug} placeholder-ийн адилаар {page} нь таны програмын controller дотор авч болох утгыг авна.

Энэ утга нь өгсөн хуудсанд аль блогын бичлэгүүдийг харуулахыг шийдхэд ашиглагдана.

Гэхдээ placeholder нь default-аар заавал байх шаардлагатай байдаг учир дээрх route маань **/blog** гэсэн утгатай тохирохгүй.

Тэгэхээр блогын 1 дэх хуудсыг харуулахын тулд **/blog/1** гэсэн URL –ээр хандах хэрэгтэй болно. Энэ асуудлыг шийдвэрлэхийн тулд {page} параметерийг optional буюу заавал байх шаардлагагүй болгох хэрэгтэй. Ингэхийн тулд **defaults**-д байх жагсаалсат **page** гэсэн хэсгийг нэмэх хэрэгтэй.

```
1  blog:
2      path: /blog/{page}
3      defaults: { _controller: AcmeBlogBundle:Blog:index, page: 1 }
```

Ингэснээр, {page} placeholder нь заавал байх шаардлагагүй болно. Одоо **/blog** гэсэн URL нь энэ route-д тохирох бөгөөд page параметерийн утгыг 1 гэж тохируулах болно. Мөн **/blog/2** гэсэн ч page параметерийн утга 2 болох болно.

| URL | route | parameters |
|---|---|---|
| /blog | blog | {page} = 1 |
| /blog/1 | blog | {page} = 1 |
| /blog/2 | blog | {page} = 2 |

Мэдээж 1 –ээс илүү optional placeholder байж болно. (жнь: /blog/{slug}/{page}). Гэхдээ optional placeholder-ийн дараах бүх placeholder-үүд optional байх шаардлагатай. Жишээ нь: **/{page}/blog** гэх нь зөв ч гэсэн **page** гэсэн placeholder нь заавал байх шаардлагатай байна гэсэн үг юм.

### Requirement буюу нөхцөл нэмэх

Дээр үүсгэсэн route-үүдийг харая.

```
1  blog:
2      path: /blog/{page}
3      defaults: { _controller: AcmeBlogBundle:Blog:index, page: 1 }
4
5  blog_show:
6      path: /blog/{slug}
7      defaults: { _controller: AcmeBlogBundle:Blog:show }
```

Энэ 2 route хоёулаа **/blog/*** гэсэн URL-тай тохирно. Гэхдээ Symfony нь хамгийн эхэнд олдсон route-ийг л сонгодог. Өөрөөр хэлбэл, **blog_show** route нь хэзээ ч ажиллахгүй гэсэн үг. **/blog/my-blog-post** гэх маягаар хандвал эхний route ажиллах боловч {page} параметер лүү **my-blog-post** –ийн хоосон утгага буцаах болно.

| URL | route | parameters |
|---|---|---|
| /blog/2 | blog | {page} = 2 |
| /blog/my-blog-post | blog | {page} = my-blog-post |

Дээрх жишээн дэх route –ийн хувьд {page} нь зөвхөн бүхэл тоон утга байх үед л зөв ажиллах болно. Тэгэхээр энэ асуудлыг шийдвэрлэхийн тулд **route requirements** буюу **route conditions** нэмж regular expression ашиглана.

```
blog:
    path: /blog/{page}
    defaults: { _controller: AcmeBlogBundle:Blog:index, page: 1 }
    requirements:
        page: \d+
```

**\d+** requirement нь {page} параметерийн авах утга нь бүхэл тоо байх ёстой гэдэгийг заасан regular expression юм. Тэгэхээр дээрх жишээнд байгаа **blog** route нь **/blog/2** гэх мэт URL-ийг авах ба харин **/blog/my-blog-post** гэх зэрэг URL-ийг авч ажиллахгүй.

Одоо харин **/blog/my-blog-post** гэх мэт URL нь **blog_show** route –тэй тохирч, ажиллах болно.

| URL | route | parameters |
|---|---|---|
| /blog/2 | blog | {page} = 2 |
| /blog/my-blog-post | blog_show | {slug} = my-blog-post |
| /blog/2-my-blog-post | blog_show | {slug} = 2-my-blog-post |

Таны програмын нүүр хуудас URL –ээс хамааараад 2 төрлийн хэл ашигладаг гэж бодоё.

```
homepage:
    path: /{culture}
    defaults: { _controller: AcmeDemoBundle:Main:homepage, culture: en }
    requirements:
        culture: en|fr
```

| | |
|---|---|
| / | {culture} = en |
| /en | {culture} = en |
| /fr | {culture} = fr |
| /es | *won't match this route* |

### HTTP method Requirement нэмэх

Мөн та орж ирж байгаа request-ийн method (**GET, HEAD, POST, PUT, DELETE** гэх мэт) дээр нөхцөл ашиглаж болно. Таны програм 2 controller-той холбоо барих формтой байлаа гэж бодоё. Нэг controller нь тухайн формыг харуулах (GET request дээр), нөгөө нь формоос илгээсэн мэдээллийг боловсруулах (POST request дээр) үүрэгтэй байг. Тэгэхээр энэ нь дараах байдалтай байж болно.

```
contact:
    path: /contact
    defaults: { _controller: AcmeDemoBundle:Main:contact }
    methods: [GET]

contact_process:
    path: /contact
    defaults: { _controller: AcmeDemoBundle:Main:contactProcess }
    methods: [POST]
```

Хэдийгээр дээрх 2 route нь 2-уулаа ижил **/contact** гэсэн path-тай боловч эхний route нь GET request байгаа үед, 2 дахь route нь POST request байгаа үед л ажиллана. Энэ нь нэг ижил URL-аар формыг харуулах, тухайн формын мэдээллийг боловсруулах 2 өөр төрлийн controller –ийг ажиллуулах боломжтой гэсэн үг юм.

### Илүү ахисан түвшний routing жишээ

Энэ хүртэл та Symfony-дг routing үүсгэхэд хангалттай мэдлэг олж авлаа. Одоо энд бас нэгэн жишээг үзье.

```
1  article_show:
2    path: /articles/{culture}/{year}/{title}.{_format}
3    defaults: { _controller: AcmeDemoBundle:Article:show, _format: html }
4    requirements:
5      culture: en|fr
6      _format: html|rss
7      year: \d+
```

Эндээс хархад **{culture}** нь **en** эсвэл **fr** –ийн аль нэг, **{year}** нь тоо байгаа үед л дээрх route ажиллахаар харагдаж байна. Тэгэхээр дараах маягийн URL энэ route –тэй тохирно.

- /articles/en/2010/my-post
- /articles/fr/2010/my-post.rss
- /articles/en/2013/my-latest-post.html

**Тусгай *_format* routing параметер**

Энэ жишээнд ашигласан *_format* параметер нь request обектийн "request format" юм. Энэ нь response-ийн Content-Type –ийг тохируулахад ашиглагдана. Тухайлбал, application/json –ний Content-Type –руу json request форматыг хөрвүүлнэ.

### Controller нэршлийн дүрэм

Route бүр **_controller** параметертэй байх ёстой. Энэ нь тухайн route тохирход аль controller ажиллах ёстойг шийднэ. Энэ параметер нь заасан PHP method, болон класс руу чиглүүлэх *logical controller name* гэх энгийн загвар ашигладаг. Энэ загвар нь 2 цэгээр (:) тусгаарлагдсан 3 хэсгээс бүтнэ.

**bundle:controller:action**

Жишээ нь, **AcmeBlogBundle:Blog:show** нь

| Bundle | Controller Class | Method Name |
|---|---|---|
| AcmeBlogBundle | BlogController | showAction |

байна.

Controller нь дараах байдалтай байж болно.

```php
// src/Acme/BlogBundle/Controller/BlogController.php
namespace Acme\BlogBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class BlogController extends Controller
{
    public function showAction($slug)
    {
    // ...
    }
}
```

Symfony нь классын нэрэнд **Controller** гэсэн үгийг (**Blog => BlogController**), method- ийн нэрэнд **Action** гэсэн үгийг (**show => showAction**) нэмдэг гэдэгийг санаарай.

## Route параметер болон Controller аргумент

Route параметер (жишээ нь {slug}) нь controller method руу аргумент болж ирдэг.

```php
public function showAction($slug)
{
    // ...
}
```

Үнэн хэрэгтээ route-н **defaults** хэсэг нь параметерийн утгуудыг нэгтгэн нэг массив үүсгэнэ. Массивын key бүр нь controller-ийн аргумент болно.

Дээр үзсэн жишээнд, дараах хувьсагчууд **showAction()** method-ийн аргумент болж ашиглагдана.

- $culture
- $year
- $title
- $_format
- $_controller

## Гаднаас өөр routing файл оруулж ирэх

Програмд ашиглагдаж байгаа бүх Route-ийг ихэвчлэн **app/config/routing.yml** гэсэн нэг л файлаас дууддаг. Гэхдээ та хүсвэл bundle доторх өөр газраас route файлтай ижил өөр route файлыг дуудан ашиглах боломжтой.

```yaml
# app/config/routing.yml
acme_hello:
    resource: "@AcmeHelloBundle/Resources/config/routing.yml"
```

**resource** түлхүүр үг нь гаднаас Route дуудхад ашиглагдана. Гаднаас дуудагдаж байгаа route файл нь дараах байдалтай байж болно.

```yaml
# src/Acme/HelloBundle/Resources/config/routing.yml
acme_hello:
    path: /hello/{name}
    defaults: { _controller: AcmeHelloBundle:Hello:index }
```

### Гаднаас дуудсан route-д Prefix залгах

Хэрэв таны програм /hello/{name} гэсэний оронд /admin/hello/{name} гэсэн зам байх шаардлагатай байвал:

```yaml
# app/config/routing.yml
acme_hello:
    resource: "@AcmeHelloBundle/Resources/config/routing.yml"
    prefix: /admin
```

гэж хийж болно.

## Route-үүдийг шалгах

Route үүсгэж байхдаа та өөрийнхөө Route-үүдийн тухай дэлгэрэнгүй мэдээллийг авах боломжтой. Програмынхаа бүх route-ийг харахдаа **router:debug** командыг ашиглана. Прожектийнхоо root хавтаснаас дараах кодыг ажиллуулна.

```
$ php app/console router:debug
```

Үүний үр дүнд програмд ашиглагдаж байгаа бүх route-ийн жагсаалтыг хэвлэн харуулна.

```
homepage           ANY   /
contact            GET   /contact
contact_process    POST  /contact
article_show       ANY   /articles/{culture}/{year}/{title}.{_format}
blog               ANY   /blog/{page}
blog_show          ANY   /blog/{slug}
```

Мөн нэр зааж тухайн нэг route-ийн мэдээллийг харж бас болно.

```
$ php app/console router:debug article_show
```

Бас өгсөн route-д URL тохирч байгаа эсэхийг шалгах бол **router:match** комманд ашиглана.

```
$ php app/console router:match /blog/my-latest-post
```

Энэ команд нь тухайн URL –тэй тохирч байгаа route-ийг хэвлэн харуулна.

```
Route "blog_show" matches
```

## URL үүсгэх

Routing систем нь мөн URL-ийг үүсгэхэд ашиглагдана. Үнэндээ routing нь 2 чиглэлт систем юм. Нэг нь controller+параметер лүү URL-ийг чиглүүлэх, нөгөө нь URL руу route+параметер –ийг буцаах. Энэ 2 чиглэлт системийг **match()**[25] болон **generate()**[26] method-ууд бий болгоно. Өмнө үзсэн **blog_show** жишээг харая.

```
$params = $this->get('router')->match('/blog/my-blog-post');
// array(
// 'slug' => 'my-blog-post',
// '_controller' => 'AcmeBlogBundle:Blog:show',
// )

$uri = $this->get('router')->generate('blog_show', array('slug' => 'my-blog-post'));
// /blog/my-blog-post
```

URL-ийг үүсгэхдээ, route-ийн нэр (жнь: blog_show) -ийг заах хэрэгтэй бөгөөд ямар нэг wildcard (жнь: slug = my-blog-post) нь тухайн route-н path-д ашиглагдана. Энэ мэдээллийг ашиглан ямар нэг URL –ийг үүсгэнэ.

```
class MainController extends Controller
{
    public function showAction($slug)
    {
        // ...

        $url = $this->generateUrl(
            'blog_show',
            array('slug' => 'my-blog-post')
        );
    }
}
```

Та өөрийн controller-тоо үндсэн Controller –ийг удамшуулж ашиглаагүй тохиолдолд router сервесийн **generate()** method –ийг ашиглах боломжтой.

Listing

```
use Symfony\Component\DependencyInjection\ContainerAware;

class MainController extends ContainerAware
{
    public function showAction($slug)
    {
        // ...
        $url = $this->container->get('router')->generate(
            'blog_show',
                array('slug' => 'my-blog-post')
            );
        }
    }
```

[25] http://api.symfony.com/master/Symfony/Component/Routing/Router.html#match()
[26] http://api.symfony.com/master/Symfony/Component/Routing/Router.html#generate()

### Query String-тэй URL үүсгэх

**generate** method нь URI үүсгэх wildcard утгын массивыг авна. Гэхдээ хэрэв та яг нэг утга дамжуулах бол query string-ийг URI-д нэмэх хэрэгтэй.

```
$this->get('router')->generate('blog', array('page' => 2, 'category' => 'Symfony'));
// /blog/2?category=Symfony
```

### Темплэйтээс URL үүсгэх

Програм дахь хуудсуудыг хооронд нь холбохдоо темплэйт дотор URL үүсгэх хэрэгтэй.

```
<a href="{{ path('blog_show', {'slug': 'my-blog-post'}) }}">
    Энэ блогыг унших.
</a>
```

### Абсолют URL үүсгэх

Default-аар router нь relative URL-ийг үүсгэдэг. Абсолют болгохын тулд **generateUrl()** method-ын 3 дахь аргументийг **true** болгоно.

```
$this->generateUrl('blog_show', array('slug' => 'my-blog-post'), true);
// http://www.example.com/blog/my-blog-post
```

Темплэйтэд бол **url()** функц ашиглана. Энэ нь **generateUrl()** функц рүү **true** утга дамжуулна.

```
<a href="{{ url('blog_show', {'slug': 'my-blog-post'}) }}">
    Энэ блогыг унших.
</a>
```

Бүлэг 7

# Темплэйт үүсгэж, ашиглах

Дээр үзсэнээр controller нь Symfony2 програм руу орж ирж байгаа Request-ийг боловсруулж, HTML, CSS зэрэг ямар нэг контентийг боловсруулахдаа template engine-тэй харицадаг. Энэ бүлэгт хэрэглэгч рүү илгээж байгаа агуулгад ашиглагдах темплэйтийг хэрхэн үүсгэхийг үзэх болно.

## Темплэйт

Темплэйт бол ямар нэг текст форматаар (HTML, XML, CSV, LaTeX ...) үүсгэж болох энгийн текст файл юм. Темплэйтийн хамгийн түгээмэл болсон төрөл бол PHP темплэйт юм. Энэ нь текст болон PHP кодын холимогоос бүрдсэн PHP файл юм.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Welcome to Symfony!</title>
    </head>
    <body>
        <h1><?php echo $page_title ?></h1>
        <ul id="navigation">
            <?php foreach ($navigation as $item): ?>
                <li>
                    <a href="<?php echo $item->getHref() ?>">
                        <?php echo $item->getCaption() ?>
                    </a>
                </li>
            <?php endforeach; ?>
        </ul>
    </body>
</html>
```

Харин Symfony2 нь **Twig** хэмээх илүү хүчирхэг темплэйтийн хэлийг ашигладаг. Twig нь вэб хөгжүүлэгчдэд илүү хялбар аргаар темплэйт үүсгэх боломж олгодог ба энэ темплэйт нь PHP темплэйтээс илүү хүчирхэг юм.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Welcome to Symfony!</title>
    </head>
    <body>
        <h1>{{ page_title }}</h1>
        <ul id="navigation">
            {% for item in navigation %}
                <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
            {% endfor %}
        </ul>
    </body>
</html>
```

Twig нь дараах 2 төрлийн дүрэмтэй байна.

- **{{ ... }}:** Темплэйт руу илэрхийллийн үр дүнг эсвэл хувьсагчийн утгыг хэвлэнэ.
- **{% ... %}:** темплэйтийг удирдах **tag** бөгөөд энэ нь ямар нэг операторыг биелүүлнэ. Түхайлбал loop гэх мэт.

Мөн Twig нь **Filters** хэмээх зүйлийг агуулдаг ба энэ нь темплэйтийг харуулахын өмнө контентийг өөрчилдөг. Доорх жишээнд: Темплэйтийг харуулахын өмнө **title** хувьсагчын бүх үсгийг том болгоно гэж зааж байна.

```
{{ title|upper }}
```

Twig темплэйт нь **tag**[27] болон **filter**[28]-үүдийг агуулж байдаг бөгөөд та өөрийн шаардлагатай **Extension**[29]-ийг нэмэх боломжтой.

Мөн twig нь өөртөө функцуудыг агуулдаг бөгөөд шинээр функц нэмэх бас боломжтой байдаг. Дараах жишээнд **for tag** ашиглан cycle функцтэй 10 ширхэг div tag хэвлэж байна.

```
{% for i in 0..10 %}
    <div class="{{ cycle(['odd', 'even'], i) }}">
        <!— энд HTML байрлана -->
    </div>
{% endfor %}
```

### Twig темплэйт кэйш

Twig темплэйт маш хурдан ажилладаг бөгөөд энэ нь PHP класс руу хөрвүүлэгдэн ажиллана. Хөрвүүлэгдсэн классууд нь **app/cache/{environment}/twig** хавтсанд байрлана({environment} –ийн оронд **dev** эсвэл **prod** гэсэн environment-үүдийн аль нэг байна).

Debug горимд ажиллаж (ерөнхийдөө dev environment дотор) байгаа үед, та темплэйтдээ ямар нэг өөрчлөлт хийхэд Twig темплэйт нь автоматаар дахин хөрвүүлэгдэж байдаг. Энэ нь таны ажлыг

[27] http://twig.sensiolabs.org/doc/tags/index.html
[28] http://twig.sensiolabs.org/doc/filters/index.html
[29] http://twig.sensiolabs.org/doc/advanced.html#creating-an-extension

хөнгөвчилж, ямар нэг кэйш цэвэрлэх шаардлагагүйгээр таны хийсэн өөрчлөлтийг темплэйт дээр шууд харагдуулна.

Харин Debug горим идэвхгүй байгаа үед (ерөнхийдөө prod environment дотор) та Twig cache хавтасыг цэвэрлэх ёстой. Та програмаа сервер лүү байрлуулахдаа үүнийг санаж байх хэрэгтэй.

## Темплэйтийн удамшил ба үндсэн темплэйт

Ихэнхдээ темплэйт нь **header, footer, sidebar** зэрэг ерөнхий хэсгүүдээс бүтдэг. Symfony2-т, үүнийг хялбар аргаар шийдсэн байдаг. Энэ нь темплэйтийн удамшил бөгөөд удамшил нь нийтлэг хэрэглэгддэг элементүүдийг агуулдаг "**layout**" буюу үндсэн темплэйтийг үүсгэх боломжыг олгоно. Эдгээр элементүүдийг үндсэн темплэйтийн **block** гэсэн хэсэгт тодорхойлж өгнө. Хүүхэд темплэйт нь үндсэн темплэйтээс удамших ба үндсэн темплэйтэд тодорхойлсон **block** –уудыг өөр дээрээ ашиглах боломжтой болно.

Эхлээд үндсэн темплэйтийг үүсгэнэ.

```
{# app/Resources/views/base.html.twig #}
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>{% block title %}Test Application{% endblock %}</title>
    </head>
    <body>
        <div id="sidebar">
            {% block sidebar %}
                <ul>
                    <li><a href="/">Home</a></li>
                    <li><a href="/blog">Blog</a></li>
                </ul>
            {% endblock %}
        </div>

        <div id="content">
            {% block body %}{% endblock %}
        </div>
    </body>
</html>
```

Энэ темплэйт нь 2 баганатай энгийн HTML хуудасны үндсэн бүтцийг тодорхойлж байна. Дээрх жишээнд байгаа **{% block %}** гэсэн 3 хэсэг нь **title**, **sidebar**, **body** хэсгүүдийг тодорхойлно. Эдгээр блок бүрийг хүүхэд элементэд дуудан ажиллуулах боломжтой ба мөн энэ темплэйтийг өөрийг нь шууд дуудан ажиллуулсан ч болно. Шууд дуудан ажиллуулсан тохиолдолд энэ темплэйтийн title, sidebar, body блокуудад ашигласан утгууд нь default утга болох юм.

Харин хүүхэд элемент нь дараах байдалтай байж болно.

```
{# src/Acme/BlogBundle/Resources/views/Blog/index.html.twig #}
{% extends '::base.html.twig' %}

{% block title %}My cool blog posts{% endblock %}
```

```
{% block body %}
    {% for entry in blog_entries %}
        <h2>{{ entry.title }}</h2>
        <p>{{ entry.body }}</p>
    {% endfor %}
{% endblock %}
```

Темплэйтийг удамшуулахдаа **{% extends %}** tag ашиглана. Энэ нь темплэйтийн үндсэн бүтэц болон түүнд тодорхойлсон блокуудыг агуулж буй үндсэн темплэйтийг эхэлж уншина гэдэгийг зааж байгаа юм. Дараа нь хүүхэд темплэйт уншигдаж, эцэг темплэйтийн **title** болон **body** блокууд хүүхэд темплэйтийн блокуудаар солигдоно. **blog_entries** –ийн утгаас хамааражаад гаралт нь дараах байдалтай байна.

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>My cool blog posts</title>
    </head>
    <body>
        <div id="sidebar">
            <ul>
                <li><a href="/">Home</a></li>
                <li><a href="/blog">Blog</a></li>
            </ul>
        </div>

        <div id="content">
            <h2>My first post</h2>
            <p>The body of the first post.</p>

            <h2>Another post</h2>
            <p>The body of the second post.</p>
        </div>
    </body>
</html>
```

Та мөн олон давхар удамшилийг ашиглаж болно. Дараагийн бүлэгт, 3 давхар удамшлын талаар болон Symfony2 прожект дотор темплэйтийг хэрхэн зохион байгуулах талаар үзэх болно.

Темплэйтийг удамшуулан ашиглаж байхдаа дараах зүйлсийг анхаарч байх хэрэгтэй.

- Хэрэв темплэйтэд **{% extends %}** –ийг ашиглах бол энэ нь тухайн темплэйтийн хамгийн эхний tag байх ёстой.
- Хэрэв хэд хэдэн темплэйтэд агуулгууд давхардаж байвал эцэг темплэйтийн **{% block %}** tag руу тэр агуулгыг хийж өгөх нь зүйтэй. Зарим тохиолдолд, шинэ темплэйт үүсгэн тухайн агуулгыг хийж өгөх нь илүү сайн арга байж болно.
- Хэрэв эцэг темплэйт дэх блокын агуулгыг авах хэрэгтэй бол **{{ parent() }}** функц ашиглаж болно.

```
{% block sidebar %}
    <h3>Table of Contents</h3>
    {# ... #}
    {{ parent() }}
{% endblock %}
```

## Темплэйтийн байрлал ба нэршил

Default-аар темплэйтүүд нь 2 өөр газар байрлаж болно.

- **app/Resources/views/:** views хавтас нь програмын үндсэн темплэйт, мөн bundle темплэйтүүдийг хавсаргасан темплэйтүүдийг агуулдаг.
- **path/to/bundle/Resources/views/:** Resources/views хавтасанд тухайн bundle –ийн темплэйт байрлана.

Symfony2 нь темплэйттэй ажиллахдаа **bundle:controller:template** гэсэн дүрмийг ашиглана.

- **AcmeBlogBundle:Blog:index.html.twig:** Энэ дүрэм нь тухайн хуудсанд темплэйтийг заахад ашиглагдана. Доор (:) тэмдгээр тусгаарлагдсан 3 хэсгийг тайлбарлалаа.
    - **AcmeBlogBundle:** (bundle) AcmeBlogBundle гэсэн bundle-д байгаа темплэйт.
    - **Blog:** (controller) Resources/views –ийн доторх Blog гэсэн дэд хавтсанд байгаа темплэйтийг заана.
    - **index.html.twig:** (template) Файлын бодит нэр нь index.html.twig юм.

  AcmeBlogBundle нь src/Acme/BlogBundle гэсэн үг бөгөөд тухайн темплэйтийн бүтэн зам нь src/Acme/BlogBundle/Resources/views/Blog/index.html.twig байх болно.
- **AcmeBlogBundle::layout.html.twig:** Энэ нь AcmeBlogBundle руу үндсэн темплэйтийг заана. Дунд байсан "controller" хэсгийг хассан нь AcmeBlogBundle доторх Resources/views/layout.html.twig темплэйт рүү зааж байгаа юм.
- **::base.html.twig:** Энэ нь програмын үндсэн темплэйт буюу бүтцийг заана. 2 давхар (::) тодорхойлох цэгээр эхэлж байгаа нь bundle болон controller-ийг хоёуланг нь хассан гэсэн үг бөгөөд энэ нь тухайн темплэйт аль нэг bundle –д байрлаагүй, харин оронд нь app/Resources/views/ хавтсанд байгааг илтгэнэ.

#### Темплэйт нэршил

Темплэйтийн нэр нь 2 өргөтгөлтэй байх ба энэ нь ***формат*** болон ***темплэйт Engine***-ийг заана.

- **AcmeBlogBundle:Blog:index.html.twig** – HTML нь формат, Twig нь engine
- **AcmeBlogBundle:Blog:index.html.php** - HTML формат, PHP engine
- **AcmeBlogBundle:Blog:index.css.twig** - CSS формат, Twig engine

Default-аар Symfony2 –т темплэйтийг Twig болон PHP 2-ийн аль нэгээр бичих боломжтой байдаг. Өргөтгөлийн эхний хэсэг (.html, .css, гэх мэт) нь темплэйтийг үүсгэх формат юм. Өргөтгөлийн сүүлийн хэсэг нь (.twig эсвэл .php гэх мэт) аль engine-ийг ашиглах ёстойг зааж байгаа юм.

## Tag болон Helper

Одоо та темплэйтийн тухай үндсэн ойлголттой болж, тэдгээрийг хэрхэн нэрлэж, удамшуулалтыг хэрхэн ашиглах талаар мэддэг боллоо. Өөрөөр хэлбэл хэцүү хэсгүүд ард үлдлээ. Одоо энэ бүлэгт темплэйтүүдийг оруулж ирэх, хуудас руу холболт хийх, зураг оруулах гэх мэт темплэйт дээр хийгддэг ерөнхий үйлдлүүдийг хийхэд туслах хэрэгсэлүүдийн тухай үзнэ.

### Өөр темплэйт оруулж ирэх

Та өөр өөр хуудсан дээр темплэйт эсвэл кодын аль нэг хэсгүүдээс ашиглахыг хүсч магадгүй юм. Тухайлбал, таны програм “news articles” гэсэн хэсэгтэй бөгөөд темплэйт код нь тухайн мэдээг дэлгэрэнгүй харуулах хуудас, хамгийн их уншсан болон хамгийн сүүлд орсон мэдээнүүдийн жагсаалтыг харуулах хуудас зэргийг харуулахад ашиглагдана.

PHP кодыг дахин ашиглах хэрэгтэй болох үед ерөнхийдөө шинээр PHP класс эсвэл функц үүсгэн тухайн кодоо хуулж тавидаг. Темплэйтийн хувьд энэ нь зөв юм. Өөрийн темплэйт рүү өөр дахин ашиглах темплэйтийн кодыг хуулсанаар өөр нэг темплэйтээс оруулж ирэх боломжтой болж байгаа юм. Эхлээд дахин ашиглах шаардлагатай темплэйтийг үүсгэнэ.

```
{# src/Acme/ArticleBundle/Resources/views/Article/articleDetails.html.twig #}
<h2>{{ article.title }}</h2>
<h3 class="byline">by {{ article.authorName }}</h3>

<p>
    {{ article.body }}
</p>
```

Ямар нэг өөр темплэйтээс энэ темплэйтийг оруулж ирэх нь их хялбар байдаг.

```
{# src/Acme/ArticleBundle/Resources/views/Article/list.html.twig #}
{% extends 'AcmeArticleBundle::layout.html.twig' %}

{% block body %}
    <h1>Recent Articles<h1>

    {% for article in articles %}
        {{ include(
            'AcmeArticleBundle:Article:articleDetails.html.twig',
            { 'article': article }
        ) }}
    {% endfor %}
{% endblock %}
```

Темплэйтийг **{{ include() }}** функц ашиглан оруулж ирнэ. **articleDetails.html.twig** темплэйт нь article гэсэн хувьсагч ашиглаж байгаа бөгөөд бид энэ хавьсагч руу ямар нэг утга дамжуулна.

### Controller-ийг хавсаргах

Зарим тохиолдолд танд темплэйт оруулж ирэхээс өөр илүү зүйл хийх шаардлага тулгардаг. Тухайлбал, хамгийн сүүлд орсон 3 мэдээг агуулах нэг sidebar байлаа гэж бодоё. Тэгвэл темплэйтээс шууд өгөгдлийн сан дахь 3 мэдээг авах боломжгүй.

Үүнийг темплэйтэд controller-ийн үр дүнг бүхэлд нь оруулж ирсэнээр шийднэ. Эхлээд сүүлд орсон мэдээллүүдийг харуулах controller-оо үүсгэнэ.

```php
// src/Acme/ArticleBundle/Controller/ArticleController.php
class ArticleController extends Controller
{
    public function recentArticlesAction($max = 3)
    {

        $articles = ...;

        return $this->render(
            'AcmeArticleBundle:Article:recentList.html.twig',
            array('articles' => $articles)
        );
    }
}
```

Үүнийг дагаад recentList темплэйт дараах байдалтай болно.

```twig
{# src/Acme/ArticleBundle/Resources/views/Article/recentList.html.twig #}
{% for article in articles %}
    <a href="/article/{{ article.slug }}">
        {{ article.title }}
    </a>
{% endfor %}
```

Энэ жишээнд тухайн мэдээний URL –ийг (/article/*slug*) гэж хийсэн нь маш муу арга юм. Дараагийн бүлэгт үүнийг хэрхэн зөв шийдэхийг үзэх болно.

Controller-ийг оруулж ирэхдээ, controller-ийн стандарт дүрмийг ашиглах хэрэгтэй. (bundle:controller:action):

```twig
{# app/Resources/views/base.html.twig #}

{# ... #}
<div id="sidebar">
    {{ render(controller('AcmeArticleBundle:Article:recentArticles', {
        'max': 3
    })) }}
</div>
```

Темплэйт рүү хандахгүйгээр controller ашиглан хувьсагч эсвэл ямар нэг мэдээллийг авах нь хурдан ба кодын зохион байгуулалт, дахин ашиглалтыг илүү сайжруулна.

### Хуудсуудыг холбох

Хуудсанд холбоос үүсгэх нь темплэйт дээр хийгддэг нийтлэг үйлдлүүдийн нэг билээ. Темплэйтэд URL ашиглахын оронд Twig –ийн **path** функц ашиглан routing тохиргоон дээр үндэслэн URL-ийг үүсгэнэ. Хэрэв та дараа тодорхой нэг хуудасны URL-ийг өөрчилөх бол routing тохиргоонд өөрчилөлт хийх хэрэгтэй. Темплэйт нь тухайн шинэ URL-ийг автоматаар үүсгэнэ.

Эхлээд, дараах тохиргооны файлд "**_welcome**" хуудасны холболтыг хийе.

```
_welcome:
    path:         /
    defaults:     { _controller: AcmeDemoBundle:Welcome:index }
```

Хуудас руу холбохдоо **path** функцийг ашиглан route рүү заана.

```
<a href="{{ path('_welcome') }}">Home</a>
```

Энэ нь **/** гэсэн URL-ийг үүсгэнэ. Одоо route нь арай цогц болж байна.

```
article_show:
    path: /article/{slug}
    defaults: { _controller: AcmeArticleBundle:Article:show }
```

Энэ тохиолдолд, route –ийн нэр (**article_show**) болон параметерийн утга {**slug**} хоёрыг хоёуланг нь заах хэрэгтэй. Энэ route-ийг ашиглан, өмнөх бүлэгт ашиглаж байсан **recentList** темплэйтийг дахин ашиглан мэдээг холбоё.

```
{# src/Acme/ArticleBundle/Resources/views/Article/recentList.html.twig #}
{% for article in articles %}
    <a href="{{ path('article_show', {'slug': article.slug}) }}">
        {{ article.title }}
    </a>
{% endfor %}
```

Мөн та Twig-ийн **url** функц ашиглан абсолют URL –ийг үүсгэж болно.

```
<a href="{{ url('_welcome') }}">Home</a>
```

### Asset(css, js, image гэх мэт) файлуудыг холбох

Темплэйтүүд нь мөн зураг, javascript, stylesheet болон бусад файлуудтай ажилладаг. Мэдээж та эдгээр файлууд (/images/logo.png гэх мэт) руу замыг холбохдоо хатуу аргаар код бичиж болох ч Symfony2 илүү динамик болгох Twig asset функцыг агуулдаг.

```
<img src="{{ asset('images/logo.png') }}" alt="Symfony!" />

<link href="{{ asset('css/blog.css') }}" rel="stylesheet" type="text/css"
/>
```

Asset функцын үндсэн зорилго бол програмыг илүү авсаархан болгох юм. Хэрэв таны програм таны хостын (http://example.com) –ийн үндсэн хавтсанд байгаа бол logo зурагны зам нь **/images/logo.png** байна. Харин програм дэд хавтсанд (http://example.com/my_app гэх мэт) байгаа бол (/my_app/images/logo.png) гэж зааж өгөх хэрэгтэй болно. Asset функц үүнийг шийдвэрлэх ба зөв замыг үүсгэж ашиглах боломж олгоно.

Хэрэв asset-д absolute URL ашиглах хэрэгтэй бол absolute аргументийг **true** болгох шаардлагатай.

```
<img src="{{ asset('images/logo.png', absolute=true) }}" alt="Symfony!" />
```

## Twig темплэйт рүү Stylesheet болон Javascript оруулах

Javascript, stylesheet оруулж ирэхгүйгээр бүрэн вэб сайт хийх боломжгүй билээ. Эдгээр файлуудыг оруулж ирэхийн тулд үндсэн темплэйтдээ 2 блок нэмэх хэрэгтэй болно.

Тэгэхээр **Head** tag дотор **stylesheet** блокыг дуудаж, javascript-ийг **body** tag хаахын яг өмнө дуудна.

```
{# app/Resources/views/base.html.twig #}
<html>
    <head>
        {# ... #}

        {% block stylesheets %}
            <link href="{{ asset('css/main.css') }}" rel="stylesheet" />
        {% endblock %}
    </head>
    <body>
        {# ... #}

        {% block javascripts %}
            <script src="{{ asset('js/main.js') }}"></script>
        {% endblock %}
    </body>
</html>
```

Мөн та stylesheet, javascript –ийг хүүхэд темплэйт дээр дуудах боломжтой. Жишээ нь, та contact хуудсандаа **contact.css** файлыг оруулах хэрэгтэй бол **contact** хуудасны темплэйтэд дараах зүйлийг хийнэ.

```
{# src/Acme/DemoBundle/Resources/views/Contact/contact.html.twig #}
{% extends '::base.html.twig' %}

{% block stylesheets %}
    {{ parent() }}

    <link href="{{ asset('css/contact.css') }}" rel="stylesheet" />
{% endblock %}

{# ... #}
```

Хүүхэд темплэйтэд **stylesheets** блокыг оруулж ирээд, тэр блок дотороо stylesheet tag-ийг шинээр үүсгэж болно. Мэдээж, эцэг блокын агуулга руу нэмэхийг хүсвэл stylesheet блокоос бүх зүйлийг нь оруулж ирэх **parent()** Twig функцийг ашиглах хэрэгтэй.

Мөн **bundle** дэх **Resources/public** хавтсанд байрлах файлуудыг оруулж ирэх боломжтой. Ингэхийн тулд **`php app/console assets:install target [--symlink]`** командыг ажиллуулах хэрэгтэй ба энэ нь файлуудыг тохирох байрлал руу нь зөөнө. (Target нь default-аар “web” байна).

```
<link href="{{ asset('bundles/acmedemo/css/contact.css') }}" rel="stylesheet" />
```

Эцсийн үр дүнд тухайн хуудас руу **main.css** болон **contact.css** файлууд хоёулаа орж ирнэ.

## Темплэйтийн глобал хувьсагчид

Request бүрийн турш, Symfony2 нь Twig болон PHP темплэйт engine хоёрт хоёуланд нь **app** гэх глобал темплэйт хувьсагчийг тохируулж өгдөг. **app** хувьсагч бол програмын зарим тусгай хувьсагчууд руу автоматаар хандах боломжтой ***GlobalVariables***[30] –ийн нэг юм.

- app.security – Хамгаалалт
- app.user - Идэвхтэй байгаа хэрэглэгч
- app.request – Request обект
- app.session – Session обект
- app.environment – Идэвхтэй байгаа environment
- app.debug – Хэрэв debag горимд байгаа бол true, үгүй бол false байна.

```
<p>Username: {{ app.user.username }}</p>
{% if app.debug %}
    <p>Request method: {{ app.request.method }}</p>
    <p>Application Environment: {{ app.environment }}</p>
{% endif %}
```

## Templating Service-ийг тохируулах, ашиглах

Symfony2-ийн темплэйт системийн зүрх нь **темплэйт Engine** юм. Энэ тусгай обект нь темплэйтийг уншиж, агуулгыг нь буцаана. Controller темплэйтийг уншихдаа темплэйт Engine service-ийг үргэлж ашиглаж байдаг. Жишээ нь:

```
return $this->render('AcmeArticleBundle:Article:index.html.twig');
```

энэ нь

```
use Symfony\Component\HttpFoundation\Response;

$engine = $this->container->get('templating');
$content = $engine->render('AcmeArticleBundle:Article:index.html.twig');

return $response = new Response($content);
```

гэсэнтэй адил байна.

Темплэйт engine (буюу сервис) нь Symfony2-т автоматаар ажиллахаар тохируулагдсан байдаг. Мэдээж үүнийг програмын тохиргооны файлд тохируулж өгсөн байна.

```
# app/config/config.yml
framework:
    # ...
    templating: { engines: ['twig'] }
```

[30] http://api.symfony.com/master/Symfony/Bundle/FrameworkBundle/Templating/GlobalVariables.html

## Гурван давхар удамшил

Темплэйтийг удамшуулах аргаар мөн 3 давхар удамшлыг хэрэгжүүлнэ. Ингэснээр 3 өөр төрлийн темплэйттэй ажиллах боломжтой болно. Эдгээр нь:

- Програмын үндсэн харагдах байдлыг агуулдаг **app/Resources/views/base.html.twig** файлыг үүсгэнэ (Өмнөх жишээтэй адил). Дотороо энэ темплэйтийг **::base.html.twigl;** гэж дуудна.
- Сайтынхаа хэсэг бүрт зориулсан темплэйтийг үүсгэнэ. Жишээ нь, AcmeBlogBundle, **AcmeBlogBundle::layout.html.twig** хэмээх темплэйт нь блогын хэсэг бүрийн элементүүдийг агуулна.

```
{# src/Acme/BlogBundle/Resources/views/layout.html.twig #}
{% extends '::base.html.twig' %}

{% block body %}
    <h1>Blog Application</h1>

    {% block content %}{% endblock %}
{% endblock %}
```

- Хуудас бүрт зориулан тус тусын темплэйтийг үүсгэх ба тохирох хэсгийн темплэйтийг оруулж ирнэ. Жишээ нь, “index” хуудас **AcmeBlogBundle:Blog:index.html.twig** рүү зарим зүйлсийг дуудах ба тухайн блогын бичиглэлүүдийг жагсаан харуулна.

```
{# src/Acme/BlogBundle/Resources/views/Blog/index.html.twig #}
{% extends 'AcmeBlogBundle::layout.html.twig' %}

{% block content %}
    {% for entry in blog_entries %}
       <h2>{{ entry.title }}</h2>
       <p>{{ entry.body }}</p>
    {% endfor %}
{% endblock %}
```

Энэ темплэйт нь **AcmeBlogBundle::layout.html.twig**- ийг өөр дээрээ дуудах ба хэсгийн темплэйт нь үндсэн темплэйт (**::base.html.twig**)-ийг мөн өөр дээрээ дуудсан байна. Энэ бол 3 давхар удамшлын ерөнхий загвар юм.

## Output Escaping /гаралтыг шүүх/

Темплэйтээс HTML кодыг үүсгэх үед энд тухайн темплэйтийн хувьсагч нь төлөвлөөгүй HTML, эсвэл клэйнт талын аюултай кодыг гаргах эрсдэл үргэлж гардаг. Үүний үр дүнд HTML хуудасны динамик агуулга эвдрэх, хакеруудад Cross Site Scripting (XSS) халдлага хийх боломж олгох зэрэг талтай. Энд нэг энгийн жишээ авч үзье.

```
Hello {{ name }}
```

Энэ тохиолдолд хэрэглэгч өөрийн нэрээ оруулахын оронд дараах кодыг оруулж болно.

```
<script>alert('hello!')</script>
```

Ингэснээр javascript –ийн анхааруулах цонх гарч ирнэ.

```
Hello <script>alert('hello!')</script>
```

Гэхдээ энэ нь аюулгүй ч хэрэглэгч илүү аюултай javascript код оруулах боломжтой гэсэн үг юм.

Тэгвэл энэ асуудлыг Output escaping ашиглан шийднэ. Үүнийг ашигласнаар темплэйт аюул багатай байх бөгөөд script зэрэг tag-ийг дэлгэц рүү энгийн текст хэлбэрээр хэвлэх болно.

```
Hello &lt;script&gt;alert(&#39;helloe&#39;)&lt;/script&gt;
```

Twig болон PHP темплэйт системүүд нь энэ асуудлыг өөр өөр аргаар шийдвэрлэдэг. Хэрэв та Twig ашиглаж байгаа бол output escaping нь default-аар идэвхтэй байх бөгөөд таны програм хамгаалагдсан байх болно. Харин PHP ашиглаж байгаа бол, Escaping нь автоматаар ажиллахгүй бөгөөд та шаардлагатай газраа өөрийн гараар Escaping хийж өгөх хэрэгтэй болно.

### Twig темплэйт дэх Output Escaping

Хэрэв та Twig темплэйтийг ашиглаж байгаа бол, output escaping нь default-аар идэвхтэй байна. Энэ нь хэрэглэгчийн оруулж байгаа кодноос таныг хамгаалах болно. Output escaping нь тухайн агуулгыг өөртөө хүлээн авч, HTML гаралтыг хянана.

Зарим тохиолдолд, та хувьсагчийг унших зэрэг үед output escaping-ийг хаах хэрэгтэй болдог. Админ хэрэглэгч HTML кодыг агуулсан мэдээ бичих боллоо гэхэд Twig нь Default-аар мэдээнд escape хийх болно.

Үүнийг хэвийн уншихын тулд raw filter нэмнэ.

```
{{ article.body|raw }}
```

Мөн та темплэйтийг бүхэлд нь эсвэл зөвхөн {% block %} дотор output escaping-ийг хааж болно.

### PHP темплэйт дэх Output Escaping

PHP темплэйт ашиглаж байгаа үед output escaping нь автоматаар ажиллахгүй. Тийм болохоор Output escaping ашиглахдаа **escape()** гэх тусгай method ашиглана.

```
Hello <?php echo $view->escape($name) ?>
```

**escape()** method нь тухайн хувьсагчийг хүлээн авч HTML доторх агуулгыг уншина. 2 дахь аргумент нэмсэнээр агуулгыг өөрчилнө. Жишээ нь: javascript кодыг тэр хэвээр нь гаргах бол **js** ашиглана.

```
var myMsg = 'Hello <?php echo $view->escape($name, 'js') ?>';
```

## Debug хийх

PHP ашиглаж байгаа үед, хэрэв та хувьсагчийн дамжуулсан утгыг хурдан хайж олох шаардлагатай бол **var_dump()** ашиглана. Мөн Twig ашиглан дээрх үйлдлийг хийж болно.

Үүнийг dump функц ашиглан хийнэ.

```
{# src/Acme/ArticleBundle/Resources/views/Article/recentList.html.twig #}
{{ dump(articles) }}

{% for article in articles %}
    <a href="/article/{{ article.slug }}">
        {{ article.title }}
    </a>
{% endfor %}
```

## Дүрмийн алдааг шалгах

Twig темплэйтийн дүрмийн алдааг **twig:lint** console команд ашиглан шалгах боломжтой.

```
# файлын нэрээр шалгах боломжтой:
$ php app/console twig:lint src/Acme/ArticleBundle/Resources/views/Article/
recentList.html.twig

# эсвэл хавтасаар нь:
$ php app/console twig:lint src/Acme/ArticleBundle/Resources/views

# эсвэл bundle-ийн нэрээр нь:
$ php app/console twig:lint @AcmeArticleBundle
```

## Темплэйт формат

Темплэйтүүд нь ямар ч форматтай байж болох ба агуулгыг нэг ерөнхий аргаар уншина. Ихэнх тохиолдолд HTML агуулгыг унших ба JavaScript, CSS, XML гэх мэт ямар ч форматтай агуулгыг унших боломжтой.

Жишээ нь XML форматаар index хуудасны мэдээг боловсруулахдаа темплэйтийн нэрэнд тухайн форматыг залгана.

- XML темплэйтийн нэр: AcmeArticleBundle:Article:index.xml.twig
- XML темплэйт файлын нэр: index.xml.twig

Зарим тохиолдолд "request format" дээр үндэслээд өөр өөр форматыг нэг controller- т боловсруулах хэрэгтэй болдог.

```
public function indexAction()
{
    $format = $this->getRequest()->getRequestFormat();

    return $this->render('AcmeBlogBundle:Blog:index.'.$format.'.twig');
}
```

Формат параметерийг залган холбоос үүсгэхдээ _format түлхүүр үг ашиглана.

```
<a href="{{ path('article_show', {'id': 123, '_format': 'pdf'}) }}">
    PDF Version
</a>
```

8-р бүлэг

# Өгөгдлийн сан ба Doctrine

Ямар ч програмд хийгддэг нийтлэг үйлдлүүдийн нэг бол өгөгдлийн санд мэдээлэл хадгалах, өгөгдлийн сангаас буцаан мэдээллийг унших явдал юм. Symfony нь **Doctrine**[31] хэмээх сантай нэгдэж энэ үйлдлийг хялбархан гүйцэтгэх боложмыг танд олгоно. Энэ бүлэгт, Doctrine –ийн үндсэн философийн тухай болон өгөгдлийн сантай хэрхэн хялбар ажиллахыг судлах болно.

Энэ бүлэгт Doctrine ORM-ийн тухай үзэх бөгөөд энэ нь relational database (MySQL, PostgreSQL, Microsoft SQL гэх мэт) –рүү обектуудыг чиглүүлэх үүрэгтэй.

## Энгийн жишээ: Product

**Doctrine** хэрхэн ажилладагийг ойлгох хялбар арга бол түүнтэй ажиллаж үзэх юм. Энэ бүлэгт, өгөгдлийн санг тохируулж, Product обект үүсгэн, түүнийгээ өгөгдлийн санд хадгалах, буцаан унших зэрэг үйлдлийг хийж үзэх болно.

Хэрэв та энд үзэх жишээг дагаж хийх бол **AcmeStoreBundle** –ийг эхлээд дараах кодоор үүсгэх хэрэгтэй.

```
1   $ php app/console generate:bundle --namespace=Acme/StoreBundle
```

[31] http://www.doctrine-project.org/

### Өгөгдлийн санг тохируулах

Хамгийн түрүүнд өгөгдлийн сангийн холболтын мэдээллийг тохируулах хэрэгтэй. Энэ тохиргоог ихэвчлэн **app/config/parameters.yml** файлд бичнэ.

```
# app/config/parameters.yml
parameters:
    database_driver:    pdo_mysql
    database_host:      localhost
    database_name:      test_project
    database_user:      root
    database_password:  password
# ...
```

Ингээд Doctrine таны өгөгдлийн сантай ажиллахад бэлэн боллоо. Одоо харин өгөгдлийн сангаа үүсгэх хэрэгтэй.

```
$ php app/console doctrine:database:create
```

**Өгөгдлийн санг UTF8 –аар тохируулах**

Хөгжүүлэгчидийн гаргадаг  нэг алдаа бол прожектоо эхэлж байх явцдаа өгөгдлийн сангийнхаа default фонтыг тохируулахаа мартдаг. Үүнийг хамгийн түрүүнд тохируулж байх нь зөв юм. Гэхдээ хэрэв та мартсан бол дараах командыг ажиллуулан тохируулах боломжтой.

```
$ php app/console doctrine:database:drop --force
$ php app/console doctrine:database:create
```

MySQL өгөгдлийн санд UTF8-ийг Default-аар тохируулахдаа тохиргооны файл (ихэнхдээ my.cnf) -даа дараах мөр кодыг нэмнэ.

```
[mysqld]
collation-server = utf8_general_ci
character-set-server = utf8
```

Хэрэв та SQLite өгөгдлийн сан ашиглах бол өгөгдлийн сан байгаа замаа зааж өгөх хэрэгтэй.

```
# app/config/config.yml
doctrine:
    dbal:
        driver: pdo_sqlite
        path: "%kernel.root_dir%/sqlite.db"
        charset: UTF8
```

### Entity класс үүсгэх

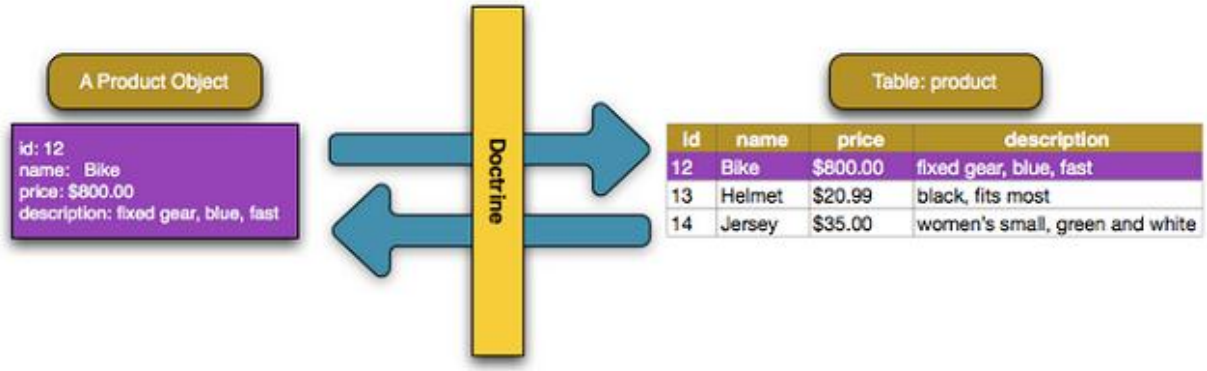
Бидний жишээ болгон хийж байгаа програм маань бүтээгдэхүүний тухай мэдээллийг харуулах зорилготой. Тэгэхээр эдгээр обектийг харуулах **Product** гэсэн обект хэрэг болно. **AcmeStoreBundle** –ийн дотор байгаа **Entity** хавтас дотор доорх класыг үүсгэе.

```
1  // src/Acme/StoreBundle/Entity/Product.php
2  namespace Acme\StoreBundle\Entity;
3
4  class Product
5  {
6      protected $name;
7      protected $price;
8      protected $description;
9  }
```

Ийм төрлийн классыг ихэвчлэн “entity” гэж нэрлэдэг. Энэ класс одоохондоо өгөгдлийн сан руу хандах боломжгүй энгийн нэг PHP класс байна.

### Mapping мэдээлэл нэмэх

Doctrine нь өгөгдлийн санд обектуудыг бүхлээр нь хадгалдаг ба мөн өгөгдлийн сангаас обектуудыг бүхлээр нь авдаг.




Doctrine үүнийг хийхийн тулд **"metadata"** үүсгэх буюу тухайн Product класс болон түүний property-үүдыг өгөгдлийн сан руу хэрхэн чиглүүлэх ёстойг Doctrine-д зааж өгөх хэрэгтэй. Энэ metadata нь YAML, XML форматад байж болно. Эсвэл Product класс дотор шууд хийж өгсөн ч болно.

```
1  // src/Acme/StoreBundle/Entity/Product.php
2  namespace Acme\StoreBundle\Entity;
3
4  use Doctrine\ORM\Mapping as ORM;
5
6  /**
7  * @ORM\Entity
8  * @ORM\Table(name="product")
9  */
10 class Product
11 {
12     /**
13     * @ORM\Column(type="integer")
14     * @ORM\Id
```

```
15      * @ORM\GeneratedValue(strategy="AUTO")
16      */
17     protected $id;
18
19     /**
20      * @ORM\Column(type="string", length=100)
21      */
22     protected $name;
23
24     /**
25      * @ORM\Column(type="decimal", scale=2)
26      */
27     protected $price;
28
29     /**
30      * @ORM\Column(type="text")
31      */
32     protected $description;
33 }
```

Bundle нь **metadata-г** зөвхөн нэг форматаар тодорхойлохыг зөвшөөрдөг. Тухайлбал, YAML форматтай PHP форматтай метадатаг хольж болохгүй.

Хэрэв хүснэгтэд нэр өгөөгүй бол entity класын нэрээр автоматаар шууд нэрлэгддэг.

Классын нэр болон property-г нэрлэхдээ SQL-ийн түлхүүр үгтэй адилхан байж болохгүй гэдэгийг сайн анхааараарай.

### Getter болон Setter үүсгэх

Хэдийгээр Doctrine-д өгөгдлийн сантай Product обект хэрхэн харилцах тухай мэдээллийг зааж өгсөн ч гэсэн энэ класс хараахан хэрэгцээтэй болоогүй байна. Тэгэхээр та энэ класст **getter** болон **setter** method –уудыг үүсгэх хэрэгтэй. Үүнийг хийхдээ дараах кодыг ажиллуулна.

```
1  $ php app/console doctrine:generate:entities Acme/StoreBundle/Entity/Product
```

Энэ команд Product класын бүх **getter, setter** method-уудыг автоматаар үүсгэнэ. Үүнийг та дахин дахин ажиллуулсан ч өмнөх дээр дараад л дахиад үүсгэчихдэг.

Doctrine-ний entity үүсгэгч нь энгийн getter, setter үүсгэдэг. Та үүссэн entity классыг шалгаж үзээд өөрийн шаардлагадаа тохируулаад getter/setter –ийг сайжруулах боломжтой.

### Өгөгдлийн сангийн хүснэгт үүсгэх

Doctrine таны програм дахь entity класс бүрт хэрэгтэй өгөгдлийн сангийн бүх хүснэгтүүдийг автоматаар үүсгэдэг. Ингэхдээ дараах командыг ажиллуулна.

```
$ php app/console doctrine:schema:update --force
```

Энэ команд нь таны өгөгдлийн сан ямар байх ёстойг entity классын тухай мэдээлэл дээр үндэслээд харьцуулалт хийж үзэх ба өгөгдлийн санг шинэчлэхэд хэрэгтэй SQL statement-үүдийг үүсгэнэ. Өөрөөр хэлбэл, хэрэв та Product класс руу шинэ property нэмэх үед энэ үйлдэл дахин ажиллаж, өмнө үүссэн байгаа product хүснэгт рүү шинэ багана нэмхэд хэрэглэгдэх "alter table" statement үүснэ.

Ингээд таны өгөгдлийн сан product гэсэн хүснэгттэй боллоо.

### Өгөгдлийн сан руу обектийг хадгалах

Бид Product entity болон түүнд тохирсон product хүснэгтийг үүсгэсэн бөгөөд түүндээ өгөгдөл хадгалхад бэлэн боллоо. Controller-оос энэ үйлдлийг хийхэд маш хялбар байдаг. DefaultController-руу дараах method-ийг нэмнэ.

```
// src/Acme/StoreBundle/Controller/DefaultController.php

// ...
use Acme\StoreBundle\Entity\Product;
use Symfony\Component\HttpFoundation\Response;

public function createAction()
{
    $product = new Product();
    $product->setName('A Foo Bar');
    $product->setPrice('19.99');
    $product->setDescription('Lorem ipsum dolor');

    $em = $this->getDoctrine()->getManager();
    $em->persist($product);
    $em->flush();

    return new Response('Created product id '.$product->getId());
}
```

Дээрх кодыг тайлбарлая:

- 9-12 р мөр: Энгийн PHP обекттой адилаар **$product** обект үүсгэн утга оноож байна.
- 14-р мөр: Өгөгдлийн сан руу өгөгдөл хадгалах, өгөгдлийг буцаан унших үйлдлийг гүйцэтгэдэг **Doctrine entity manager** обектийг дуудаж байна.
- 15-р мөр: **persist()** method нь **$product** обекттой ажиллана гэдэгийг Doctrine –д зааж өгч байна.
- **flush()** method дуудагдсанаар Doctrine обектуудыг хайж олоод, боломжтой бол өгөгдлийн сан руу хадгална. Өөрөөр хэлбэл entity manager INSERT query-г биелүүлэн, product хүснэгтэд нэг мөрийг үүсгэнэ.

### Өгөгдлийн сангаас обектийг унших

Өгөгдлийн сангаас өгөгдлийг буцаан авах нь хялбар байдаг. Жишээ нь: тухайн өгөгдлийг **id** утгаар нь ялган авч үзүүлэхээр route-ээ тохируулсан бол controller дараах байдалтай байж болно.

```
public function showAction($id)
{
    $product = $this->getDoctrine()
        ->getRepository('AcmeStoreBundle:Product')
        ->find($id);

    if (!$product) {
        throw $this->createNotFoundException(
            'No product found for id '.$id
        );
    }

}
```

**"repository"** хэмээх зүйлийг ашиглан query-г ажиллуулна. Үүнийг тухайн классын entity-г авхад л тусалдаг PHP класс гэж ойлгож болно. Entity классаар **repository** обект руу хандаж болно.

```
$repository = $this->getDoctrine()
    ->getRepository('AcmeStoreBundle:Product');
```

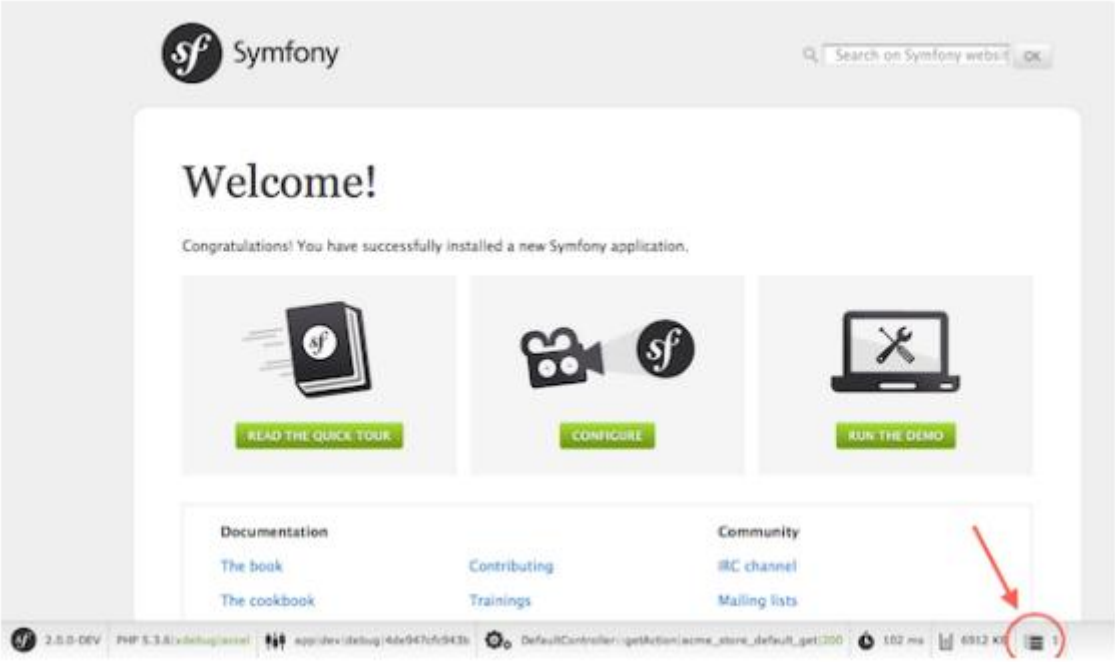
Repository-г нэг удаа дуудаад, түүний бүх method руу хандаж болно.

```
// primary key (ихэнхдээ "id") -р нь авах
$product = $repository->find($id);

// баганы нэрээр нь
$product = $repository->findOneById($id);
$product = $repository->findOneByName('foo');

// бүх бүтээгдэхүүнийг
$products = $repository->findAll();

// Дурын нэг утгаар бүтээгдэхүүнүүдийг багцлан авах
$products = $repository->findByPrice(19.99);
```

Мөн **findBy, findOneBy** method –уудыг ашиглан нөхцөл шалгаж өгөгдлийг авах боломжтой.

```
// нэр болон үнээр нь нэг л бүтээгдэхүүн авах
$product = $repository->findOneBy(
    array('name' => 'foo', 'price' => 19.99)
);

// тухайн нэртэй тохирч байгаа бүх бүтээгдэхүүний үнийн өсөх дарааллаар
$products = $repository->findBy(
    array('name' => 'foo'),
    array('price' => 'ASC')
);
```

Ямар нэг хуудсыг ажиллуулах үед хичнээн query ажиллаж байгааг web debug toolbar-ийн баруун доод буланд байх товч дээр дарж харах боломжтой.




### Обектийг засах

Doctrine-аас өгөгдлийг авч, түүнд засвар хийнэ.

```
1  public function updateAction($id)
2  {
3      $em = $this->getDoctrine()->getManager();
4      $product = $em->getRepository('AcmeStoreBundle:Product')->find($id);
5
6      if (!$product) {
7          throw $this->createNotFoundException(
8              'No product found for id '.$id
9          );
10     }
11
12     $product->setName('New product name!');
13     $em->flush();
14
15     return $this->redirect($this->generateUrl('homepage'));
16 }
```

Обектийг засах ажил нь 3 алхамтай:

1. Doctrine-аас обектийг авна.
2. Обектод өөрчлөлт хийнэ.
3. Entity manager –н **flush()** method-ийг дуудна.

Энд **$em->persist($product)** –ийг дуудах ёсгүй гэдэгийг анхааараарай. Энэ method-ийг дахин дуудвал $product обекттой ажиллана гэдэгийг Doctrine-д заана. Энэ тохиолдолд Doctrine-аас $product обектийг авсанаар энэ үйлдэл нь аль хэдийн хийгдсэн байгаа юм.

### Обектийг устгах

Обектийг устгахдаа Entity manager-ээс **remove()** method-ийг дуудна.

```
$em->remove($product);
$em->flush();
```

Энэ үед DELETE query ажиллах хэдий ч flush() method дуудагдах хүртэл биелэгдэхгүй.

### Обекттой ажиллах query бичих

Ямар нэг ажиллагаагүйгээр үндсэн query-нүүдийг хэрхэн ажиллуулахыг үзлээ.

```
$repository->find($id);
$repository->findOneByName('Foo');
```

Мэдээж, Doctrine ашиглан илүү цогц query-г бичихдээ **Doctrine Query Language** (DQL) ашиглана. DQL нь SQL –тэй ижил боловч хүснэгт (жишээ нь product)-д зориулан query бичихийн оронд Entity класс (жишээ нь Product)-ийн обектуудад зориулан query бичнэ гэдэгийг ойлгох хэрэгтэй.

Doctrine-д query бичихэд танд 2 сонголт байна. Эдгээр нь цэвэр Doctrine query бичих, эсвэл Doctrine-ний **Query Builder-**ийг ашиглах юм.

### Doctrine Query Builder ашиглан query бичих

Бүтээгдэхүүний үнийг авах query бичих хэрэгтэй боллоо гэж төсөөлөе. Гэхдээ 19.99  -ээс эхлээд, үнийг өсөх дарааллаар гаргана. Ингэхийн тулд Doctrine-ний QueryBuilder-ийг ашиглаж болно.

```
$repository = $this->getDoctrine()
    ->getRepository('AcmeStoreBundle:Product');

$query = $repository->createQueryBuilder('p')
    ->where('p.price > :price')
    ->setParameter('price', '19.99')
    ->orderBy('p.price', 'ASC')
    ->getQuery();

$products = $query->getResult();
```

QueryBuilder обект нь query-г үүсгэхэд шаардлагатай method-уудыг агуулдаг. Иймд **getQuery()** method-ийг дуудсанаар query-ийн үр дүнг гаргахад ашигладах энгийн Query обект буцаана.

**getResult()** method нь үр дүнг массив болгон буцаана. Зөвхөн нэг л үр дүнг авах бол **getSingleResult(),** эсвэл **getOneOrNullResult()** method ашиглана.

```
$product = $query->getOneOrNullResult();
```

### DQL ашиглан query бичих

QueryBuilder –ийн оронд та DQL ашиглан шууд Query бичиж болно.

```
$em = $this->getDoctrine()->getManager();
$query = $em->createQuery(
```

```
    'SELECT p
    FROM AcmeStoreBundle:Product p
    WHERE p.price > :price
    ORDER BY p.price ASC'
)->setParameter('price', '19.99');

$products = $query->getResult();
```

Хэрэв танд SQL тэй ажиллах нь илүү эвтэйхэн байдаг бол DQL тэй ажиллахад мөн л хялбар байх болно. Хамгийн том ялгаа нь гэвэл  өгөгдлийн сангийн мөрүүдтэй ажиллахын оронд обекттой ажиллах юм. Үүний жишээ бол, **AcmeStoreBundle:Product** обектоос **SELECT** хийгээд **p** тэмдэглэгээг ашиглаж байна.

### Хэрэглэгчийн үүсгэсэн Repository класс

Өмнөх бүлэгт, controller дотор илүү query-нүүдийг үүсгэн ашиглаж эхэлсэн. Эдгээр query-г тусгаарлах, шалгах, дахин ашиглах-ын тулд хөгжүүлэгч өөрийн repository классыг үүсгэн, ашиглах нь маш сайн арга юм.

Үүнийг хийхдээ, repository классын нэрийг **mapping definision** рүү нэмнэ.

```
// src/Acme/StoreBundle/Entity/Product.php
namespace Acme\StoreBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
* @ORM\Entity(repositoryClass="Acme\StoreBundle\Entity\ProductRepository")
*/
class Product
{
    //...
}
```

Өмнө нь getter, setter method-уудыг үүсгэхэд ашигласан командтай адил командыг ажиллуулснаар Doctrine нь repository классыг үүсгэх болно.

```
$ php app/console doctrine:generate:entities Acme
```

Дараа нь, **findAllOrderedByName()** хэмээх method-ийг шинээр үүсгэсэн repository класс даа нэмж өгнө. Энэ method нь Product –ийн бүх entity-г цагаан толгойн дарааллаар авна.

```
// src/Acme/StoreBundle/Entity/ProductRepository.php
namespace Acme\StoreBundle\Entity;

use Doctrine\ORM\EntityRepository;

class ProductRepository extends EntityRepository
{
    public function findAllOrderedByName()
    {
        return $this->getEntityManager()
            ->createQuery(
              'SELECT p FROM AcmeStoreBundle:Product p ORDER BY p.name ASC'
            )
        ->getResult();
    }
}
```

## Entity Relationship/Association

Та бүтээгдэхүүнүүдийг ангилан нэг категорт оруулах шаардлагатай байлаа гэж бодоё. Энэ тохиолдолд танд **Category** гэсэн обект хэрэгтэй бөгөөд түүнтэй **Product** обектийг холбоно. Ингэхийн тулд эхлээд, **Category entity** класс үүсгэнэ.

```
$ php app/console doctrine:generate:entity --entity= "AcmeStoreBundle:Category"
--fields="name:string(255)"
```

Энэ нь **id, name** гэсэн талбарууд болон **getter, setter** функцүүдийг агуулсан **Category entity** классыг үүсгэнэ.

### Relationship Mapping Metadata

**Category** болон **Product** Entity-г холбохдоо **Category** класс дотор **products** property-г үүсгэнэ.

```
// src/Acme/StoreBundle/Entity/Category.php

// ...
use Doctrine\Common\Collections\ArrayCollection;

class Category
{
    // ...
    /**
    * @ORM\OneToMany(targetEntity="Product", mappedBy="category")
    */
    protected $products;

    public function __construct()
    {
        $this->products = new ArrayCollection();
    }
}
```

Нэгдүгээрт, **Category** обект нь олон тооны бүтээгдэхүүн обектуудтай холбогдох учир **products** гэсэн property нь эдгээр **Product** обектууд руу нэмэгдсэн байна.

Doctrine нь ArrayCollection обектод $products property-г заавал байхыг шаарддаг учираас _construct() method нь чухал ач холбогдолтой. ArrayCollection обект нь массивтай яг адил боловч ажиллахад илүү уян хатан байдаг.

Дараа нь, Product класс тус бүр нь яг нэг Category обекттой холбоотой учираас та Product класс руу , **$category** property –г нэмнэ.

```
// src/Acme/StoreBundle/Entity/Product.php
// ...

class Product
{
    // ...

    /**
    * @ORM\ManyToOne(targetEntity="Category", inversedBy="products")
```

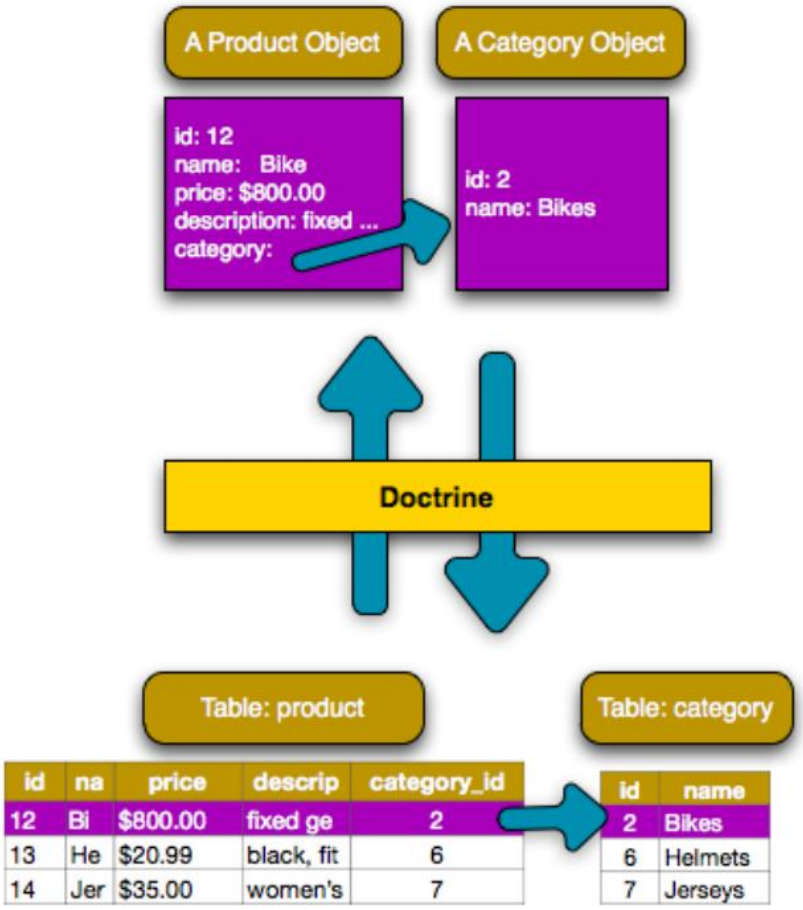
```
10      * @ORM\JoinColumn(name="category_id", referencedColumnName="id")
11      */
12 protected $category;
13 }
```

Эцэст нь, Category болон Product класс руу нэмсэн шинэ property-д хамаарах getter болон setter method-уудыг үүсгэхийг Doctrine –д зааж өгөх хэрэгтэй.

```
1  $ php app/console doctrine:generate:entities Acme
```

Ингээд **one-to-many** гэсэн холбоотой **Category** болон **Product** хэмээх 2 класстай боллоо. **Category** класс нь **Product** обектуудын массивыг, **Product** обект нь нэг **Category** обектыг агуулна.

Одоо, **Product** классын **$category** property-н metadata нь, тухайн холбогдсон класс нь **Category** бөгөөд тэр нь **product** хүснэгтийн **category_id** баганд тухайн категорийн **id** утгыг хадгална гэдэгийг Doctrine-д зааж өгөх юм.  Өөрөөр хэлбэл, **Category** обект нь **$category** property-д хадгалагдаж байгаа хэдий ч цаанаа Doctrine нь product хүснэгтийн category_id багана руу тухайн категорийн id утгыг хадгалж байдаг гэсэн үг юм.




Ингээд шинээр **category** хүснэгт, **product.category_id** болон шинэ **foreign key** нэмнэ гэдэгийг Doctrine-д зааж өгнө:

```
1  $ php app/console doctrine:schema:update --force
```

### Хоорондоо холбоо хамааралтай Entitie-нүүдийг хадгалах

Одоо controller –т байх дараах Action method-ийг харая.

```
// ...

use Acme\StoreBundle\Entity\Category;
use Acme\StoreBundle\Entity\Product;
use Symfony\Component\HttpFoundation\Response;

class DefaultController extends Controller
{
    public function createProductAction()
    {
        $category = new Category();
        $category->setName('Main Products');

        $product = new Product();
        $product->setName('Foo');
        $product->setPrice(19.99);
        // энэ product-ийг category-той холбоно
        $product->setCategory($category);

        $em = $this->getDoctrine()->getManager();
        $em->persist($category);
        $em->persist($product);
        $em->flush();

        return new Response(
            'Created product id: '.$product->getId()
            .' and category id: '.$category->getId()
        );
    }
}
```

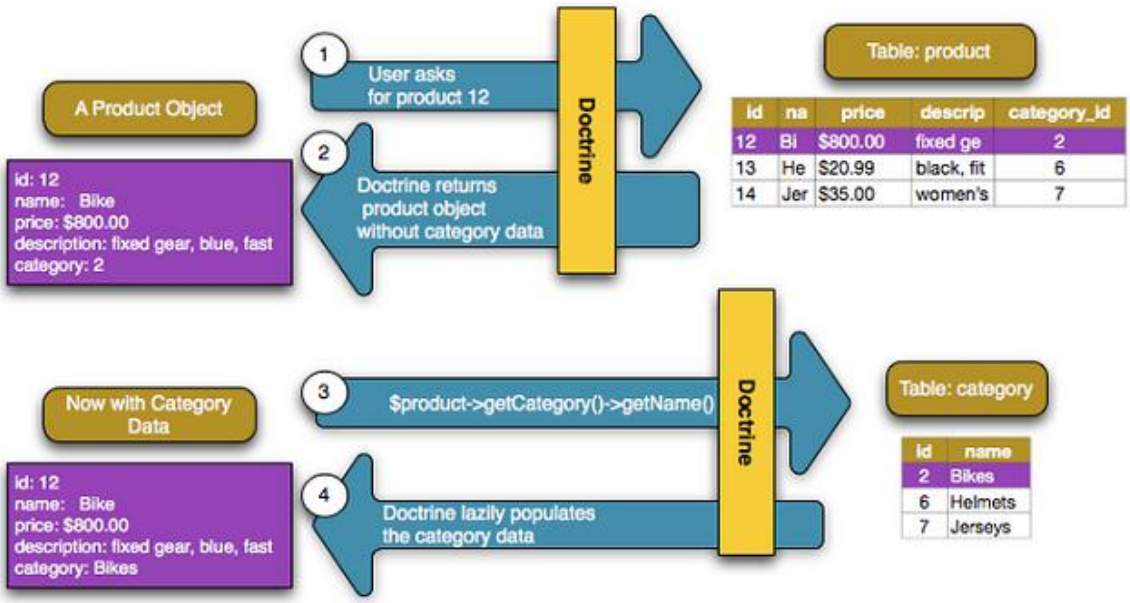
Дээрх код нь **category** болон **product** гэсэн өгөгдлийн сангийн хүснэгтүүд рүү 1 мөр мэдээлэл нэмнэ. Шинээр нэмэгдэн орж байгаа категорийн **id** утга нь шинээр нэмэгдэж байгаа product-ийн **product.category_id** баганд орно. Ингэж хоорондоо холбоотой өгөдлүүдийг хадгална.

### Хоорондоо холбоотой обектуудыг авах

Хоорондоо холбоотой өгөгдлүүдийг дараах аргаар авна. Эхлээд, **$product** обектыг аваад, дараа нь харгалзах **Category** луу хандана.

```
public function showAction($id)
{
    $product = $this->getDoctrine()
        ->getRepository('AcmeStoreBundle:Product')
        ->find($id);
    $categoryName = $product->getCategory()->getName();

    // ...
}
```

Дээрх жишээнд, тухайн бүтээгдэхүүний **id** –гаар **Product** обектийг авна. Дараа нь **$product->getCategory()->getName()** –ийг дуудахад Doctrine энэ **Product** -тай холбоотой **Category**-ийг хайх query-г биелүүлнэ. Энэ нь **$category** обектийг бэлтгээд түүнийгээ тан руу буцаана.




Тухайн product-ийн холбоотой категор луу хандахад хялбар ч категорийн өгөгдлийг хэрэглэгч асуух хүртэл авах боломжгүй.

Мөн та өөр аргаар авч бас болно.

```
1  public function showProductAction($id)
2  {
3      $category = $this->getDoctrine()
4          ->getRepository('AcmeStoreBundle:Category')
5          ->find($id);
6
7      $products = $category->getProducts();
8
9      // ...
10 }
11
```

Энэ тохиолдолд, эхлээд та ганц **Category** обектийг аваад, дараа нь Doctrine тухайн Category-тай холбоотой **Product** обектуудыг авна. Гэвч, таныг дуудсан тохиолдолд (->getProducts()) нэг л удаа дуудагдана. **$products** хувьсагч бол тухайн **Product** обектуудын массив юм.

### Холбоотой мэдээллүүдийг нэгтгэх

Дээрх жишээнд, хоорондоо холбоотой обектуудыг авахад 2 ширхэг query ашиглаж байна. Нэг нь эх обект (жнь: Category) -ийг, нөгөө нь түүнд холбоотой обект (жнь: Product objects)-ийг авах.

Харин query-г нэгтгэснээр 2 дахь query-г бичихээс зайлсхийж болно. **ProductRepository** класс руу дараах method-ийг нэмэе.

```
// src/Acme/StoreBundle/Entity/ProductRepository.php
public function findOneByIdJoinedToCategory($id)
{
    $query = $this->getEntityManager()
        ->createQuery(
            'SELECT p, c FROM AcmeStoreBundle:Product p
            JOIN p.category c
            WHERE p.id = :id'
        )->setParameter('id', $id);
    try {
       return $query->getSingleResult();
    } catch (\Doctrine\ORM\NoResultException $e) {
        return null;
    }
}
```

Одоо, та controller –тоо дээрх функцийг дуудан, Product обектийг гарган авна.

```
public function showAction($id)
{
    $product = $this->getDoctrine()
        ->getRepository('AcmeStoreBundle:Product')
        ->findOneByIdJoinedToCategory($id);

    $category = $product->getCategory();

// ...

}
```

### Lifecycle Callback

Заримдаа, ямар нэг Entity –г хадгалах, засах, устгах үйлдэд хийхийн өмнө эсвэл дараа нь ямар нэг үйлдэл хийх хэрэг гардаг. Ийм төрлийн үйлдлүүдийг "lifecycle" callback гэж нэрлэдэг ба эдгээр нь entity-гийн үйлдэл (хадгалах, засах, устгах) -ээс хамааран тохирох утгыг буцаадаг method юм. Хэрэв та metadata –д annotation/тэмдэглэгээ/ ашиглаж байгаа бол lifecycle callback-ийг идэвхжүүлэх хэрэгтэй. Харин YAML эсвэл XML ашиглаж байгаа бол энэ үйлдлийг хийх шаардлагагүй.

```
/**
* @ORM\Entity()
* @ORM\HasLifecycleCallbacks()
*/
class Product
{
    // ...
}
```

Ингээд ямар нэг lifecycle үйлдэл хийнэ гэдэгийг Doctrine –д зааж өгнө. Жишээ нь, эхлээд entity хадгалагдах үед тухайн огноог **createdAt** багана руу хадгалдаг байя.

```php
/**
* @ORM\PrePersist
*/
public function setCreatedAtValue()
{
    $this->createdAt = new \DateTime();
}
```

Ингээд тухайн entity орохын өмнө Doctrine энэ функцийг автоматаар дуудан, createdAt талбарт тухайн огноог оноож өгөх болно.

9-р бүлэг

# Өгөгдлийн сан ба Propel

Өгөгдлийн сангаас мэдээллийг унших, өгөгдлийн сан руу мэдээллийг хадгалах нь програм бүтээхэд ашиглагддаг ерөнхий үйлдлүүдийн нэг юм. Symfony2 -ийг Propel –той нэгтгэх нь амархан. Propel –ийг хэрхэн суулгахыг Propel documentation –ний Working With Symfony2 хэсгээс үзнэ үү.

## Энгийн жишээ: Product

Энэ бүлэгт өгөгдлийн санг тохируулж, Product обектийг үүсгэн, тухайн өгөгдлийн сандаа мэдээлэл хадгалж, буцаан унших болно.

### Өгөгдлийн санг тохируулах

Эхлээд өгөгдлийн сангийн холболтын тохиргоог хийнэ. Энэ тохиргоо нь **app/config/parameters.yml** файлд хадгалагдана.

```
# app/config/parameters.yml
parameters:
    database_driver:    mysql
    database_host:      localhost
    database_name:      test_project
    database_user:      root
    database_password:  password
    database_charset:   UTF8
```

**parameters.yml** файлд тодорхойлсон эдгээр параметерүүдийг тохиргооны файл болох **config.yml** файлд оруулан ирнэ.

```
propel:
    dbal:
        driver:   "%database_driver%"
        user:     "%database_user%"
        password: "%database_password%"
        dsn:
```

```
7   "%database_driver%:host=%database_host%;dbname=%database_name%;charset=%dat
    abase_charset%"
```

Ингээд Propel-д өгөгдлийн сангаа зааж өгсөн болохоор Symfony2 –д өгөгдлийн санг үүсгэж болно.

```
1   $ php app/console propel:database:create
```

### Model класс үүсгэх

Propel-д **ActiveRecord** хэмээх классууд байх ба эдгээрийг **models** гэж нэрлэнэ. Эдгээр классуудыг business logic хэсгийг агуулж байдаг Propel үүсгэсэн байдаг.

Та бүтээгдэхүүний тухай мэдээллийг харуулах програм хийж байна гэж бодоё. Эхлээд, **AcmeStoreBundle** bundle-ийн **Resources/config** хавтсанд **schema.xml** файлыг үүсгэнэ.

```
1   <?xml version="1.0" encoding="UTF-8" ?>
2   <database name="default"
3       namespace="Acme\StoreBundle\Model"
4       defaultIdMethod="native"
5   >
6       <table name="product">
7           <column name="id"
8               type="integer"
9               required="true"
10              primaryKey="true"
11              autoIncrement="true"
12          />
13          <column name="name"
14              type="varchar"
15              primaryString="true"
16              size="100"
17          />
18          <column name="price"
19              type="decimal"
20          />
21          <column name="description"
22              type="longvarchar"
23          />
24       </table>
25  </database>
```

**schema.xml** файлыг үүсгэсний дараа доорх кодыг ажиллуулан model-оо үүсгэнэ.

```
1   $ php app/console propel:model:build
```

Энэ код нь **AcmeStoreBundle**-ийн **Model/** хавтсанд тухайн model классуудыг үүсгэнэ.

### Өгөгдлийн сангийн хүснэгт үүсгэх

Ингээд **Product** гэсэн класстай болсон бөгөөд энэ класс нь мэдээллийг хадгалхад ашиглагдана. Мэдээж өгөгдлийн сандаа тохирох **product** хүснэгтийг хараахан үүсгээгүй байгаа. Гэсэн ч Propel нь програмд ашиглаж байгаа **model** бүрт харгалзах хүснэгтүүдийг автоматаар үүсгэх боломжтой байдаг. Ингэхийн тулд доорх кодыг ажиллуулна.

```
$ php app/console propel:sql:build
$ php app/console propel:sql:insert --force
```

Ингээд  заасан баганууудыг агуулсан product хүснэгт бэлэн боллоо.

### Өгөгдлийн санд обект хадгалах

Product обект болон энэ обектод харгалзах product хүснэгтийг үүсгэсэн ба ингээд өгөгдлийн сан руу өгөгдөлөө хадгалахад бэлэн боллоо. Үүнийг Controller ашиглан хийнэ. Инэгхдээ Bundle-ийнхаа DefaultController руу дараах method-ийг нэмнэ.

```
// src/Acme/StoreBundle/Controller/DefaultController.php
// ...
use Acme\StoreBundle\Model\Product;
use Symfony\Component\HttpFoundation\Response;

public function createAction()
{
    $product = new Product();
    $product->setName('A Foo Bar');
    $product->setPrice(19.99);
    $product->setDescription('Lorem ipsum dolor');
    $product->save();

    return new Response('Created product id '.$product->getId());
}
```

Дээрх кодонд, $product обектыг үүсгэн, түүнд утгуудаа оноож өгөөд, дараа нь **save()** method-ийг дуудсанаар өгөгдлийн сан руу хадгалах болно.

### Өгөгдлийн сангаас обектийг авах

Өгөгдлийн сангаас обектийг авах нь мөн л хялбар байдаг.

```
// ...
use Acme\StoreBundle\Model\ProductQuery;
public function showAction($id)
{
    $product = ProductQuery::create()
        ->findPk($id);
    if (!$product) {
        throw $this->createNotFoundException(
            'No product found for id '.$id
        );
    }
// ... темплэйт руу $product обектийг дамжуулах гэх мэт үйлдлийг хийнэ
}
```

### Обектийг засах

Propel –ээс авсан обектдоо засвар хийнэ.

```
1  // ...
2  use Acme\StoreBundle\Model\ProductQuery;
3
4  public function updateAction($id)
5  {
6      $product = ProductQuery::create()
7          ->findPk($id);
8
9      if (!$product) {
10         throw $this->createNotFoundException(
11             'No product found for id '.$id
12         );
13     }
14
15     $product->setName('New product name!');
16     $product->save();
17
18     return $this->redirect($this->generateUrl('homepage'));
19 }
```

Обектийг засхад 3 үе шад дамжина.

1. Propel-оос обектийг авна. (6 - 13)
2. Обектод засвар хийнэ. (15)
3. Зассан обектоо хадгална. (16)

### Обект устгах

Обектийг устгах нь обектийг засахтай бараг адил. Харин save() –ийн оронд delete() method-ийг дуудах шаардлагатай.

```
1  $product->delete();
```

## Обекттой ажиллах query бичих

Propel нь ямар нэг ажиллагаагүйгээр query-нуудыг ажиллуулах **Query** хэмээх классыг агуулж байдаг.

```
1  \Acme\StoreBundle\Model\ProductQuery::create()->findPk($id);
2
3  \Acme\StoreBundle\Model\ProductQuery::create()
4      ->filterByName('Foo')
5      ->findOne();
```

Бүтээгдэхүүний үнийг авах query бичих хэрэгтэй боллоо гэж төсөөлөе. Гэхдээ 19.99  -ээс эхлээд, үнэтэй рүү өсөх дарааллаар гаргана. Controller-тоо дараах кодыг нэмнэ.

```
1  $products = \Acme\StoreBundle\Model\ProductQuery::create()
2      ->filterByPrice(array('min' => 19.99))
3      ->orderByPrice()
4      ->find();
```

Ингэснээр та SQL бичиж цаг үрэх хэрэггүй болж байгаа юм.

Хэрэв та, зарим query-г дахин ашиглахыг хүсвэл **ProductQuery** класс руу өөрийн үүсгэсэн method-ийг нэмэх хэрэгтэй.

```php
// src/Acme/StoreBundle/Model/ProductQuery.php
class ProductQuery extends BaseProductQuery
{
    public function filterByExpensivePrice()
    {
        return $this
            ->filterByPrice(array('min' => 1000));
    }
}
```

Гэхдээ Propel нь маш олон method-уудыг танд үүсгэж өгдөг. Доор ямар нэг ажиллагаагүйгээр findAllOrderedByName() –ийг хэрэгжүүлж байна.

```php
\Acme\StoreBundle\Model\ProductQuery::create()
    ->orderByName()
    ->find();
```

## Харилцаа/Холбоо

Та бүтээгдэхүүнийхээ тухай мэдээллийг категорит хийх хэрэгтэй байлаа гэж төсөөлөө. Энэ тохиолдолд, танд **Category** обект хэрэг болох бөгөөд **Product** обектийг **Category** обекттой холбох хэрэгтэй. Ингэхдээ эхлээд, **schema.xml** файлд category-ийн тодорхойлолтыг нэмнэ.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<database name="default"
    namespace="Acme\StoreBundle\Model"
    defaultIdMethod="native">
    <table name="product">
        <column name="id"
            type="integer"
            required="true"
            primaryKey="true"
            autoIncrement="true" />

        <column name="name"
            type="varchar"
            primaryString="true"
            size="100" />

        <column name="price"
            type="decimal" />
            <column name="description"
            type="longvarchar" />

        <column name="category_id"
            type="integer" />
```

```
24
25          <foreign-key foreignTable="category">
26              <reference local="category_id" foreign="id" />
27          </foreign-key>
28      </table>
29
30      <table name="category">
31          <column name="id"
32              type="integer"
33              required="true"
34              primaryKey="true"
35              autoIncrement="true" />
36          <column name="name"
37              type="varchar"
38              primaryString="true"
39              size="100" />
40       </table>
41  </database>
```

Классуудыг үүсгэхдээ,

```
1  $ php app/console propel:model:build
```

кодыг ажиллуулна.

Та өгөгдлийн сандаа бүтээгдэхүүнийхээ тухай мэдээллийг хадгалсан байгаа бөгөөд тэднийг устгахгүй байхыг хүсэж байгаа байх. Propel нь хуучин өгөгдлийг устгахгүйгээр таны өгөгдлийн санг шинэчлэх болно.

```
1  $ php app/console propel:migration:generate-diff
2  $ php app/console propel:migration:migrate
```

Ингээд таны өгөгдлийн сан шинэчлэгдсэн бөгөөд одоо та програмаа цааш нь үргэлжлүүлэх боломжтой.

### Холбоотой обектуудыг хадгалах

Дараах кодыг харая.

```
1  // ...
2  use Acme\StoreBundle\Model\Category;
3  use Acme\StoreBundle\Model\Product;
4  use Symfony\Component\HttpFoundation\Response;
5  class DefaultController extends Controller
6  {
7      public function createProductAction()
8      {
9          $category = new Category();
10         $category->setName('Main Products');
11         $product = new Product();
12         $product->setName('Foo');
13         $product->setPrice(19.99);
14         // product-ийг category-той холбоно
15         $product->setCategory($category);
16         // бүхэлд нь хадгална
17         $product->save();
```

```
18          return new Response(
19              'Created product id: '.$product->getId().' and category id:
20 '.$category->getId()
21          );
22      }
23 }
```

Ингээд **category** болон **product** хүснэгтүүдэд нэг мөр мэдээлэл орно. Шинээр нэмэгдсэн бүтээгдэхүүний **product.category_id** баганд шинээр нэмэгдсэн category-ийн **id** орох болно. Propel нь энэ харилцааг удирдан ажиллана.

### Холбоотой обектийг авах

Эхлээд **$product** обектийг авааД, түүнтэй холбоотой Categor луу хандана.

```
1  // ...
2  use Acme\StoreBundle\Model\ProductQuery;
3  public function showAction($id)
4  {
5      $product = ProductQuery::create()
6          ->joinWithCategory()
7          ->findPk($id);
8      $categoryName = $product->getCategory()->getName();
9  // ...
10 }
```

### Lifecycle Callbacks

Заримдаа, ямар нэг Entity –г хадгалах, засах, устгах үйлдэд хийхийн өмнө эсвэл дараа нь ямар нэг үйлдэл хийх хэрэг гардаг. Ийм төрлийн үйлдлүүдийг "lifecycle" буцаах гэж нэрлэдэг ба эдгээр нь entity-гийн үйлдэл (хадгалах, засах, устгах) -ээс хамааран тохирох утгыг буцаадаг method юм.

Класс руу шинэ method нэмнэ.

```
1  // src/Acme/StoreBundle/Model/Product.php
2  // ...
3  class Product extends BaseProduct
4  {
5      public function preInsert(\PropelPDO $con = null)
6      {
7          // обектийг хадгалахын өмнө ямар нэг үйлдэл хийнэ
8      }
9  }
```

Propel нь дараах hook-үүдийг агуулна.

- **preInsert()** код нь шинэ обект оруулахын өмнө биелэгдэнэ.
- **postInsert()** нь шинэ обект орсоны дараа биелэгдэнэ.
- **preUpdate()** нь хуучин обектийг шинэчлэхийн өмнө биелэгдэнэ.
- **postUpdate()** нь хуучин обектийг шинэчилсэний дараа биелэгдэнэ.
- **preSave()** нь обект (шинэ эсвэл хуучин байсан)-ыг хадгалхын өмнө биелэгдэнэ.
- **postSave()** нь  обект (шинэ эсвэл хуучин байсан)-ыг хадгалсаны дараа биелэгдэнэ.
- **preDelete()** нь обектийг устгахын өмнө биелэгдэнэ.
- **postDelete()** нь обектийг устгасны дараа биелэгдэнэ.

10-р бүлэг

# Тест хийх

Та код бичиж байх явцдаа ямар нэг алдаа гаргах магадлал их байдаг. Илүү сайн, найдвартай програм бүтээхийн тулд, функциональ болон нэгжийн тест 2-ийг ашиглан кодоо тестлэх хэрэгтэй.

## PHPUnit ашиглан тестлэх

Symfony2 нь прожектод шалгалт хийх үүрэгтэй **PHPUnit** гэх тусдаа санг агуулж байдаг. Нэгжийн болон Функциональ тестийн аль аль нь PHP класс бөгөөд таны bundle-ийн **Tests/** хавтсанд байрлана. Үүнийг ашиглахдаа дараах командаар програмын бүх тестийг ажиллуулна.

```
# specify the configuration directory on the command line
$ phpunit -c app/
```

**-c** тэмдэг нь тохиргооны файл байрлах **app/** хавтсанд ажиллана гэдэгийг PHPUnit-д зааж өгнө. Хэрэв та PHPUnit –ийн бүх тэмдэгүүдийн тухай сонирхож байгаа бол **app/phpunit.xml.dist** файлаас хараарай.

## Unit буюу нэгжийн тест

Нэгжийн тест нь ихэвчлэн тухайн заасан PHP классыг шалгахад ашиглагдана. Symfony2 нэгжийн тестийг бичих нь стандарт PHPUnit нэгжийн тест бичихтэй ижил байна. Жишээ нь, таны bundle-ийн **Utility/** хавтсанд **Calculator** класс байгаа гэж бодвол:

```
// src/Acme/DemoBundle/Utility/Calculator.php
namespace Acme\DemoBundle\Utility;
class Calculator
{
    public function add($a, $b)
    {
        return $a + $b;
    }
}
```

Энэ классыг шалгахдаа, bundle-ийн **Tests/Utility** хавтсанд **CalculatorTest** файлыг үүсгэнэ.

```php
// src/Acme/DemoBundle/Tests/Utility/CalculatorTest.php
namespace Acme\DemoBundle\Tests\Utility;

use Acme\DemoBundle\Utility\Calculator;

class CalculatorTest extends \PHPUnit_Framework_TestCase
{
    public function testAdd()
    {
        $calc = new Calculator();
        $result = $calc->add(30, 12);

        // assert that your calculator added the numbers correctly!
        $this->assertEquals(42, $result);
    }
}
```

Ингээд заасан файл, хавтасыг шалгахын тулд дараах командуудыг ажиллуулна:

```
# Utility хавтсанд бүх тестийг ажиллуулна
$ phpunit -c app src/Acme/DemoBundle/Tests/Utility/

# Calculator классд бүх тестийг ажиллуулна
$ phpunit -c app src/Acme/DemoBundle/Tests/Utility/CalculatorTest.php

# Bundle-ийг бүхэлд нь шалгах тестийг ажиллуулна
$ phpunit -c app src/Acme/DemoBundle/
```

## Функциональ тест

Функциональ тест нь програмын өөр өөр хэсгүүдийг нэгтгэн шалгадаг. Үүнийг дараах дарааллын дагуу хийж гүйцэтгэнэ.

- Request үүсгэх
- Response-ийг шалгах
- Холбоос буюу link дээр дарах, эсвэл формын мэдээлэл илгээх
- Response-ийг шалгах
- Алдааг засаад дахин давтах

### Таны анхны функциональ тест

Функциональ тест бол энгийн PHP функц бөгөөд таны програмын bundle-ийн **Tests/Controller** хавтсан дотор байрлана. Хэрэв та **DemoController** классаар үүсгэгдсэн хуудсуудыг шалгахыг хүсвэл **WebTestCase** хэмээх түсгай классаас удамшсан **DemoControllerTest.php** файлыг шинээр үүсгэнэ.

Жишээ нь: Symfony2 Standard Edition нь DemoController –т зориулсан энгийн функциональ тестийг ажиллуулж байна.

```php
// src/Acme/DemoBundle/Tests/Controller/DemoControllerTest.php
namespace Acme\DemoBundle\Tests\Controller;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

class DemoControllerTest extends WebTestCase
{
    public function testIndex()
    {
        $client = static::createClient();
        $crawler = $client->request('GET', '/demo/hello/Fabien');
        $this->assertGreaterThan(
            0,
            $crawler->filter('html:contains("Hello Fabien")')->count()
        );
    }
}
```

Функциональ тестийг ажиллахад, WebTestCase класс нь таны програмын kernel-ийг ажиллагаанд оруулна. Ихэнх тохиолдолд энэ нь автоматаар хийгддэг. Гэхдээ, хэрэв таны kernel стандарт бус хавтсанд байгаа бол та **phpunit.xml.dist** файлд өөрчлөлт хийх хэрэгтэй.

Listing 1

```xml
<phpunit>
    <!-- ... -->
    <php>
        <server name="KERNEL_DIR" value="/path/to/your/app/" />
    </php>
    <!-- ... -->
</phpunit>
```

**createClient()** method нь таны сайтыг харуулахад ашиглагдах браузертай адил визуал клэйнтийг буцаадаг.

```php
$crawler = $client->request('GET', '/demo/hello/Fabien');
```

**request()** method **Crawler**[32] обектийг буцаадаг. Энэ обект нь Response дахь элементүүдийг сонгох, линк дээр дарах, формоос мэдээлэл илгээх зэрэгт ашиглагдана.

Тухайн response нь XML эсвэл HTML байгаа үед л Crawler нь ажиллана. Response-ийн контентийг авахдаа **$client->getResponse()->getContent()** –ийг дуудна.

Хамгийн эхэнд сонгогдсон link дээр дархад Crawler нь XPath expression эсвэл CSS selector-ийг ашигладаг ба дараа нь Client-ийг ашиглана. Жишээ нь: дараах код **Greet** хэмээх үгтэй бүх link-үүдийг хайж олоод аль нэгийг нь сонгоод түүн дээр click хийнэ.

```php
$link = $crawler->filter('a:contains("Greet")')->eq(1)->link();

$crawler = $client->click($link);
```

[32] http://api.symfony.com/master/Symfony/Component/DomCrawler/Crawler.html

Формын мэдээлэл илгээх нь үүндтэй бараг адилхан.

```
$form = $crawler->selectButton('submit')->form();

// утга онооно
$form['name'] = 'Lucas';
$form['form_name[subject]'] = 'Hey there!';

// формыг илгээнэ
$crawler = $client->submit($form);
```

Ингээд та прогрмаа хялбархан удирдах боломжтой болж, хүссэн бүхнээ тест хийхэд ашиглаж болно. Crawler ашиглаад DOM обектод шалгалт хийе.

```
// Assert that the response matches a given CSS selector.
$this->assertGreaterThan(0, $crawler->filter('h1')->count());
```

Эсвэл та response-ийн обектийг шууд тестлэх боломжтой.

```
$this->assertRegExp(
    '/Hello Fabien/',
    $client->getResponse()->getContent()
);
```

**Хэрэгцээт шалгалтууд**

Listing

```
use Symfony\Component\HttpFoundation\Response;
// ...
// Assert that there is at least one h2 tag
// with the class "subtitle"
$this->assertGreaterThan(
0,
$crawler->filter('h2.subtitle')->count()
);
// Assert that there are exactly 4 h2 tags on the page
$this->assertCount(4, $crawler->filter('h2'));
// Assert that the "Content-Type" header is "application/json"
$this->assertTrue(
$client->getResponse()->headers->contains(
'Content-Type',
'application/json'
)
);
// Assert that the response content matches a regexp.
$this->assertRegExp('/foo/', $client->getResponse()->getContent());
// Assert that the response status code is 2xx
$this->assertTrue($client->getResponse()->isSuccessful());
// Assert that the response status code is 404
$this->assertTrue($client->getResponse()->isNotFound());
// Assert a specific 200 status code
$this->assertEquals(
Response::HTTP_OK,
```

Listing

```
27    $client->getResponse()->getStatusCode()
28    );
29    // Assert that the response is a redirect to /demo/contact
30    $this->assertTrue(
31    $client->getResponse()->isRedirect('/demo/contact')
32    );
33    // or simply check that the response is a redirect to any URL
34    $this->assertTrue($client->getResponse()->isRedirect());
```

HTTP status code constant-ийг Symfony 2.4-т танилцуулсан

## Test Client –тэй ажиллах

Test Client нь вэб браузертай ижил HTTP client –ийг дүрслэн үзүүлэх бөгөөд Symfony2 програмд request-ийг үүсгэнэ.

```
1  $crawler = $client->request('GET', '/hello/Fabien');
```

**request()** method нь HTTP method болон URL-ийг авч, Crawler-ийг буцаадаг.

Crawler нь Response дахь DOM элементүүдийг хайхад ашиглагдана. Эдгээр элементүүд нь link дээр дарах болон формоос мэдээлэл илгээхэд ашиглагдаж байдаг.

```
1  $link = $crawler->selectLink('Go elsewhere...')->link();
2  $crawler = $client->click($link);
3
4  $form = $crawler->selectButton('validate')->form();
5  $crawler = $client->submit($form, array('name' => 'Fabien'));
```

**click()** болон **submit()** method-ууд нь 2-уулаа Crawler обектийг буцаадаг.

**request** method мөн формын мэдээлэл шууд илгээх, эсвэл илүү цогц request-ийг биелүүлхэд ашиглагдана.

```
1  // Directly submit a form (but using the Crawler is easier!)
2  $client->request('POST', '/submit', array('name' => 'Fabien'));
3
4  // Submit a raw JSON string in the request body
5  $client->request(
6      'POST',
7      '/submit',
8      array(),
9      array(),
10     array('CONTENT_TYPE' => 'application/json'),
11     '{"name":"Fabien"}'
12 );
13
14 // Form submission with a file upload
15 use Symfony\Component\HttpFoundation\File\UploadedFile;
16
17 $photo = new UploadedFile(
18     '/path/to/photo.jpg',
```

```
    'photo.jpg',
    'image/jpeg',
    123
);
$client->request(
    'POST',
    '/submit',
    array('name' => 'Fabien'),
    array('photo' => $photo)
);

// Perform a DELETE requests, and pass HTTP headers
$client->request(
    'DELETE',
    '/post/12',
    array(),
    array(),
    array('PHP_AUTH_USER' => 'username', 'PHP_AUTH_PW' => 'pa$$word')
);
```

### Browsing

Client нь жинхэнэ браузерт хийгддэг олон тооны үйлдлүүдийг гүйцэтгэх боломжтой.

```
$client->back();
$client->forward();
$client->reload();

// бүх cookie болон history-ийг устгана
$client->restart();
```

### internal Object-руу хандах

**getInternalRequest()** болон **getInternalResponse()** method-уудийг Symfony 2.3-т танилцуулсан.

Хэрэв та программаа тест хийхдээ client ашиглаж байгаа бол client-ийн internal object руу хандах боломжтой.

```
$history = $client->getHistory();
$cookieJar = $client->getCookieJar();
```

Мөн хамгийн сүүлчийн request-тэй холбоотой обектуудыг авах бол:

```
// the HttpKernel request instance
$request = $client->getRequest();
// the BrowserKit request instance
$request = $client->getInternalRequest();
// the HttpKernel response instance
$response = $client->getResponse();
// the BrowserKit response instance
$response = $client->getInternalResponse();
$crawler = $client->getCrawler();
```

### Container луу хандах

Энэ нь response-т тест хийхэд маш их хэрэгтэй функциональ тест юм. Гэвч үнэхээр ховор тохиолдолд ашиглагдана. Тест бичиж байхдаа internal object-руу хандахыг хүсвэл дараах байдлаах хандана:

```
1  $container = $client->getContainer();
```

### Profiler Data руу хандах

Request –ийг боловсруулах тухай мэдээллийг цуглуулхын тулд Symfony profiler –ийг идэвхтэй болгох хэрэгтэй. Жишээлбэл: profiler нь тухайн хуудсыг ачааллах үед өгөгдлийн сангийн олон тооны query ажиллуулж шалгахад ашиглагдана.

Profiler ашиглан хамгийн сүүлчийн query-г дараах аргаар авна.

```
1  // enable the profiler for the very next request
2  $client->enableProfiler();
3
4  $crawler = $client->request('GET', '/profiler');
5
6  // get the profile
7  $profile = $client->getProfile();
```

### Redirecting

Request нь Redirect response буцаахад client нь үүнийг автоматаар дагадаггүй. Иймд Response-ийг шалгаад, **followRedirect()** method-оор Redirection-г хүчээр хийж болно.

```
1  $crawler = $client->followRedirect();
```

Харин та бүх Redirect-ийг автоматаар биелүүлхийг хүсвэл **followRedirects()** method ашиглах хэрэгтэй.

```
1  $client->followRedirects();
```

## Crawler

**Crawler** нь Client request гаргах бүрт түүнийг буцааж байдаг. Энэ нь HTML болон сонгосон node-ийг судлах, link болон формуудыг хайх зэрэгт ашиглагдана.

### Traversing

jQuery-тэй адилаар Crawler нь HTML/XML –ийн DOM-той харилцах method-ийг агуулдаг. Жишээ нь: доорх жишээнд, бүх **input[type=submit]** элементийг хайн, тухайн хуудсан дээрх хамгийн сүүлчийн нэг элементийг сонгоод, тэр элементийн эцэг элементийг шууд сонгоно.

```
1  $newCrawler = $crawler->filter('input[type=submit]')
2      ->last()
3      ->parents()
4      ->first()
5  ;
```

Үүнээс гадна өөр олон method байдаг.

| Method | Тайлбар |
|---|---|
| filter('h1.title') | Өгсөн CSS selector-той тохирох node-үүдийг |
| filterXpath('h1') | Өгсөн Xpath илэрхийлэлтэй тохирох- node-үүдийг |
| eq(1) | Заасан индексд тохирох node-ийг |
| first() | Эхний node-ийг |
| last() | Сүүлийн node-ийг |
| siblings() | |
| nextAll() | |
| previousAll() | |
| parents() | |
| children() | |
| reduce($lambda) | |

Эдгээр бүх method бүр шинэ Crawler-ийг буцаадаг.

### Мэдээллийг задлах

Crawler ашиглан node-ээс мэдээллийг задлан авах боломжтой.

```
// Эхний node-ийн аттрибутыг буцаана.
$crawler->attr('class');

// эхний node-ийн node утгыг буцаана.
$crawler->text();

// Бүх node-ийн аттрибутуудын массивийг задлана
// (_text node-ийн утгыг буцаана)
// Crawler дахь элемент бүрийн массивыг буцаана
$info = $crawler->extract(array('_text', 'href'));

// Executes a lambda for each node and return an array of results
$data = $crawler->each(function ($node, $i) {
    return $node->attr('href');
});
```

### Link

link сонгохдоо, дээрх method-уудаас гадна **selectLink()** –ийг ашиглаж болно.

```
$crawler->selectLink('Click here');
```

Энэ нь өгсөн тексктийг агуулж бүй бүх link, эсвэл өгсөн текстийг агуулсан **alt** аттрибуттай зургийг сонгоно.

link-ийг нэг удаа сонгосон бол **Link** хэмээх тусгай обект руу хандах боломжтой. Энэ нь link-тэй ажиллах тусгай method (**getMethod() and getUri()** гэх мэт) -уудыг агуулдаг. Тухайн link дээр дархад **Client**-ийн **click()** method ашиглагдах ба түүнийгээ **Link** обект руу дамжуулна.

```
$link = $crawler->selectLink('Click here')->link();

$client->click($link);
```

### Form

Link-тэй адилаар **selectButton()** method ашиглан формыг сонгоно.

```
$buttonCrawlerNode = $crawler->selectButton('submit');
```

**selectButton()** method нь **button** tag-уудыг болон **input** tag-уудыг сонгоно. Тэдгээрийг хайхдаа button-ний өөр өөр хэсгүүдийг ашиглаж болно.

- **value** аттрибутын утгаар
- image tag-ийн **id, alt** атрибутын утгаар
- button tag-ийн **id, name** аттрибутын утгаар

button-ийг сонгосон бол **Form** обектийг авах **form()** method-ийг дуудна.

```
$form = $buttonCrawlerNode->form();
```

**form()** method дуудагдахад, field-үүдийн утгыг массив хэлбэрээр дамжуулж болно.

```
$form = $buttonCrawlerNode->form(array(
    'name' => 'Fabien',
    'my_form[subject]' => 'Symfony rocks!',
));
```

Мөн хэрэв та формд HTTP method-ийг зааж үзүүлхийг хүсвэл иүүнд 2 дахь аргументийг дамжуулна.

```
$form = $buttonCrawlerNode->form(array(), 'DELETE');
```

Client нь Form обектийг илгээнэ.

```
$client->submit($form);
```

Field-ийн утгыг **submit()** method-ийн 2 дахь аргумент болгон дамжуулна.

```
$client->submit($form, array(
    'name' => 'Fabien',
    'my_form[subject]' => 'Symfony rocks!',
));
```

## Тест хийх тохиргоо

Client нь функциональ тестийг ашиглан **test** environment-д ажиллах зорилготой Kernel –ийг үүсгэдэг. **Test** environment-д **app/config/config_test.yml** файлыг дуудсанаар тест хийх зорилгоор програмынхаа тохиргоонуудтай ажиллах боломжтой болно.

### PHPUnit тохиргоо

Програм болгонд өөрийн гэсэн PHPUnit тохиргоо байх ба энэ нь **app/phpunit.xml.dist** файлд бичигдсэн байдаг. Та энэ файлыг өөрчилөх эсвэл локал машин дээр тохиргоо хийх **app/phpunit.xml** файлыг үүсгэж бас болно.

Default-аар, **src/*/*Bundle/Tests** эсвэл **src/*/Bundle/*Bundle/Tests** гэсэн стандарт хавтсанд байх bundle-аас тест хийхдээ **app/phpunit.xml.dist** файлд тохируулсанаар **phpunit** командыг ажиллуулна.

```
<!-- app/phpunit.xml.dist -->
<phpunit>
```

```
    <!-- ... -->
    <testsuites>
        <testsuite name="Project Test Suite">
            <directory>../src/*/*Bundle/Tests</directory>
            <directory>../src/*/Bundle/*Bundle/Tests</directory>
        </testsuite>
    </testsuites>
    <!-- ... -->
</phpunit>
```

Гэхдээ бас энд өөр хавтас нэмэх бүрэн боломжтой. Доорх жишээнд, хэрэглэгчийн үүсгэсэн **lib/tests** хавтаснаас тест нэмхээр тохируулсан байна.

```
<!-- app/phpunit.xml.dist -->
<phpunit>
    <!-- ... -->
    <testsuites>
        <testsuite name="Project Test Suite">
            <!-- ... --->
            <directory>../lib/tests</directory>
        </testsuite>
    </testsuites>
    <!-- ... --->
</phpunit>
```

Кодонд хамаарах бусад хавтаснуудыг оруулж ирэхдээ **<filter>** хэсгийг засах хэрэгтэй.

```
<!-- app/phpunit.xml.dist -->
<phpunit>
    <!-- ... -->
    <filter>
        <whitelist>
            <!-- ... -->
            <directory>../lib</directory>
            <exclude>
            <!-- ... -->
                <directory>../lib/tests</directory>
            </exclude>
        </whitelist>
    </filter>
    <!-- ... --->
</phpunit>
```

11-р бүлэг

# Validation /баталгаажуулалт/

Validation бол вэб програмд хийгддэг үнэхээр чухал үйлдлүүдийн нэг юм. Энэ нь формд оруулж байгаа мэдээллийг баталгаажуулахад хэрэглэгдэх ба мөн тухайн өгөгдлийг өгөгдлийн санд хадгалах, эсвэл вэб сервис рүү дамжуулахын өмнө баталгаажуулсан байх хэрэгтэй байдаг.

Энэ үйлдлийг Symfony2 –ийн **Validator** компонентийг ашиглан хялбархан гүйцэтгэнэ. Энэ компонент нь **JSR303 Bean Validation**[33] дээр үндэслэгдэн бүтээгдсэн байна.

## Validation –ний тухай үндсэн ойлголт

Validation-ий тухай судлахын тулд түүнийг хийж үзье. Таны програмд ашиглагдах нэг PHP обект байлаа гэж бодоё.

```
1  // src/Acme/BlogBundle/Entity/Author.php
2  namespace Acme\BlogBundle\Entity;
3
4  class Author
5  {
6      public $name;
7  }
```

Энэ нь таны програм дотор ямар нэг үйлдэл гүйцэтгэх зорилготой энгийн л нэг класс байна. Validation-ний үндсэн зарчим бол тухайн обектийн өгөгдөл зөв байгаа эсэхийг шалгах. Үүнийг хийхийн тулд, та тухайн обект яг ямар байх ёстой дүрэм (**constraint** гэж нэрлэдэг)-үүдийг тохируулах хэрэгтэй. Эдгээр дүрэмүүд нь олон янзын форматтай файлд (YAML, XML, annotations, PHP) байж болно.

Жишээ нь: дээрх **$name** property нь хоосон байж болохгүй гэсэн дүрэм заахдаа дараах арыг ашиглана.

```
1  # src/Acme/BlogBundle/Resources/config/validation.yml
2  Acme\BlogBundle\Entity\Author:
3      properties:
4          name:
5              - NotBlank: ~
```

[33] http://jcp.org/en/jsr/detail?id=303

protected болон private, мөн getter method зэргийг validate хийж болно.

### `Validator` Service ашиглах

Одоо validator service-ийн **validate** method-ийг ашиглан **Author** обектийг баталгаажуулая. Validator нь классын constraint-уудыг уншаад, тухайн обектийн өгөгдөл constraint-уудтай тохирч байгаа эсэхийг шалгана. Хэрвээ тохирохгүй бол алдааны мэдээлэл буцаана. Дараах жишээг харая.

```
// ...
use Symfony\Component\HttpFoundation\Response;
use Acme\BlogBundle\Entity\Author;

public function indexAction()
{
    $author = new Author();
    // ... do something to the $author object

    $validator = $this->get('validator');
    $errors = $validator->validate($author);

    if (count($errors) > 0) {
        /*
        * Uses a __toString method on the $errors variable which is a
        * ConstraintViolationList object. This gives us a nice string
        * for debugging
        */
        $errorsString = (string) $errors;

        return new Response($errorsString);
    }

    return new Response('author зөв байна!');
}
```

Хэрэв **$name** property хоосон байвал дараах алдааны мессеж танд харагдах болно.

```
Acme\BlogBundle\Author.name:
    Энэ талбар хоосон байж болохгүй.
```

Харин **name** property –руу утга оруулвал амжилттай болсон тухай мессеж харагдана.

Ихэнх тохиолдолд формын өгөгдлийг боловсруулхад validation-ийг шууд ашиглана.

Мөн темплэйт рүү алдааны тухай мессежийг дамжуулж болно.

```
if (count($errors) > 0) {
    return $this->render('AcmeBlogBundle:Author:validate.html.twig', array(
        'errors' => $errors,
    ));
}
```

Харин темплэйт дотор та яг хэрэгтэй алдааны мэдээллийг харуулахдаа дараах байдлаар бичиж болно.

```twig
{# src/Acme/BlogBundle/Resources/views/Author/validate.html.twig #}
<h3>The author has the following errors</h3>
<ul>
{% for error in errors %}
    <li>{{ error.message }}</li>
{% endfor %}
</ul>
```

### Validation ба Form

Validator service нь ямар нэг обектыг баталгаажуулхад ашигладдаг. Гэсэн хэдий ч формтой ажиллаж байх үед validator –той ихэвчлэн ажилладаг. Symfony-ийн форм сан нь формоос утга илгээсний дараа тухайн обектийг шалгахдаа validator сервесийг ашигладаг. Тухайн обекттой constraint/дүрэм/ тохирохгүй бол обект нь **FieldError** обект руу хөрвүүлэгдэн дэлгэцэнд харуулна. Формоос мэдээлэл илгээх ажил нь дараах байдлаар хийгдэнэ.

```php
// ...
use Acme\BlogBundle\Entity\Author;
use Acme\BlogBundle\Form\AuthorType;
use Symfony\Component\HttpFoundation\Request;

public function updateAction(Request $request)
{
    $author = new Author();
    $form = $this->createForm(new AuthorType(), $author);

    $form->handleRequest($request);

    if ($form->isValid()) {
        // the validation passed, do something with the $author object
        return $this->redirect($this->generateUrl(...));
    }

    return $this->render('BlogBundle:Author:form.html.twig', array(
        'form' => $form->createView(),
    ));
}
```

Дээрх жишээнд **AuthorType** класс ашиглаж байгаа ба тэр классыг энд үзүүлээгүй байна.

## Тохиргоо

Symfony2 validator нь Default-аар идэвхтэй байдаг ч хэрэв та constraint-уудыг заахдаа **annotation** method ашиглаж байшаа бол annotation-уудыг идэвхтэй болгох ёстой.

```
# app/config/config.yml
framework:
    validation: { enable_annotations: true }
```

## Constraint

Validator нь constraint-уудыг ашиглан обектуудыг баталгаажуулах зорилготой юм. Обектийг баталгаажуулахдаа нэг классд constraint-уудаа тодорхойлоод, түүнийгээ validator сервес рүү дамжуулна.

Үнэн хэрэгтээ, constraint-ууд нь энгийн PHP обект юм. Symfony2-т constraint-үүд нь бүгд тухайн нөхцөл үнэн байх үед л тухайн өгөгдлийг баталгаажуулна.

### Symfony-д ашиглагддаг Constraint-ууд

Symfony2 package нь хэрэгцээтэй олон тооны constraint-уудыг агуулдаг. Үүнд:

#### Үндсэн Constraint-ууд

- NotBlank
- Blank
- NotNull
- Null
- True
- False
- Type

#### Тескттэй ажиллах Constraint-ууд

- Email
- Length
- Url
- Regex
- Ip
- Uuid

#### Тоотой ажиллах Constraint-ууд

- Range

### Харицуулах Constraint-ууд

• EqualTo

• NotEqualTo

• IdenticalTo

• NotIdenticalTo

• LessThan

• LessThanOrEqual

• GreaterThan

• GreaterThanOrEqual

### Огноотой ажиллах Constraint-ууд

• Date

• DateTime

• Time

### Collection Constraint-ууд

• Choice

• Collection

• Count

• UniqueEntity

• Language

• Locale

• Country

### Файлтай ажиллах Constraint-ууд

• File

• Image

### Санхүүгийн болон бусад тоон өгөгдөлтэй ажиллах Constraint-ууд

• CardScheme

• Currency

• Luhn

• Iban

• Isbn

• Issn

### Бусад Constraint-ууд

• Callback

• Expression

• All

• UserPassword

• Valid

Эдгээрээс гадна та өөрийн constraint-ийг үүсгэн ашиглах бас боломжтой.

### Constraint –ийн тохиргоо

Энд мөн **Choice constraint** гэж байдаг ба энэ нь хэд хэдэн тохиргооны сонголтуудыг агуулдаг. Жишээ нь Author классд gender property-д "**male**" эсвэл "**female**" 2 утгын аль нэгийг авах хэрэгтэй байг.

```
# src/Acme/BlogBundle/Resources/config/validation.yml
Acme\BlogBundle\Entity\Author:
     properties:
        gender:
             - Choice: { choices: [male, female], message: Choose a valid
gender. }
```

Constraint-ийн сонголтууд нь массив хэлбэрээр дамжина. Гэхдээ зарим үед нэг л утга дамжуулах хэрэгтэй болдог. Энэ тохиолдолд дараах аргаар **choice** сонголтыг заана.

```
# src/Acme/BlogBundle/Resources/config/validation.yml
Acme\BlogBundle\Entity\Author:
    properties:
        gender:
            - Choice: [male, female]
```

## Constraint ашиглах обект

Constraint нь классын **property** (name гэх мэт) эсвэл public хандалттай **getter** method (getFullName гэх мэт) –д хэрэглэгдэнэ.

### Property

Классын property-нуудыг баталгаажуулах нь validation-ний үндсэн арга юм. Symfony2 т **private**, **protected**, **public** хандалттай property-д **validation** хийх боломжтой. Дараах жишээнд, **Author** классын **$firstName** property хамгийн багадаа 3 тэмдэгт авах хэрэгтэй гэдэгийг хэрхэн тохируулахыг үзүүллээ.

```
# src/Acme/BlogBundle/Resources/config/validation.yml
Acme\BlogBundle\Entity\Author:
    properties:
        firstName:
            - NotBlank: ~
            - Length:
                min: 3
```

### Getter

Constraint нь мөн method-ийн утгыг бүцаахад хэрэглэгдэж болно. Symfony2 –т "**get**", "**is**", "**has**" гэсэн үгнүүдээр эхэлсэн нэртэй ямар ч method руу constraint нэмж болно. Ийм төрлийн method –уудыг "**getters**" хэмээн нэрлэдэг.

Энэ аргын давуу тал нь обектод динамикаар validate хийх боломжыг олгодог. Тухайлбал, нууц үг авдаг талбарт тухайн хэрэглэгчийн нэрийг ашиглахыг хориглох (хамгаалалтыг сайжруулах зорилгоор) шаардлагатай байлаа. Тэгвэл та **isPasswordLegal** method үүсгэж болох ба тухайн утгууд ялгаатай байгаа үед л **true** утга бүцаах ёстой.

```
# src/Acme/BlogBundle/Resources/config/validation.yml
Acme\BlogBundle\Entity\Author:
   getters:
       passwordLegal:
         - "True": { message: "Нууц үгэнд таны нэр байж болохгүй!" }
```

Одоо **isPasswordLegal()** method –оо үүсгэе.

```
public function isPasswordLegal()
{
     return $this->firstName != $this->password;
}
```

### Класс

Зарим constraint-ууд классыг бүхэлд нь validate хийхэд хэрэглэгдэж болно. Жишээ нь, *Callback* constraint тухайн класст хэрэглэгдэнэ. Тухайн классыг баталгаажуулахад тухайн constraint –оор заагдсан method-ууд нь биелэгдэнэ.

## Validation бүлгүүд

Зарим тохиолдолд, тухайн нэг класст зөвхөн хэдхэн constraint –оор обектийг баталгаажуулах хэрэгтэй үе байдаг. Энэ үед, constraint-уудыг нэг эсвэл хэд хэдэн **validation group** буюу бүлэг болгон зохион байгуулж болох ба дараа нь тухайн нэг бүлгийг дуудан ашиглаж болдог.

Жишээ нь, **User** гэсэн класс хэрэглэгчийг бүртгэх болон хэрэглэгчийн мэдээллийг засдаг байлаа гэж бодоё.

```
# src/Acme/BlogBundle/Resources/config/validation.yml
Acme\BlogBundle\Entity\User:
    properties:
        email:
            - Email: { groups: [registration] }
        password:
            - NotBlank: { groups: [registration] }
            - Length: { min: 7, groups: [registration] }
        city:
            - Length:
                min: 2
```

Энэ тохирогооны файлд 3 төрлийн Validation group тодорхойлсон байна.

- **Default-** Тухайн класс дахь constraint-уудыг агуулах ба өөр ямарч бүлэгт хамаарагдахгүй бүх классуудыг заана.
- **User-** Default бүлэг дахь **User** обектийн бүх constraint-уудтай тэнцүүлнэ.
- **registration –** зөвхөн email болон password гэсэн талбаруудын constraint-ыг агуулна.

Заасан бүлэгийг ашиглахдаа тухайн бүлэгүүдийн нэрийг **validate()** method-ийн 2 дахь аргументэд зааж өгнө.

```
1  $errors = $validator->validate($author, array('registration'));
```

Хэрэв ямар нэг бүлэг заагдаагүй бол бүх constraint Default-ийг ашиглана.

## Бүлгүүдийн дараалал

Зарим тохиолдолд бүлэгүүдийг дараалан хэрэгжүүлэх шаардлага гардаг. Энэ үед, **GroupSequence** ашиглах ба ямар дарааллаар тухайн бүлэгүүдийг биелэгдэх ёстойг зааж өгнө.

Жишээ нь: **User** класст хэрэглэгчийн нэр, нууц үг нь бусад бүх validation-аас өөрөөр validation хийх хэрэгтэй бол дараах байдлаах хийж болно (алдааны олон мессеж гаргахаас сэргийлэх зорилгоор):

```
1  # src/Acme/BlogBundle/Resources/config/validation.yml
2  Acme\BlogBundle\Entity\User:
3      group_sequence:
4          - User
5          - Strict
6      getters:
7          passwordLegal:
8              - "True":
9                  message: "Нууц үгэнд хэрэглэгчийн нэрийг оруулж болохгүй"
10                 groups: [Strict]
11     properties:
12         username:
13             - NotBlank: ~
14         password:
15             - NotBlank: ~
```

Энэ жишээнд, хамгийн эхлээд **User** бүлэгийн бүх constraint-уудыг validate хийнэ. Хэрэв тэр бүлэгийн бүх constraint баталгаажвал **Strict** гэсэн 2 дахь бүлгийг validate хийнэ.

### Group Sequence Provider буюу дараалал зохицуулагч

**User** класс нь энгийн болон төлбөртэй хэрэглэгчтэй ажилладаг байг. Тэгэхээр төлбөртэй хэрэглэгч орж ирсэн тохиолдолд **user** класс руу зарим нэмэлт constraint (кредит картны мэдээлэл гэх мэт) нэмж өгөх шаардлагатай. Тэгэхээр тухайн үед аль бүлэгийг идэвхжүүлэх шаардлагатай эсэхийг динамикаар заахдаа **Group Sequence Provider** –ийг ашиглана. Үүний тулд, эхлээд entity-гээ үүсгээд **Premium** гэх шинэ бүлэг үүсгэнэ.

```
1  # src/Acme/DemoBundle/Resources/config/validation.yml
2  Acme\DemoBundle\Entity\User:
3      properties:
```

```
        name:
            - NotBlank: ~
        creditCard:
            - CardScheme:
                schemes: [VISA]
                groups: [Premium]
```

Одоо, **GroupSequenceProviderInterface** –ээс **User** классыг удамшуулж, **getGroupSequence()** method нэмнэ.

```php
// src/Acme/DemoBundle/Entity/User.php
namespace Acme\DemoBundle\Entity;

// ...
use Symfony\Component\Validator\GroupSequenceProviderInterface;

class User implements GroupSequenceProviderInterface
{
    // ...
    public function getGroupSequence()
    {
        $groups = array('User');
        if ($this->isPremium()) {
            $groups[] = 'Premium';
        }
        return $groups;
    }
}
```

Дараа нь, бүлгийн дарааллыг агуулж байгаа **User** класстаа **Validator** компонентийг зарлана.

```yaml
# src/Acme/DemoBundle/Resources/config/validation.yml
Acme\DemoBundle\Entity\User:
    group_sequence_provider: ~
```

### Утга болон массивт Validate хийх

Зарим үед, энгийн нэг утгыг **validate** хийх хэрэгтэй болдог. Жишээ нь, мэйл хаягийн бүтцийг шалгах. Үүнийг хэрхэн хийхийг дараах кодоос харна уу.

```php
use Symfony\Component\Validator\Constraints\Email;
// ...
public function addEmailAction($email)
{
    $emailConstraint = new Email();
    // all constraint "options" can be set this way
    $emailConstraint->message = 'email хаяг буруу байна';

    // use the validator to validate the value
    $errorList = $this->get('validator')->validateValue(
        $email,
        $emailConstraint
    );
    if (count($errorList) == 0) {
        // this IS a valid email address, do something
    } else {
```

```
17          // this is *not* a valid email address
18          $errorMessage = $errorList[0]->getMessage();
19          // ... do something with the error
20      }
21      // ...
22  }
```

**validateValue** –г дуудан түүндээ шалгах гэж буй утга болон тухайн утгыг баталгаажуулах constraint обектийг дамжуулна.

**validateValue** method нь **ConstraintViolationList** обектийг буцаах ба энэ обект нь алдаануудын жагсаалтыг агуулсан массив юм. Массив дахь алдааны мессеж бүр нь **ConstraintViolation** обект байх бөгөөд үүнийг **getMessage** method –оор тухайн алдааны мессежийг авна.

12-р бүлэг

# Форм

Энэ бүлэгт энгийн формоос эхлээд илүү цогц форм хүртэл байгуулж, форм сангийн чухал хэсгүүдийн талаар суралцах болно.

## Энгийн форм үүсгэх

Тухайн хэрэглэгч өөрийн хийх ажлын дарааллаа тэмдэглэдэг, ажлынхаа тухай мэдээллийг засах, шинэ ажил нэмж оруулах гэх мэт үйлдэлтэй нэг жижиг програм хийж байлаа гэж бодоё. Тэгэхээр та форм байгуулах хэрэгтэй болно. Гэхдээ форм байгуулж эхлэхийнхээ өмнө эхлээд тухайн ажлыг хадгалах, хэрэглэгчид ажлын жагсаалтыг харуулах үүрэгтэй Task гэсэн класст үүсгэе.

```php
// src/Acme/TaskBundle/Entity/Task.php
namespace Acme\TaskBundle\Entity;
class Task
{
    protected $task;
    protected $dueDate;

    public function getTask()
    {
        return $this->task;
    }
    public function setTask($task)
    {
        $this->task = $task;
    }
    public function getDueDate()
    {
        return $this->dueDate;
    }
    public function setDueDate(\DateTime $dueDate = null)
    {
       $this->dueDate = $dueDate;
    }
}
```

Хэрэв та энэ жишээг дагаж хийх бол эхлээд дараах командаар **AcmeTaskBundle** –ийг үүсгэх хэрэгтэй.

```
$ php app/console generate:bundle --namespace=Acme/TaskBundle
```

### Форм байгуулах

Ингээд Task классыг үүсгэлээ. Дараагийн алхам бол HTML форм үүсгэн харуулах. Symfony2-т форм үүсгэхдээ форм обектийг байгуулж, түүнийгээ темплэйт рүү илгээнэ. Үүнийг controller-т хэрхэн хийхийг харъя.

```
// src/Acme/TaskBundle/Controller/DefaultController.php
namespace Acme\TaskBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Acme\TaskBundle\Entity\Task;
use Symfony\Component\HttpFoundation\Request;

class DefaultController extends Controller
{
    public function newAction(Request $request)
    {
        // create a task and give it some dummy data for this example
        $task = new Task();
        $task->setTask('Write a blog post');
        $task->setDueDate(new \DateTime('tomorrow'));
        $form = $this->createFormBuilder($task)
            ->add('task', 'text')
            ->add('dueDate', 'date')
            ->add('save', 'submit')
            ->getForm();
        return $this->render('AcmeTaskBundle:Default:new.html.twig', array(
            'form' => $form->createView(),
        ));
    }
}
```

Дээрх жишээнд шууд controller дотор форм байгуулхыг үзэж байгаа бөгөөд харин зөвхөн формд зориулан класс үүсгэх талаар дараа үзэх болно. Ингэснээр тэр кодоо дахин дахин ашиглах боломжтой болох юм.

Symfony2-т форм үүсгэхдээ **form builder** ашигладаг.

Дээрх жишээнд, **Task** классын **task** болон **dueDate** гэсэн property-уудтай тохирох task болон dueDate гэсэн форм талбаруудыг нэмсэн байна. Төгсгөлд нь сервер лүү тухайн формын утгыг илгээх **submit** товч нэмж өгсөн байна.

Submit товчыг Symfony 2.3-т анх танилцуулсан. Өмнөх хувилбаруудад form-ийн HTML рүү товч нэмж хийдэг.

### Формыг дэлгэц рүү гаргах

Ингээд формоо үүсгэлээ. Дараагийн алхам бол формыг дэлгэц рүү хэвлэн харуулах. Ингэхдээ, темплэйт рүү "**view**" хэмээх тусгай форм обектийг дамжуулж (дээрх controller –т $form->createView() гэж ашигласан), формын helper функцийг ашиглан харуулна.

```
{# src/Acme/TaskBundle/Resources/views/Default/new.html.twig #}

{{ form(form) }}
```


### Формоос илгээж байгаа мэдээллийг боловсруулах

Дараагийн алхам бол обектийн property-руу хэрэглэгчийн илгээсэн мэдээллийг хөрвүүлэх. Хэрэглэгч нь мэдээллийг илгээхдээ тухайн формыг бөглөсөн байх шаардлагатай. Controller тоо дараах кодыг нэмэе.

```
// ...
use Symfony\Component\HttpFoundation\Request;

public function newAction(Request $request)
{
    // just setup a fresh $task object (remove the dummy data)
    $task = new Task();

    $form = $this->createFormBuilder($task)
        ->add('task', 'text')
        ->add('dueDate', 'date')
        ->add('save', 'submit')
        ->getForm();

    $form->handleRequest($request);

    if ($form->isValid()) {
        // өгөгдлийн сан руу хадгалах гэх мэт үйлдлийг энд хийж болно

        return $this->redirect($this->generateUrl('task_success'));
}
// ...
}
```

Энэ controller нь дараах 3 замаар формыг боловсруулна.

1. Эхлээд вэб хөтчид хуудсыг ачааллах үед формыг үүсгэж, хэрэглэгчид харуулна. **handleRequest()** method нь формоос ямар нэг утга илгээгдээгүй байвал үйлдэл гүйцэтгэхгүй. Хэрэв **isValid()** нь формоос ямар нэг утга ирээгүй бол **false** утга буцаана.
2. Хэрэглэгч формыг бөглөөд мэдээллээ илгээхэд **handleRequest()** үүнийг мэдрээд **$task** обектийн **task** болон **dueDate** property-руу тухайн мэдээллийг илгээнэ. Тэгээд тухайн обектийг **validate**

хийнэ (баталгаажуулалт хэрхэн хийхийг дараагийн сэдэвт үзнэ.). Форм **validate** хийхэд ямар нэг алдаа гарвал **isValid()** нь дахиад л **false** утга буцаана.

3. Хэрэглэгч мэдээлээ алдаагүй зөв илгээсэн бол **isValid()** нь true утга буцааж, энэ үед та **$task** обектийг ашиглан үйлдэл гүйцэтгэх (мэдээллийг өгөгдлийн санд хадгалах гэх мэт) боломжтой болно. Үүний дараа өөр хуудас руу (амжилттай боллоо, баярлаа гэх мэт) хэрэглэгчийн броузерийг шилжүүлж болно.

**handleRequest()** method-ийг Symfony 2.3 –т анх танилцуулсан. Өмнөх хувилбаруудад **$request** –ийг submit method руу дамжуулдаг. Энэ аргыг Symfony 3.0-т хасч ашиглахгүй болгох төлөвлөгөөтэй байгаа.

### Олон товчтой формоос мэдээлэл илгээх

Таны форм нэгээс илүү **submit** товч агуулж байгаа үед аль товч дарагдсан эсэхийг шалгах хэрэг гарна. Үүнийг хэрхэн хийхийг харахын тулд, тухайн формдоо "**Save and add**" гэсэн 2 дахь товчыг нэмээе.

```
$form = $this->createFormBuilder($task)
    ->add('task', 'text')
    ->add('dueDate', 'date')
    ->add('save', 'submit')
    ->add('saveAndAdd', 'submit')
    ->getForm();
```

Controller –т **isClicked()** method ашиглан "**Save and add**" товч дарагдсан үед хийгдэх үйлдлийг заана.

```
if ($form->isValid()) {
    // өгөгдлийн сан руу хадгалах гэх мэт үйлдэл хийж болно
    $nextAction = $form->get('saveAndAdd')->isClicked()
        ? 'task_new'
        : 'task_success';
    return $this->redirect($this->generateUrl($nextAction));
}
```

## Форм Validation

Өмнөх бүлэгт, формоос илгээж байгаа утгыг хэрхэн баталгаажуулахыг үзсэн билээ. Symfony2-н **validation** нь үндсэн обектод (Жнь: Task) ашиглагддаг. Өөрөөр хэлбэл тухайн "form" зөв байгаа эсэхийг асуухгүйгээр формоос мэдээлэл илгээсний дараа $task обектийг зөв байгаа эсэхийг асууна. **$form->isValid()** –ийг дуудсанаар тухайн өгөгдөл зөв байгаа эсэхийг $task обект асууна.

Validation нь тухайн класс руу дүрэм буюу constraint-уудыг нэмсэнээр ажиллана. Ингээд **Task** болон **dueDate** талбар нь хоосон байж болохгүй ба зөв **DateTime** обект байх ёстой гэсэн constraint нэмээе.

```
# Acme/TaskBundle/Resources/config/validation.yml
Acme\TaskBundle\Entity\Task:
    properties:
        task:
            - NotBlank: ~
        dueDate:
            - NotBlank: ~
            - Type: \DateTime
```

Ингээд боллоо. Хэрэв хэрэглэгч буруу өгөгдлөл илгээвэл тохирох алдааны мессеж харуулна.

**HTML5 validation**

Ихэнх броузерууд клейнт талд тодорхой хэмжээний Validation хийх боломжтой болсон. Түгээмэл хийгддэг Validation-ний нэг бол заавал бөглөх шаардлагатай формын талбарт хэрэглэгддэг **required** аттрибут юм.

Энэ клейнт талын Validation-ийг идэвхгүй болгохын тулд form эсвэл submit tag руу **novalidate** аттрибут нэмэх хэрэгтэй. Энэ үйлдэл нь ялангуяа сервер талын Validation-ийг шалгаж байх үед маш их хэрэгцээтэй байдаг.

```
{# src/Acme/DemoBundle/Resources/views/Default/new.html.twig #}

{{ form(form, {'attr': {'novalidate': 'novalidate'}}) }}
```

## Validation бүлгүүд

Мөн та validation group-ийг формдоо ашиглаж болно.

```
$form = $this->createFormBuilder($users, array(
    'validation_groups' => array('registration'),
))->add(...);
```

Хэрэв та тусдаа form класс (хамгийн сайн арга) үүсгэн ашиглаж байгаа бол **setDefaultOptions()** method-ийг нэмэх хэрэгтэй.

```
use Symfony\Component\OptionsResolver\OptionsResolverInterface;

public function setDefaultOptions(OptionsResolverInterface $resolver)
{
    $resolver->setDefaults(array(
        'validation_groups' => array('registration'),
    ));
}
```

Дээрх 2 аргын аль нь ч обектийг баталгаажуулхад зөвхөн **registration validation** group л ашиглагдана.

## Validation-ийг идэвхгүй болгох

**validation_groups**-ийг идэвхгүй болгох тохиргоог Symfony 2.3-т танилцуулсан.

Зарим үед формыг бүхэлд нь validate хийхийг хориглох шаардлага тулгардаг. Ийм тохиолдолд та **validation_groups** сонголтыг **false** болгох хэрэгтэй.

```
use Symfony\Component\OptionsResolver\OptionsResolverInterface;

public function setDefaultOptions(OptionsResolverInterface $resolver)
{
    $resolver->setDefaults(array(
        'validation_groups' => false,
```

```
    ));
}
```

Ийм зүйл хийж байгаа тохиолдолд формыг үндсэн аргаар шалгах хэрэгтэйг анхааараарай. Жишээ нь, сервер лүү хуулж байгаа файлын хэмжээ хэтэрхий их байгаа эсэх, эсвэл огт байхгүй талбараас утга илгээх гэх мэт. Хэрэв баталгаажуулалтыг хаахыг хүсвэл **POST_SUBMIT** –ийг ашиглах боломжтой байдаг.

### Оруулсан мэдээлэл дээр үндэслэсэн бүлгүүд

Хэрэв validation group-ийг тодорхойлохын тулд зарим нэг validation_groups –ийн сонголтуудыг тохируулах хэрэгтэй.

```
use Symfony\Component\OptionsResolver\OptionsResolverInterface;

public function setDefaultOptions(OptionsResolverInterface $resolver)
{
    $resolver->setDefaults(array(
        'validation_groups' => array(
            'Acme\AcmeBundle\Entity\Client',
            'determineValidationGroups',
        ),
    ));
}
```

Форм мэдээлэл илгээсний дараа **Client** класст **determineValidationGroups()** гэсэн статик method дуудагдах боловч validation  өмнө нь ажилласан байна. Форм обект нь тэр method руу аргумент болж дамжигдана. Мөн **Closure** ашиглан шугаман логик тодорхойлж болно.

```
use Symfony\Component\Form\FormInterface;
use Symfony\Component\OptionsResolver\OptionsResolverInterface;

public function setDefaultOptions(OptionsResolverInterface $resolver)
{
    $resolver->setDefaults(array(
        'validation_groups' => function(FormInterface $form) {
            $data = $form->getData();
            if (Entity\Client::TYPE_PERSON == $data->getType()) {
                return array('person');
            } else {
                return array('company');
            }
        },
    ));
}
```

### Дарагдсан товч дээр үндэслэсэн бүлгүүд

Таны форм олон submit товч агуулж байгаа бол формын мэдээллийг илгээхэд аль товч дарагдаж байгаагаас хамаараад validation group –ийг өөрчилж болно. Жишээ нь, өмнөх болон дараагийн алхамтай форм байлаа гэж бодоё. Өмнөх алхам руу буцахад формд бөглөсөн утгыг хадгалсан байх боловч түүнийг баталгаажуулаагүй байна.

Тэгэхээр эхлээд, формдоо 2 товчоо нэмнэ.

```
$form = $this->createFormBuilder($task)
    // ...
    ->add('nextStep', 'submit')
    ->add('previousStep', 'submit')
    ->getForm();
```

Бид өмнөх алхам руу буцахад, заасан validation ажилладаг байхаар товчоо тохируулдаг. Харин энэ жишээнд, validation-ийг хаах бөгөөд ингэхдээ validation_groups сонголтыг false болгоно.

```
$form = $this->createFormBuilder($task)
    // ...
    ->add('previousStep', 'submit', array(
        'validation_groups' => false,
    ))
    ->getForm();
```

Ингэснээр форм нь validation constraint-ийг алгасах болно.

## Built-in Field-ийн төрлүүд

Symfony – д form field буюу форм талбарын томоохон бүлэг байх ба та эдгээр өгөгдлийн төрлүүдтэй олон удаа ажиллана.

### Текст талбарууд

• *text*

• *textarea*

• *email*

• *integer*

• *money*

• *number*

• *password*

• *percent*

• *search*

• *url*

### Сонголтын талбарууд

• *choice*

• *entity*

• *country*

• *language*

• *locale*

• *timezone*

• *currency*

**Он, сарын талбарууд**

• *date*

• *datetime*

• *time*

• *birthday*

**Бусад талбарууд**

• *checkbox*

• *file*

• *radio*

**Талбарын бүлэгүүд**

• *collection*

• *repeated*

**Нуугдмал талбарууд**

• *hidden*

**Товчнууд**

• *button*

• *reset*

• *submit*

**Үндсэн талбар**

• *form*

Эдгээрээс гадна та өөрийн талбарын төрлийг үүсгэх бас боломжтой.

### Талбарын төрөлийн сонголтууд

Талбарын төрөл болгон нь олон тооны сонголтуудтай байдаг. Жишээ нь, **dueDate** талбарыг 3 төрлийн сонголттойгоор харуулдаг байг. Харин, **date** талбарийг энгийн текст талбар байдлаар гаргахаар тохируулая.

```
1  ->add('dueDate', 'date', array('widget' => 'single_text'))
```

Талбарын төрөл бүр нь олон тооны сонголтуудтай байдаг.

**required сонголт**

Хамгийн түгээмэл ашиглагддаг сонголт бол **required** сонголт юм. HTML -5 дэмжидэг хөтчүүд клейнт талын validation-ийг ашиглан хэрэв тухайн талбар хоосон байвал Default-аар required сонголт нь **true** байдаг. Хэрэв та үүнийг ашиглахыг хүсэхгүй бол required сонголтыг false болгох эсвэл HTML5 validation-ийг disable болгох хэрэгтэй.

**label сонголт**

Форм талбарын шошгыг label сонголт ашиглан тохируулна.

```
1 ->add('dueDate', 'date', array(
2     'widget' => 'single_text',
3     'label' => 'Due Date',
4 ))
```

## Темплэйтэд формыг боловсруулж гаргах

Өмнө ганцхан мөр кодоор формыг хэрхэн гаргаж харуулах талаар үзсэн. Одоо харин хэрхэн үян хатан байдлаар харуулахыг үзье.

```
1 {# src/Acme/TaskBundle/Resources/views/Default/new.html.twig #}
2 {{ form_start(form) }}
3     {{ form_errors(form) }}
4     {{ form_row(form.task) }}
5     {{ form_row(form.dueDate) }}
6 {{ form_end(form) }}
```

Дээрх кодыг тайлбарлая.

- **form_start(form)** – Формын эхлэх tag-ийг үүсгэнэ.
- **form_errors(form)** – бүх формын тухай ямар нэг алдааны тухай мэдээлэл харуулна.
- **form_row(form.dueDate)** – формын шошго, алдааны мессеж, ямар нэг талбар зэргийг боловсруулна.
- **form_end()** – Форм болон ямар нэг талбарын хаах tag-тийг үүсгэнэ.

### Гар аргаар талбаруудыг гаргах

**form_row** нь формын талбарыг хурдан үүсгэх боломж олгоно. Гэхдээ, талбар бүрийг өөрөө зааж өгөх хэрэгтэй болдог. **form_row** ашигласан дараах жишээг харая.

```
1 {{ form_start(form) }}
2     {{ form_errors(form) }}
3     <div>
4         {{ form_label(form.task) }}
5         {{ form_errors(form.task) }}
6         {{ form_widget(form.task) }}
7     </div>
8     <div>
9         {{ form_label(form.dueDate) }}
```

```
        {{ form_errors(form.dueDate) }}
        {{ form_widget(form.dueDate) }}
    </div>
    <div>
        {{ form_widget(form.save) }}
    </div>
{{ form_end(form) }}
```

Хэрэв label-ийг автоматаар үүсгэж ашиглах бол дараах аргаар зааж өгч болно.

```
{{ form_label(form.task, 'Task Description') }}
```

Зарим төрлийн талбаруудыг боловсруулах нэмэлт сонголтуудтай байдаг. Нийтлэг хэрэглэгддэг сонголтуудын нэг нь **attr** бөгөөд энэ нь форм элемент дэх атрибутуудыг засах боломжыг олгодог. Дараах кодын жишээнд **input text** талбар луу **task_field** классыг нэмэх болно.

```
{{ form_widget(form.task, {'attr': {'class': 'task_field'}}) }}
```

Хэрэв та форм талбаруудыг өөрөө үүсгэхийг хүсвэл **id, name, label** гэх мэт талбаруудын утга руу тус тусд нь хандах хэрэгтэй болно. Жишээ нь: **id** –г авая.

```
{{ form.task.vars.id }}
```

Формын нэрээр нь авах бол:

```
{{ form.task.vars.full_name }}
```

## Формын Action болон Method-ийг өөрчилөх

Өмнө бид **form_start()**-ийг формын эхлэх tag-ийг үүсгэхэд ашигласан ба **POST** –оор тухайн **URL** руу формын мэдээллийг илгээж үзсэн. Заримдаа, эдгээр параметерүүдийг өөрчлөх шаардлага гардаг. Хэрэв та controller дотор форм байгуулж байгаа бол **setAction()** болон **setMethod()** –ийг ашиглах хэрэгтэй.

```
$form = $this->createFormBuilder($task)
    ->setAction($this->generateUrl('target_route'))
    ->setMethod('GET')
    ->add('task', 'text')
    ->add('dueDate', 'date')
    ->add('save', 'submit')
    ->getForm();
```

Дараагийн сэдвээр тусдаа класст формыг хэрхэн үүсгэхийг үзэх болно. Controller-т гаднаас класс оруулж ашиглаж байгаа бол **action** болон **method**-ийг дамжуулна.

```
$form = $this->createForm(new TaskType(), $task, array(
    'action' => $this->generateUrl('target_route'),
    'method' => 'GET',
));
```

Эцэст нь **form()** эсвэл **form_start()** руу эдгээрийг дамжуулсанаар темплэйтэд **action**, **method** 2-ийг оруулж ирнэ.

```
{# src/Acme/TaskBundle/Resources/views/Default/new.html.twig #}
{{ form(form, {'action': path('target_route'), 'method': 'GET'}) }}
{{ form_start(form, {'action': path('target_route'), 'method': 'GET'}) }}
```

## Форм класс үүсгэх

Өмнөх жишээнүүдэд бид controller дотор формыг шууд үүсгэн, ашиглаж үзсэн. Гэхдээ, тусдаа PHP класс файлд формыг үүсгэх нь хамгийн сайн арга юм. Энэ нь програмын хаана ч түүнийг дахин дахин ашиглах боломжыг бүрдүүлж өгнө.

```
// src/Acme/TaskBundle/Form/Type/TaskType.php
namespace Acme\TaskBundle\Form\Type;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;

class TaskType extends AbstractType
{
   public function buildForm(FormBuilderInterface $builder, array $options)
   {
        $builder
            ->add('task')
            ->add('dueDate', null, array('widget' => 'single_text'))
            ->add('save', 'submit');
        }

    public function getName()
    {
        return 'task';
    }
}
```

Энэ класс форм үүсгэхэд шаардлагатай бүх зааварыг агуулна (**getName()** method нь энэ формын дахин давхардахгүй нэрийг буцаана гэдэгийг анхааараарай). Үүнийг controller дотор ашигласнаар формыг хялбархан байгуулах болно.

```
// src/Acme/TaskBundle/Controller/DefaultController.php

// add this new use statement at the top of the class
use Acme\TaskBundle\Form\Type\TaskType;

public function newAction()
{
    $task = ...;
    $form = $this->createForm(new TaskType(), $task);
    // ...
}
```

Ингэж формыг өөр тусдаа класст үүсгэснээр програмын хаана ч дахин дахин ашиглах боломж бий болно. Энэ бол форм үүсгэх хамгийн сайн арга юм.

### Сервис маягаар формыг тодорхойлох

Форм төрлөө сервис маягаар тодорхойлох нь маш сайн арга бөгөөд програмын ашиглалтыг маш хялбар болгоно.

```
# src/Acme/TaskBundle/Resources/config/services.yml
services:
    acme_demo.form.type.task:
```

```
        class: Acme\TaskBundle\Form\Type\TaskType
        tags:
            - { name: form.type, alias: task }
```

Ингээд controller-тоо форм төрлийг шууд ашиглах боломжтой болно.

```php
// src/Acme/TaskBundle/Controller/DefaultController.php
// ...
public function newAction()
{
    $task = ...;
    $form = $this->createForm('task', $task);
    // ...
}
```

Эсвэл өөр формын форм төрлөөс ч ашиглаж болно.

```php
// src/Acme/TaskBundle/Form/Type/ListType.php
// ...
class ListType extends AbstractType
{
   public function buildForm(FormBuilderInterface $builder, array $options)
   {
       // ...
       $builder->add('someTask', 'task');
   }
}
```

## Форм ба Doctrine

Формын үндсэн зарчим бол обект (жишээ нь Task) –оос HTML формыг дамжуулах ба буцаагаад хэрэглэгчээс формоор илгээгдэж байгаа мэдээллийг обект руу дамжуулах юм. Та **Doctrine** ашиглаад тухайн форм зөв байгаа тохиолдолд формоос ирж байгаа мэдээллийг аваад өгөгдлийн санд хадгалж болно.

```php
if ($form->isValid()) {
    $em = $this->getDoctrine()->getManager();
    $em->persist($task);
    $em->flush();

    return $this->redirect($this->generateUrl('task_success'));
}
```

Заримд тохиолдолд **$task** обектруу хандахгүйгээр формоос тухайн мэдээллийг нь шууд авч болно.

```php
 $task = $form->getData();
```

## Хавсаргасан форм

Заримдаа, өөр өөр обектуудаас бүтсэн форм үүсгэх шаардлага гардаг. Жишээ нь, хэрэглэгч бүртгэх форм нь нэг **User** гэсэн обект болон олон тооны **Address** обектоос бүтдэг байг. Үүнийг **Form** компонент ашиглан хялбархан хийж гүйцэтгэнэ.

### Ганц дан обект хавсаргах

**Category** обекттой холбоотой **Task** обект хэрэгтэй байна гэж төсөөлөе. Тэгэхээр эхлээд, **Category** обектийг үүсгэнэ.

```
// src/Acme/TaskBundle/Entity/Category.php
namespace Acme\TaskBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Category
{
    /**
    * @Assert\NotBlank()
    */
    public $name;
}
```

Дараа нь, **Task** класс руу шинээр **category** property-г нэмнэ.

```
// ...

class Task
{
    // ...
    /**
    * @Assert\Type(type="Acme\TaskBundle\Entity\Category")
    */
    protected $category;

    // ...

    public function getCategory()
    {
        return $this->category;
    }

    public function setCategory(Category $category = null)
    {
        $this->category = $category;
    }
}
```

Одоо хэрэглэгч өөрчлөх боломжтой **Category** обектийн форм классыг үүсгэнэ.

```
// src/Acme/TaskBundle/Form/Type/CategoryType.php
namespace Acme\TaskBundle\Form\Type;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolverInterface;

class CategoryType extends AbstractType
{
   public function buildForm(FormBuilderInterface $builder, array $options)
    {
```

```
        $builder->add('name');
    }
    public function setDefaultOptions(OptionsResolverInterface $resolver)
    {
        $resolver->setDefaults(array(
            'data_class' => 'Acme\TaskBundle\Entity\Category',
        ));
    }
    public function getName()
    {
        return 'category';
    }
}
```

Үүний эцсийн зорилго нь **task** форм дотор **Task**-ийн **Category** обектод өөрчлөлт хийх боломж олгох явдал юм. Үүний тулд **CategoryType** классын төрөл обект болох **TaskType** обект руу **category** талбарыг нэмнэ.

```
use Symfony\Component\Form\FormBuilderInterface;
public function buildForm(FormBuilderInterface $builder, array $options)
{
    // ...
    $builder->add('category', new CategoryType());
}
```

**CategoryType** –т validation-ийг идэвхжүүлэх бол TaskType –руу **cascade_validation**-ийг нэмнэ.

```
public function setDefaultOptions(OptionsResolverInterface $resolver)
{
    $resolver->setDefaults(array(
        'data_class' => 'Acme\TaskBundle\Entity\Task',
        'cascade_validation' => true,
    ));
}
```

Category талбарыг хэрэглэгч рүү гаргахдаа:

```
{# ... #}
<h3>Category</h3>
<div class="category">
    {{ form_row(form.category.name) }}
</div>
{# ... #}
```

Хэрэглэгч форм бөглөөд илгээхэд, **Category** талбарт бөглөгдсөн мэдээлэл нь **Category** –ийн обектийг үүсгэхэд ашиглагдана.

Category обект руу **$task->getCategory()** гэж хандан өгөгдлийн сан руу хадгалах гэх мэт дуртай үйлдлээ хийж болно.

### Формын цуглуулгыг хавсаргах

Та мөн нэг формд формуудын цуглуулгыг оруулж болно (жишээ нь: олон тооны Product гэсэн дэд формуудтай Category форм байж болно гэх мэт). Үүнийг collection талбарын төрлийг ашиглан хийнэ.

## Формд темплэйт ашиглах

Symfony нь label, input, алдааны мессеж гэх мэт формын хэсэг бүрийг харуулахдаа темплэйт ашигладаг.

Twig темплэйтэд формын хэсэг бүр нь Twig-ийн блок дотор байрлана. Тухайн хэсгүүдийг зохион байгуулахдаа тохирох блокыг оруулж ирэн ашигладаг.

Харин PHP темплэйтэд формын хэсэг бүрийг тус тусдаа темплэйт файлд хадгалдаг. Энэ нь мөн тухайн хэсгүүдийг зохион байгуулахдаа үүссэн байгаа темплэйт рүү шинээр темплэйт үүсгэн оруулж ирэх хэрэгтэй байдаг.

Тэгвэл одоо **form_row** гэсэн хэсэг үүсгээд, тухайн мөр бүрийг агуулсан **div** элемент рүү класс нэмэе.

Үүнийг хийхдээ, шинэ темплэйт файл үүсгэнэ.

```
{# src/Acme/TaskBundle/Resources/views/Form/fields.html.twig #}
{% block form_row %}
{% spaceless %}
    <div class="form_row">
        {{ form_label(form) }}
        {{ form_errors(form) }}
        {{ form_widget(form) }}
    </div>
{% endspaceless %}
{% endblock form_row %}
```

**form_row** хэсэг нь form_row функц талбаруудыг харуулах үед хэрэглэгдэнэ. Дээр тодорхойлсон form_row хэсэгийг ашиглахын тулд формыг харуулах гэж байгаа темплэйт файлынхаа дээд талд дараах кодыг нэмэе.

```
{# src/Acme/TaskBundle/Resources/views/Default/new.html.twig #}
{% form_theme form 'AcmeTaskBundle:Form:fields.html.twig' %}
{% form_theme form 'AcmeTaskBundle:Form:fields.html.twig'
'AcmeTaskBundle:Form:fields2.html.twig' %}
<!-- ... render the form -->
```

**form_theme** tag нь заасан темплэйтэд тодорхойлсон хэсгүүдийг энд оруулж ирнэ. Өөрөөр хэлбэл, **form_row** функцийг энэ темплэйтэд дуудах үед энэ нь хэрэглэгчийн үүсгэсэн темплэйтээс form_row блокыг дуудан ашиглана.

Үүсгэсэн темплэйтдээ бүх блокыг заавал оруулж ирэх шаардлагагүй. Хэрэглэгчийн үүсгэсэн темплэйтэд оруулж ирээгүй блокыг харуулах үед темплэйт engine нь тэдгээрийг глобал темплэйт рүү буцаана.

### Форм хэсгүүдийг нэрлэх

Symfony-д формын хэсэг бүр нь HTML элемент, алдааны мессеж, labels гэх мэт үндсэн темплэйтэд тодорхойлсон элементүүдийг агуулдаг.

Twig темплэйтэд шаардлагатай блок бүрийг нэг темплэйт файлд тодорхойлж өгдөг.

PHP –д хэсэгүүд нь тусдаа темплэйт файлд байрлана. Энэ нь Form хавтасны **Resources/views/** хавтсанд байрлаж байдаг.

Формын хэсгийн нэр нь доогуур зураасаар ( _ ) тусгаарласан 2 хэсгээс бүрдэнэ. Жишээ нь:

- **form_row** – нь талбаруудыг хэвлэхэд ашиглагдана.
- **textarea_widget** – текст талбарыг харуулна.
- **form_errors** – тухайн талбартай холбоотой алдааны мессежийг харуулна.

Эндээс хархад хэсэг бүр нь **type_part** гэсэн үндсэн загвартай байна. **Type** нь тохирох талбарын төрөл (жишээ нь **textarea, checkbox, date,** гэх мэт) бол **part** гэдэг нь юу (**label, widget, errors**, гэх мэт) харуулах гэж байгааг заана. Default-аар 4 төрлийн формын хэсэг байна.

| | |
|---|---|
| Label | Формын талбарын шошго буюу label-ийг хэвлэнэ |
| Widget | Талбарын HTML кодыг хэвлэнэ |
| Error | Талбарын алдааны мессежийг хэвлэнэ |
| row | Талбарын мөрийг бүхэлд нь хэвлэнэ |

Талбарын төрөл (textarea гэх мэт)-ийг мэдэж авсанаар шаардлагатай хэсгийн нэрийг ашиглах боломжтой болох юм.

## CSRF хамгаалалт

CSRF - Cross-site request forgery[34] нь формд мэдээ оруулахдаа санамсаргүй болон санаатай байдлаар хортой код илгээснээр үүсдэг халдлага юм. Энэ халдлагаас сэргийлэхийн тулд формдоо CSRF token ашиглах хэрэгтэй. Гэхдээ Symfony нь CSRF token –ийг автоматаар оруулж ирдэг.

CSRF хамгаалалт нь **_tolen** –ийг дуудан формдоо hidden field нэмсэнээр ажиллах ба энэ field нь зөвхөн таны мэдэх утгыг агуулна. Энэ нь тухайн өгөгдлөөс өөр ямар нэг хэсгийг илгээгээгүй хэрэглэгч гэдэгийг баталгаажуулна.

_token бол hidden field ба темплэйтдээ form_end() функц оруулж ирсэн үед энэ field нь автоматаар боловсруулагдана.

```php
use Symfony\Component\OptionsResolver\OptionsResolverInterface;

class TaskType extends AbstractType
{
    // ...
    public function setDefaultOptions(OptionsResolverInterface $resolver)
    {
        $resolver->setDefaults(array(
            'data_class' => 'Acme\TaskBundle\Entity\Task',
            'csrf_protection' => true,
            'csrf_field_name' => '_token',
            // a unique key to help generate the secret token
            'intention' => 'task_item',
        ));
    }
    // ...
}
```

CSRF хамгаалалтыг идэвхгүй болгохын тулд **csrf_protection** –ийг false болгох хэрэгтэй.

[34] http://en.wikipedia.org/wiki/Cross-site_request_forgery

## Класс үүсгэхгүйгээр форм ашиглах

Зарим үед тусдаа класс үүсгэлгүйгээр форм ашиглах, ирж байгаа өгөгдлийн массивийг буцааж авах хэрэг гардаг.

```
// make sure you've imported the Request namespace above the class
use Symfony\Component\HttpFoundation\Request;
// ...
public function contactAction(Request $request)
{
    $defaultData = array('message' => 'Type your message here');
    $form = $this->createFormBuilder($defaultData)
        ->add('name', 'text')
        ->add('email', 'email')
        ->add('message', 'textarea')
        ->add('send', 'submit')
        ->getForm();
    $form->handleRequest($request);
    if ($form->isValid()) {
        // data is an array with "name", "email", and "message" keys
        $data = $form->getData();
    }
    // ... render the form
}
```

Default –аар форм нь обектийн оронд массивтай ажилладаг. Үүнийг өөрчилж, обекттой ажиладаг болгох дараах 2 төрлийн арга байдаг.

1 Форм үүсгэхдээ обектийг дамжуулах (createFormBuilder –ийн эхний аргумент эсвэл createForm-руу 2 дахь аргумент болгон дамжуулна.)
2 Формдоо **data_class** –ийг зарлах

Хэрэв энэ 2 аргын алийг ч ашиглаагүй бол форм нь өгөгдлийн массиваар буцаадаг. Дээрх жишээнд, **$defaultData** нь обект биш бөгөөд **$form->getData()** нь массив буцааж байна.

### Validation нэмэх

Ихэнхдээ **$form->isValid()** –ийг дуудан тухайн класст ашиглагдаж буй constraint-уудыг уншиж validate хийнэ. Хэрэв таны форм өгөгдөл обект байдлаар ашиглагдаж байгаа бол энэ нь сайн арга юм.

Харин обектийн оронд массив байдлаар өгөгдлийг авч байгаа бол constraint-аа та өөрөө үүсгэж, форм талбаруудаа тэдгээрийг хавсаргаж өгөх хэрэгтэй.

```
use Symfony\Component\Validator\Constraints\Length;
use Symfony\Component\Validator\Constraints\NotBlank;
$builder
    ->add('firstName', 'text', array(
        'constraints' => new Length(array('min' => 3)),
    ))
    ->add('lastName', 'text', array(
        'constraints' => array(
            new NotBlank(),
            new Length(array('min' => 3)),
        ),
    ))
;
```

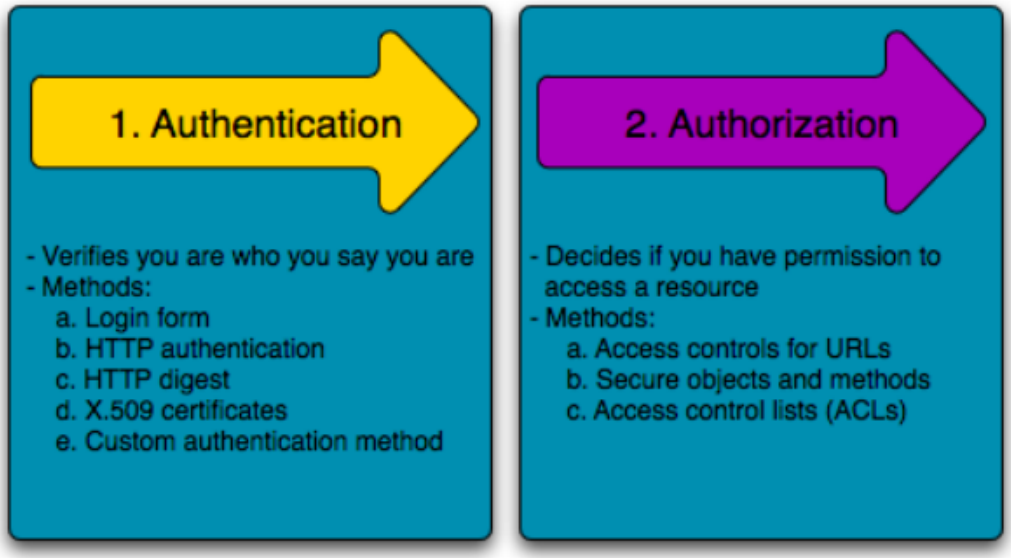
13-р бүлэг

# Хамгаалалт

Хамгаалалт нь програмын хандах ёсгүй хэсэг рүү хэрэглэгчийг хандахаас сэргийлэх зорилгоор 2 алхамыг хэрэгжүүлдэг.

Эхний алхамд, тухайн хамгаалалтын систем нь хэрэглэгчийг таних бөгөөд үүнийг **authentication** гэж нэрлэнэ.

Систем нь хэрэглэгчийг таньж аваад, дараагийн алхамд тухайн хэрэглэгч програмын аль хэсэгт хандах ёстойг тогтооно. Энэ 2 дахь алхамыг **authorization** гэж нэрлэнэ.




Үүнийг судлаж үзэхийн тулд энгийн HTTP authentication ашиглан програмыг хэрхэн хамгаалах талаар нэг жишээ үзье.

## Энгийн жишээ: HTTP Authentication

Security компонентийг програмын тохиргооны файлд тохируулж өгнө. Дараах жишээнд, **/admin/*** гэсэн үгийг агуулж буй ямар ч URL-ийг хамгаална гэдэгийг Symfony-д зааж өгөх ба HTTP Authentication ашиглан тохирох хэрэглэгчийг шалгана.

```
# app/config/security.yml
security:
    firewalls:
        secured_area:
            pattern: ^/
            anonymous: ~
            http_basic:
                realm: "Secured Demo Area"
    access_control:
        - { path: ^/admin/, roles: ROLE_ADMIN }
        # Include the following line to also secure the /admin path itself
        # - { path: ^/admin$, roles: ROLE_ADMIN }
    providers:
        in_memory:
            memory:
                users:
                    ryan: { password: ryanpass, roles: 'ROLE_USER' }
                    admin: { password: kitten, roles: 'ROLE_ADMIN' }
    encoders:
        Symfony\Component\Security\Core\User\User: plaintext
```

Үүний үр дүнд хамгаалалтын систем дараах үйлдлийг хэрэгжүүлнэ.

- Системд 2 төрлийн хэрэглэгч байна. (**ryan** ба **admin**)
- Хэрэглэгчийг HTTP authentication –аар танина.
- **/admin/*** гэсэн хэсгийг агуулж буй URL хамгаалагдсан байх бөгөөд зөвхөн админ хэрэглэгч л хандах боломжтой.
- /admin/* хэсгийг агуулаагүй бусад бүх URL –руу бүх хэрэглэгч хандах боломжтой.

## Хамгаалалтын хэрхэн хэрэгжүүлдэг вэ?: Authentication болон Authorization

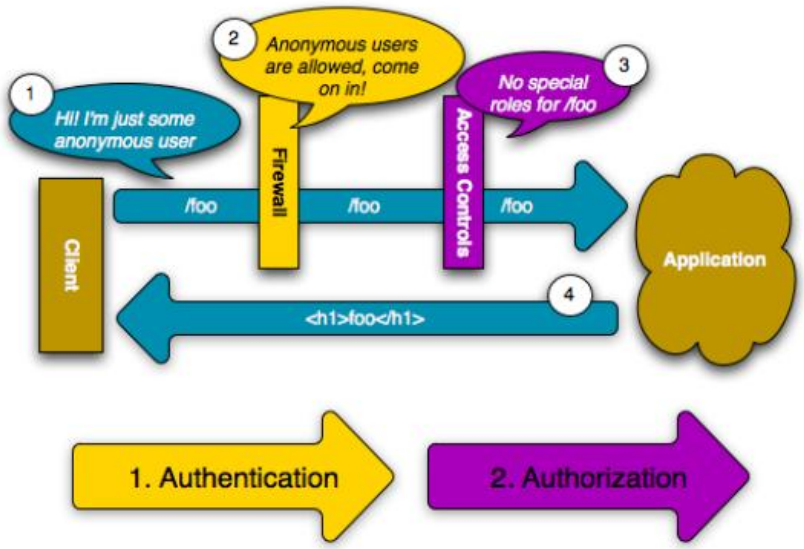
Symfony-ийн хамгаалатын систем нь хэрэглэгчийг хэн болохыг тодорхойлоод (authentication), тухайн хэрэглэгч програмын аль хэсэг буюу аль URL –руу хандах ёстойг шалгаж үзнэ.

### Firewall (Authentication)

Хэрэглэгч firewall-оор хамгаалагдсан URL руу request илгээх үед хамгаалалтын систем нь идэвхсэн байна. Firewall –ийн ажил бол тухайн хэрэглэгч өөрийгөө таниулан нэвтрэх хэрэгтэй эсэхийг шийдэх ба хэрэв таниулах шаардлагатай бол authentication хийх response-ийг хэрэглэгч рүү буцаана.

Энэ жишээнд **pattern** (^/) орж ирж буй бүх request-ийг тохируулж үзэх болно. Энэ нь firewall идэвхтэй байгаа гэсэн үг биш ч гэлээ HTTP authentication нь URL рүү хандах бүрт хэрэглэгчийн нэр, нууц үгийг

оруулах цонхыг харуулах болно. Жишээ нь, ямар ч хэрэглэгч **/foo** гэсэн URL-рүү authenticate хийхгүйгээр хандах боломжтой.

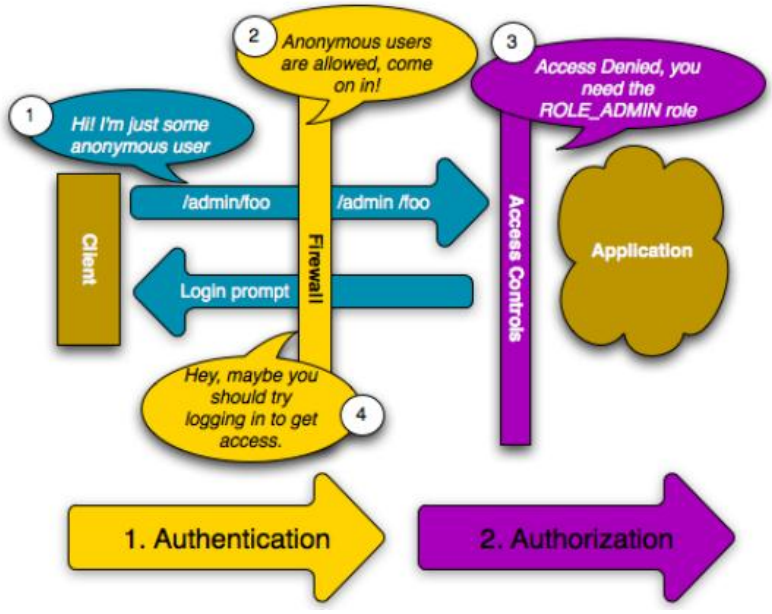



Энэ нь хамгийн түрүүнд ажиллах учир firewall нь **anonymous** параметерээр anonymous хэрэглэгч хандахыг зөшөөрнө. Өөрөөр хэлбэл, тухайн хэрэглэгчийг бүрэн таних шаардлагагүй гэсэн үг. Мөн **foo/** рүү хандахад **role** заах хэрэггүй ба request нь тухайн хэрэглэгчийг authenticate хийгүйгээр биелэгдэх болно.

Хэрэв **anonymous** түлүүр үгийг хасвал firewall нь хэрэглэгчийг үргэлж бүрэн authenticate хийж байхыг шаардах болно.

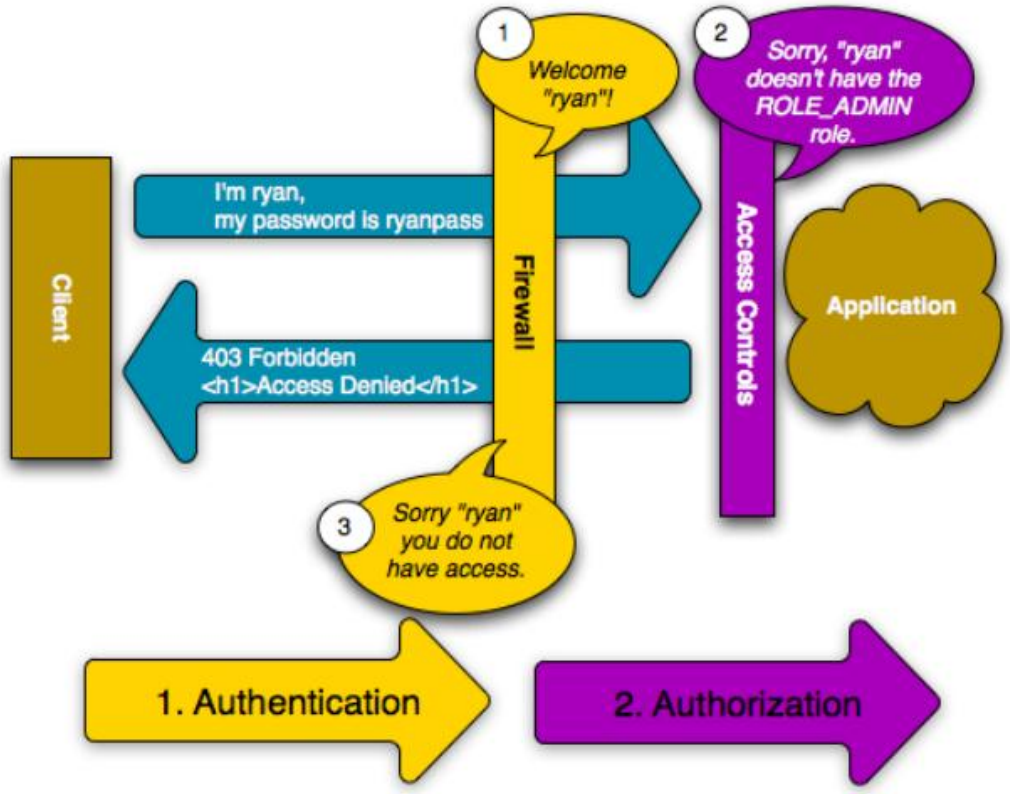
### Хандалтыг хянах (Authorization)

Хэрэв хэрэглэгч **/admin/foo** рүү хандвал энэ нь арай өөрөөр ажиллана. Учир нь **access_control** нь **^/admin** regular expression-тэй тохирч байгаа тухайн нэг URL-ийг **ROLE_ADMIN** эрхтэй байхыг шаардана. Role нь authorization –ийн гол хэсэг байдаг. Хэрэглэгч нь **ROLE_ADMIN** эрхтэй байгаа үед л **/admin/foo** рүү хандах боломжтой байна.



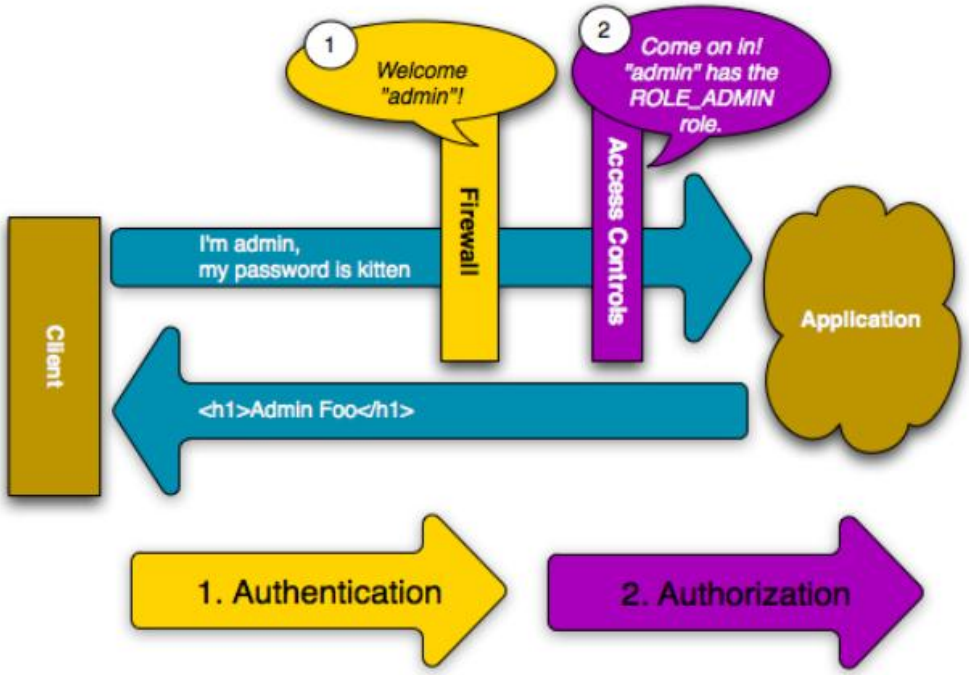
Өмнөхтэй ижилээр бүх хэрэглэгч request илгээхэд firewall нь ямар нэг таних үйл ажиллагаа хийхгүй. Гэхдээ **access control** хэсэг рүү бүх хэрэглэгч орох боломжгүй (учир нь энийн хэрэглэгчидэд ROLE_ADMIN эрх байхгүй) учир хэрэглэгч энэ хэсэг рүү хандсан тохиолдолд firewall нь хэрэглэгчийг таних үйл ажиллагааг эхлүүлнэ. Харин Хэрэглэгчийг таних үйл ажиллагаа нь таны ашиглаж буй хэрэглэгчийг таних механизмаас хамаарна. Жишээ нь: Хэрэв та хэрэглэгч нэвтрэх форм ашиглаж байгаа бол тухайн хэрэглэгчийг нэвтрэх хуудас руу шилжүүлэх болно. Харин HTTP authentication ашиглаж байгаа бол HTTP 401 response илгээж тухайн хэрэглэгчид нэр болон нууц үг оруулах цонхыг харуулах болно.

Ингээд хэрэглэгч нэвтрэх мэдээллээ програм руу илгээх боломжтой болно. Хэрэв нэвтрэх мэдээлэл зөв байвал хэрэглэгчийн анхны хүсэлтийг дахин илгээнэ.




Энэ жишээнд, ryan нэртэй хэрэглэгчийг firewall ашиглан амжилттай authenticate хийсэн ч энэ хэрэглэгч ROLE_ADMIN эрхгүй байгаа учир **/admin/foo** –рүү хандах боломжгүй. Тиймээс тухайн хэрэглэгчид хандах эрх хүрэхгүй байгаа тухай мессежийг харуулна.

Эцэст нь, хэрэв админ хэрэглэгч **/admin/foo** рүү request илгээвэл дээрхтэй ижил процесс явагдаж authenticate хийсний дараа **access control** давхрага нь хүсэлтийг дамжуулан өнгөрүүлэх болно.




## Нэвтрэх форм ашиглах

Энэ бүлгээр HTML форм ашиглан хэрэглэгчийг хэрхэн програмд нэвтрүүлэхийг үзэх болно.

Эхлээд, firewall-ийн дагуу нэвтрэх формоо идэвхжүүлэх хэрэгтэй.

```
# app/config/security.yml
security:
    firewalls:
        secured_area:
            pattern: ^/
            anonymous: ~
            form_login:
                login_path: login
                check_path: login_check
```

Ингэснээр хамгаалатын систем хэрэглэгчийг нэвтрүүлэх үйл ажиллагааг эхлүүлэхдээ хэрэглэгчийг нэвтрүүлэх форм (default-аар /login) руу хэрэглэгчийн шилжүүлнэ. Эхлээд, хамгаалалтын тохиргооны файлд ашиглагдах 2 route-ийг үүсгэнэ. **login** route (жнь: /login) нь нэвтрэх формыг харуулах бол **login_check** route (жнь: /login_check) нь нэврэх формоос илгээгдэж байгаа утгыг боловсруулна.

```
# app/config/routing.yml
login:
    path: /login
    defaults: { _controller: AcmeSecurityBundle:Security:login }
login_check:
    path: /login_check
```

**login** route-ийн нэр нь хамгаалалтын тохиргооны файлд бичсэн **login_path** –ийн утгатай ижил байх ёстой гэдэгийг анхааараарай. Ингэснээр хамгаалалтын систем нь нэвтрэх форм руу хэрэглэгчийг шилжүүлнэ.

Дараа нь, нэвтрэх формыг харуулах controller үүсгэнэ.

```
1  // src/Acme/SecurityBundle/Controller/SecurityController.php;
2  namespace Acme\SecurityBundle\Controller;
3
4  use Symfony\Bundle\FrameworkBundle\Controller\Controller;
5  use Symfony\Component\HttpFoundation\Request;
6  use Symfony\Component\Security\Core\SecurityContextInterface;
7  class SecurityController extends Controller
8  {
9      public function loginAction(Request $request)
10     {
11         $session = $request->getSession();
12         // get the login error if there is one
13         if ($request-> attributes-
   >has(SecurityContextInterface::AUTHENTICATION_ERROR)) {
14             $error = $request->attributes->get(
15                 SecurityContextInterface::AUTHENTICATION_ERROR
16             );
17         } else {
18             $error = $session-
   >get(SecurityContextInterface::AUTHENTICATION_ERROR);
19             $session-
   >remove(SecurityContextInterface::AUTHENTICATION_ERROR);
20         }
21         return $this->render(
22             'AcmeSecurityBundle:Security:login.html.twig',
23             array(
24                 // last username entered by the user
25                 'last_username' => $session-
   >get(SecurityContextInterface::LAST_USERNAME),
26                 'error' => $error,
27             )
28         );
29     }
30 }
```

Хэрэглэгч фомд бөглөсөн мэдээллээ илгээхэд хамгаалалтын систем нь тухайн мэдээллийг автоматаар боловсруулна. Хэрэв хэрэглэгчийн илгээж байгаа нэр, нууц үг буруу байвал дээрх controller нь хамгаалатын системээс алдааны мессежийг уншиж, хэрэглэгч рүү буцаана.

Өөрөөр хэлбэл, таны хийх ажил бол нэвтрэх формыг болон гарч болох ямар нэг алдааг хэрэглэгчид харуулах явдал юм. Харин хамгаалатын систем нь хэрэглэгчийн нэр, нууц үгийг шалгана.

```
1  {# src/Acme/SecurityBundle/Resources/views/Security/login.html.twig #}
2  {% if error %}
3      <div>{{ error.message }}</div>
4  {% endif %}
5      <form action="{{ path('login_check') }}" method="post">
6      <label for="username">Username:</label>
7      <input    type="text"    id="username"    name="_username"    value="{{
   last_username }}" />
```

```
8       <label for="password">Password:</label>
9       <input type="password" id="password" name="_password" />
10      {#
11          If you want to control the URL the user
12          is redirected to on success (more details below)
13          <input type="hidden" name="_target_path" value="/account" />
14      #}
15      <button type="submit">login</button>
16  </form>
```

Формоос утга илгээхэд хамгаалатын систем хэрэглэгчийн илгээж байгаа мэдээллийг автоматаар шалгаад, хэрэглэгчийг нэвтрүүлэх, эсвэл буцааж нэвтрэх форм руу шилжүүллж алдаан харуулах гэсэн үйлдлийг биелүүлнэ.

Энэ үйл ажиллагааг дараах алхамуудаар хийгдэнэ.

1. Хэрэглэгч програмын хамгаалагдсан хэсэг рүү хандах оролдлого хийнэ.
2. Firewall нь нэвтрэх форм (**/login**) руу хэрэглэгчийг шилжүүлж, хэрэглэгчийг танин нэвтрүүлэх ажлыг эхлүүлнэ.
3. **/login** хуудас нь нэвтрэх формыг харуулна.
4. Хэрэглэгч нэвтрэх формоос **/login_check** рүү мэдээллээ илгээнэ.
5. Хамгаалалтын систем нь хэрэглэгчийн илгээж байгаа мэдээллийг шалгаад, хэрэв хэрэглэгчийн нэвтрэх мэдээлэл зөв байвал хамгаалагдсан хэсэг рүү нэвтрүүлж, харин буруу байвал хэрэглэгчийг буцаагаад нэвтрэх форм руу шилжүүлнэ.

Хэрэглэгчийн нэвтрэх мэдээлэл зөв байвал хэрэглэгчийг нэвтрэхийг хүссэн хуудас (жнь: **/admin/foo**) - руу нь шилжүүлнэ.

## Authorization (эрх олгох)

Хэрэглэгчийг танин нэвтрүүлэх үйл ажиллагаа хийгдсэний дараа хэрэглэгчид эрх олгох үйл ажиллагаа эхлэнэ. Эрх олгох үйл ажиллагаа нь хэрэглэгчийг програмын ямар хэсэгт (URL, model object, method call, ...) хандах боломжтой эсэхийг тогтооно. Энэ нь хэрэглэгч бүрт тусгай эрх олгоно.

Эрх олгох үйл ажиллагаа хийгдэхийн тулд дараах 2 хэсэг байна.

1. Хэрэглэгч нь тодорхой нэг эрх үүрэгтэй байна.
2. Програмын тусгай хэсэгт хандахын тулд тодорхой эрх шаардлагатай байна.

### Тодорхой URL-ийг хамгаалах

Програмын тодорхой хэсгийг хамгаалах хамгийн нийтлэг арга бол тухайн URL-ийг бүхэлд нь хамгаалах явдал юм.

Regular expression ашиглан хэрэгтэй URL pattern тодорхойлно.

```
1   # app/config/security.yml
2   security:
3       # ...
4       access_control:
```

```
5           - { path: ^/admin/users, roles: ROLE_SUPER_ADMIN }
6           - { path: ^/admin, roles: ROLE_ADMIN }
```

^ тэмдэг нь тухайн URL яг ингэж эхэлнэ гэдэгийг заана. Жишээ нь: **/admin** (^ тэмдэггүйгээр) гэсэн URL нь **/admin/foo** гэсэнтэй тохирох боловч мөн **/foo/admin** гэсэнтэй ч бас тохирно.

### IP-аар хамгаалах

Заримдаа IP хаяг дээр үндэслээд өгсөн URL рүү хандахыг хязгаарлах хэрэгтэй байдаг. Энэ нь ялангуяа, Edge Side Includes (ESI) гэх мэт тохиолдолд их хэрэгтэй. ESI нь програмд нэвтэрсэн байгаа хэрэглэгчийн тухай мэдээлэл зэрэг хувийн агуулгыг агуулна. Эдгээр хувийн мэдээлэл рүү вэб браузераас шууд хандахаас сэргийлэх зорилгоор зөвхөн итгэлтэй proxy-оос л хандаж болохоор ESI route нь хамгаалагдсан байх ёстой.

Доорх жишээнд **/esi** –ээр эхэлсэн ESI route-үүдийг хэрхэн хамгаалах тухай үзүүллээ.

```
1  # app/config/security.yml
2  security:
3      # ...
4      access_control:
5          - { path: ^/esi, roles: IS_AUTHENTICATED_ANONYMOUSLY, ips:
   [127.0.0.1, ::1] }
6          - { path: ^/esi, roles: ROLE_NO_ACCESS }
```

- Эхний access control нь **path** тохирсон хэдий ч **ip** нь тохироогүй бол request-ийг үл биелүүлнэ.
- Хоёр дахь access control нь ROLE_NO_ACCESS эрхтэй хэрэглэгч хандах эрхгүй гэдэгийг тус тус заана.

Хэрэв 127.0.0.1 хаягнаас request ирвэл:

- Эхний access control нь идэвхжиж, **path** болон **ip** тохирж байгаа **IS_AUTHENTICATED_ANONYMOUSLY** эрхтэй хэрэглэгчийг хандахыг зөвшөөрнө.
- Эхний route тохирсон учир хоёр дахь access control шалгахгүй.

### Expression-ээр хамгаалах

**Roles** түлхүүр үгийг ашиглахаас гадна **allow_if** түлхүүр үг ашиглан regular expression зааж өгч болно.

```
1  # app/config/security.yml
2  security:
3      # ...
4      access_control:
5          -
6              path: ^/_internal/secure
7              allow_if: "'127.0.0.1' == request.getClientIp() or
   has_role('ROLE_ADMIN')"
```

Энэ тохиолдолд хэрэглэгч **/_internal/secure** гэж эхэлсэн URL-аас хандахад тухайн хандалт нь зөвхөн **127.0.0.1** IP-гаас эсвэл **ROLE_ADMIN** эрхтэй байгаа үед л хандахыг зөвшөөрнө.

Regural expression-д олон тооны хувьсагч, функцүүд ашиглаж болно.

### Forcing a Channel (http, https)

Мөн та тухайн хэрэглэгчийг SSL-р URL руу хандахыг шаардаж болно. Инэгхдээ ямар нэг **access_control**-т **requires_channel** аргументийг ашиглана. Хэрэв access_control тохирч байвал request нь http channel-ийг ашиглаад хэрэглэгчийг  https рүү шилжүүлнэ.

```
# app/config/security.yml
security:
    # ...
    access_control:
        - { path: ^/cart/checkout, roles: IS_AUTHENTICATED_ANONYMOUSLY, requires_channel:https }
```

### Controller дотороос хамгаалах

URL pattern ашиглаад програмыг хамгаалах нь хэдийгээр хялбар байдаг ч зарим тохиолдолд энэ нь хангалтгүй байх үе гардаг. Тэгэхээр шаардлагатай үед controller дотроо хэрэглэгчийн эрхийг оноож өгч болно.

```
// ...

public function helloAction($name)
{
    if (false === $this->get('security.context')->isGranted('ROLE_ADMIN')) {
    throw $this->createAccessDeniedException('Энэ хуудас руу хандах эрхгүй байна!');
    }
    // ...
}
```

**createAccessDeniedException** method-ийг Symfony 2.5 –д таницуулсан.

Мөн **annotation** ашиглаж болно.

```
// ...
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Security;

/**
* @Security("has_role('ROLE_ADMIN')")
*/
public function helloAction($name)
{
    // ...
}
```

### Access Control Lists (ACLs): Өгөгдлийн сангийн обектийг хамгаалах

Таны блог програмын нэг блогын бичлэг дээр хэрэглэгч сэтгэгдэл бичдэг байг. Хэрэглэгч өөрийн бисэн сэтгэгдэлээ засах боломжтой ба харин өөр бусад хэрэглэгч түүнийг засах боломжгүй байх ёстой. Бас админ хэрэглэгч бүх сэтгэгдэлийг засах боломжтой байна.

Security компонент нь access control list (ACL)  -ийг агуулдаг. ACL ашиглаагүй тохиолдолд сэтгэгдэлийг зөвхөн итгэлтэй хэрэглэгчид л засварлах боломжтойгоор хийж болно. Харин ACL ашиглаад тухайн сэтгэгдэл рүү хандахыг зөвшөөрөх болон хориглох болмжтой болгох юм.

## Хэрэглэгч

Энэ бүлэгт authorization –ийг хэрэгжүүлэх өөр нэг хэсэг болох хэрэглэгчийн тухай үзэх болно.

### Хэрэглэгч хаанаас ирдэг вэ? (User Providers)

Symfony2 – т хэрэглэгч нь тохиргооны файл, өгөгдлийн сан, вэб сервес, гэх мэт өөр хаанаас хандаж болно.

```
# app/config/security.yml
security:
    # ...
    providers:
        default_provider:
            memory:
                users:
                    ryan: { password: ryanpass, roles: 'ROLE_USER' }
                    admin: { password: kitten, roles: 'ROLE_ADMIN' }
```

Энэ user provider-ийг "in-memory" гэж нэрлэх ба энэ хэрэгэлгч өгөгдлийн санд хадгалагдахгүй. Жинхэнэ user обектийг Symfony гаргаж өгдөг. (User[35])

User provider нь тохиргооны файлд users параметерт заасан хэрэглэгчидийг шууд дуудна.

Жижиг хэмжээний сайтад энэ арга нь маш хурдан ажиллах ба суулгахад ашиглахад хялбархан байна. Харин илүү цогц системийн хувьд өгөгдлийн сангаас хэрэглэгчийг дуудан ажиллуулах боломжтой.

### Өгөгдлийн сангаас хэрэглэгчийг дуудах

Doctrine ORM ашиглан хэрэглэгчийг дуудахын тулд **User** классыг үүсгэн, **entity** provider-ийг тохируулах хэрэгтэй.

[35] http://api.symfony.com/master/Symfony/Component/Security/Core/User/User.html

Doctrine ORM эсвэл ODM ашиглан өгөгдлийн санд хадгалаатай байгаа хэрэглэгчтэй ажиллах маш сайн чанартай нээлттэй эхийн bundle-ийг ашиглах боломжтой. GitHub дээрх FOSUserBundle[36] -ийг үзнэ үү.

Өгөгдлийн санд байх хэрэглэгчтэй ажиллах User классыг үүсгэе.

```php
// src/Acme/UserBundle/Entity/User.php
namespace Acme\UserBundle\Entity;

use Symfony\Component\Security\Core\User\UserInterface;
use Doctrine\ORM\Mapping as ORM;
/**
* @ORM\Entity
*/
class User implements UserInterface
{
    /**
    * @ORM\Column(type="string", length=255)
    */
protected $username;
    // ...
}
```

Хамгаалалтын систем нь UserInterface[37]-ээс implement хийгдсэн энэ класстай шаардлагатай үед л холбогдоно.

User обект нь request илгээж байх явцад **serialize** хийгдсэн **session**-д хадгалагдсан байна.

Дараа нь **entity** user provider-ийг тохируулж **User** классаа зааж өгнө.

```yaml
# app/config/security.yml
security:
    providers:
        main:
            entity:
                class: Acme\UserBundle\Entity\User
                property: username
```

Хэрэглэгчийг танин нэвтрүүлэх систем нь тухайн классын **username** –ийг ашиглан **User** обектийг өгөгдлийн сангаас дуудна.

[36] https://github.com/FriendsOfSymfony/FOSUserBundle
[37] http://api.symfony.com/master/Symfony/Component/Security/Core/User/UserInterface.html

### Хэрэглэгчийн нууц үгийг encoding буюу кодлох

Хамгаалалтын зорилгоор хэрэглэгчийн нууц үгийг кодлох хэрэгтэй. built-in "encoders" –ийн нэгийг ашиглаж болно.

```
# app/config/security.yml
security:
    # ...
    providers:
        in_memory:
            memory:
                users:
                    ryan:
                    password: $2a$12$w/aHvnC/
XNeDVrrl65b3dept8QcKqpADxUlbraVXXsC03Jam5hvoO
                    roles: 'ROLE_USER'
                admin:
                    password:
$2a$12$HmOsqRDJK0HuMDQ5Fb2.AOLMQHyNHGD0seyjU3lEVusjT72QQEIpW
                    roles: 'ROLE_ADMIN'

    encoders:
        Symfony\Component\Security\Core\User\User:
            algorithm: bcrypt
            cost: 12
```

BCrypt encoder –ийг Symfony 2.2-т танилцуулсан.

Одоо та hash хийгдсэн нууц үгийг програмын аргаар (жнь: **password_hash('ryanpass', PASSWORD_BCRYPT, array('cost' => 12));**) эсвэл онлайн хэрэгсэл ашиглан тооцоолж болно.

Хэрэв та PHP5.4 эсвэл түүнээс бага хувилбар ашиглаж байгаа бол bcrypt encoder –ийг ашиглахын тулд **ircmaxell/password-compat** санг суулгах шаардлагатай.

Таны ашиглаж буй PHP хувилбарт ямар алгоритм тохирохыг **hash_algos**[38] жагсаалтаас харна уу.

### Hashed нууц үгийг тодорхойлох

Хэрэв өгөгдлийн санд хэрэглэгчийг хадгалж байгаа бөгөөд хэрэглэгчийг бүртгэлийн форм ашиглан хадгалж байгаа бол хэрэглэгчийн нууц үгийг өгөгдлийн сан руу оруулахын өмнө hash хийгдсэн нууц үгийг тодорхойлох хэрэгтэй. Инэснээр user обектод ямар алгоритм ашигласанаас үл хамааран hash хийгдсэн нууц үгийг controller дотор тодорхойлж болно.

[38] http://php.net/manual/en/function.hash-algos.php

```php
$factory = $this->get('security.encoder_factory');
$user = new Acme\UserBundle\Entity\User();

$encoder = $factory->getEncoder($user);
$password = $encoder->encodePassword('ryanpass', $user->getSalt());
$user->setPassword($password);
```

Хэрэв энэ ажилхгүй байвал app/config/security.yml файлын **encoders**-т тохируулсан encoder-ийг user классдаа хийж өгнө.

### User обектийг авах

Хэрэглэгчийг танин нэвтрүүлсэний дараа User обект руу **security.context** сервесээр хандах боломжтой болно.

```php
public function indexAction()
{
    $user = $this->get('security.context')->getToken()->getUser();
}
```

Twig темплэйтэд энэ обект руу хандахдаа **app.user** -түлхүүр үгтийг ашиглах ба ***GlobalVariables::getUser()***[39] –ийг дуудна.

```twig
<p>Username: {{ app.user.username }}</p>
```

## Role

Хэрэглэгч бүрт өөрийн гэсэн эрх үүрэг олгосон байх ба програмын хэсэг бүрт хандахын тулд нэг эсвэл хэд хэдэн role хэрэгтэй байдаг.

Жишээ нь: Хэрэв вэб сайтын блог админ хэсэг рүү хандах хандалтыг хязгаарлах бол ROLE_BLOG_ADMIN гэсэн role-ийг ашиглаж болно.

Бүх role нь Symfony2 –т **ROLE_** гэсэн үгээр эхэлнэ.

### Шаталсан Role

Хэрэглэгчидэд олон тооны role олгохын оронд шаталсан role үүсгэж удамшуулан ашиглаж болно.

```yaml
# app/config/security.yml
security:
    role_hierarchy:
        ROLE_ADMIN: ROLE_USER
        ROLE_SUPER_ADMIN: [ROLE_ADMIN, ROLE_ALLOWED_TO_SWITCH]
```

[39] http://api.symfony.com/master/Symfony/Bundle/FrameworkBundle/Templating/GlobalVariables.html#getUser()

Энэ тохиргоонд **ROLE_ADMIN** эрхтэй хэрэглэгч мөн **ROLE_USER** эрхтэй байна. **ROLE_SUPER_ADMIN** нь **ROLE_ADMIN, ROLE_ALLOWED_TO_SWITCH** болон **ROLE_USER** эрхтэй байна. **ROLE_USER** нь **ROLE_ADMIN** –аас удамшиж ирсэн байна.

### Expression ашигласан Access Control

**isGranted** method нь ROLE_ADMIN-аас гадна бас Expression[40] обектийг авч болно.

```
use Symfony\Component\Security\Core\Exception\AccessDeniedException;
use Symfony\Component\ExpressionLanguage\Expression;
// ...

public function indexAction()
{
    if (!$this->get('security.context')->isGranted(new Expression(
        '"ROLE_ADMIN" in roles or (user and user.isSuperAdmin())'
    ))) {
        throw new AccessDeniedException();
    }
    // ...
}
```

Энэ жишээнд хэрэв хэрэглэгч нь ROLE_ADMIN эрхтэй, эсвэл isSuperAdmin() method нь true утга буцааж байвал тухайн хэрэглэгчийн хандалтыг зөвшөөрнө.

### Темплэйтэд Access Control ашиглах

Темплэйт дотор тухайн хэрэглэгч ямар role –той байгааг шалгах бол дараах built-in функцийг ашиглана.

```
{% if is_granted('ROLE_ADMIN') %}
    <a href="...">Delete</a>
{% endif %}
```

Мөн та темплэйтдээ expression ашиглаж болно.

```
{% if is_granted(expression(
    '"ROLE_ADMIN" in roles or (user and user.isSuperAdmin())'
)) %}
    <a href="...">Delete</a>
{% endif %}
```

## Logging Out буюу гарах

logout параметерийг идэвхжүүлэх үед firewall нь автоматаар энэ үйлдлийг гүйцэтгэнэ.

```
# app/config/security.yml
security:
    firewalls:
```

[40] http://api.symfony.com/master/Symfony/Component/ExpressionLanguage/Expression.html

```
        secured_area:
            # ...
            logout:
                path: /logout
                target: /
    # ...
```

Хэрэглэгч **/logout** руу хандсанаар нэвтэрсэн байгаа хэрэглэгчийг системээс гаргаж, сайтын нүүр хуудас руу шилжүүлэх болно.

Log out хийсний дараа хэрэглэгчийг **target** параметерт зааж өгсөн зам руу (жнь: **homepage**) шилжүүлнэ.

# Хэрэгсэлүүд

Symfony Security component нь хамгаалаттай холбоотой хэрэгсэлүүдийг агуулдаг.

### String-үүдийг харьцуулах

Нууц үгийг баталгаажуулах зорилгоор 2 string-ийг харьцуулах үед хакерууд үүнийг ашиглан халдлага хийх боломжтой байдаг. Ийм төрлийн халдлагыг **Timing Attack**[41] гэдэг.

Хоёр нууц үгийг харьцуулахдаа Symfony нь constant-time алгоритмыг ашигладаг.

```
use Symfony\Component\Security\Core\Util\StringUtils;

// is password1 equals to password2?
$bool = StringUtils::equals($password1, $password2);
```

### Хамгаалалтын санамсаргүй тоо үүсгэх

Хамгаалалтын санамсаргүй тоо үүсгэхдээ **SecureRandom**[42] классыг ашиглана.

```
use Symfony\Component\Security\Core\Util\SecureRandom;

$generator = new SecureRandom();
$random = $generator->nextBytes(10);
```

**nextBytes()**[43] method нь аргементэд өгсөн тоог (жнь: 10) ашиглан үүсгэсэн санамсаргүй тоог буцаана.

[41] http://en.wikipedia.org/wiki/Timing_attack
[42] http://api.symfony.com/master/Symfony/Component/Security/Core/Util/SecureRandom.html
[43] http://api.symfony.com/master/Symfony/Component/Security/Core/Util/SecureRandom.html#nextBytes()

14-р бүлэг

# HTTP кэйш

Баялаг вэб сайтын гол зарчим бол динамик байх явдал юм. Энэ нь тухайн програмын request бүр статик файлаас илүү хэмжээний мэдээллийг агуулна. Ихэнх вэб програмуудад энэ бол асуудал биш. Гэхдээ програм томорч өсөөд ирэхээр энэ нь асуудал болж эхэлдэг.

## Shoulders of Giants дээр кэйшлэх

Програмын ажиллагааг илүү сайжруулахын тулд тухайн хуудасны үр дүнг  кэйшлэх буюу нөөцлөх хэрэгтэй байдаг.

Энэ бүлэгт Symfony2 кэйш систем хэрхэн ажилладаг талаар үзнэ. Symfony2 кэйш систем нь HTTP specification –д заасан HTTP cache дээр тулгуурлан ажилладаг.

Symfony2 – т кэйш нь хэрхэн ажилладаг талаар судлахын тулд дараах сэдэвийг үзэх болно.

1. gateway cache буюу reverse proxy
2. HTTP cache
3. Edge Side Includes (ESI)

## Gateway Cache ашиглах

HTTP ашиглан кэйш хийх үед кэйш нь таны програмаас бүрэн тусгаарлагдаж, програм болон request илгээж буй клейнт хоёрын хооронд байрлана.

Кэйшийн үүрэг бол клейнтээс ирж байгаа request-ийг хүлээн авч, түүнийг програмд дамжуулна. Мөн буцаагаад програмаас response-ийг хүлээн авч, клейнт рүү буцаана.

Үүний зэрэгцээ кэйш нь нөөцлөх боломжтой response-ийг хадгалж авна. Ингэснээр дараа тухайн request дахин орж ирхэд програм руу хандалгүйгээр түүнд тохирох нөөцөлсөн response-ийг клейнт рүү илгээнэ.

Ийм төрлийн кэйшийг **HTTP gateway cache** гэж нэрлэнэ.

### Кэйшийн төрөлүүд

Гэхдээ gateway cache бол кэйшийн ганц төрөл биш юм. Програмын илгээж байгаа HTTP cache header нь кэйшийн гурван төрлөөр хөрвүүлэгдэн ашиглагдана.

- Browser caches: Браузер бүр өөрийн гэсэн локал кэйштэй байна. Браузер кэйш бол private кэйш учираас нөөцөлсөн агуулгыг хэн нэгэнтэй хуваалцах боломжгүй.
- Proxy caches: Прокси бол олон хүнтэй хуваалцах боломжтой кэйш юм. Энэ нь ихэвчлэн томоохон корпрациуд, болон ISP комданиудад сүлжээний сааталыг багасгах зорилгоор хэрэглэгддэг.
- Gateway cashe: Proxy –тай ижил shared кэйш боловч энэ нь сервер тал дээр ажиллана. Иймд үүнийг системийн админ суулгах ба үүнийг ашигласнаар вэб сайт илүү уян хатан, ажиллагаа сайтай болно.

Програмд эхний 2 кэйшийн аль нэгийг хэрэглэнэ. Эдгээр кэйшүүд нь таны хяналтын гадна байх боловч дараах HTTP cache зааврыг ашиглан response –д кэйшийг тохируулна.

### Symfony2 Reverse Proxy

Symfony2 нь PHP дээр бичигдсэн reverse proxy (өөрөөр gateway cache гэж нэрлэнэ. ) –той байдаг. Үүнийг идэвхтэй болгосоноор програмаас буцаагдах response кэйшлэгдэнэ. Symfony2 програм нь урьдчилан тохируулсан caching kernel (AppCache)-тэй байна. Товчихондоо бол caching Kernel нь reverse proxy юм.

Кэйшийг идэвхтэй болгохын тулд caching kernel ашиглаж байгаа front controller-ийн кодонд өөрчилөлт хийнэ.

```
1  // web/app.php
2  require_once __DIR__.'/../app/bootstrap.php.cache';
3  require_once __DIR__.'/../app/AppKernel.php';
4  require_once __DIR__.'/../app/AppCache.php';
5
6  use Symfony\Component\HttpFoundation\Request;
7
8  $kernel = new AppKernel('prod', false);
9  $kernel->loadClassCache();
10 // wrap the default AppKernel with the AppCache one
11 $kernel = new AppCache($kernel);
12 $request = Request::createFromGlobals();
13 $response = $kernel->handle($request);
14 $response->send();
15 $kernel->terminate($request, $response);
```

Ингээд caching kernel нь програмын буцааж байгаа response-ийг кэйшилж аваад, түүнийгээ клейнт рүү буцаах үүрэгтэй reverse proxy-ийг ажиллуулна.

cache kernel нь getLog() гэх тусгай төрлийн method-той байх ба энэ нь кэйш давхарга дээр юу болж байгааг харуулах мессежийг буцаадаг.

```
1  error_log($kernel->getLog());
```

**AppCache** обект default тохиргоотой байх ба үүнийг getOptions() method-ийг оруулж ирсэнээр өөрийн хүссэнээр тохируулах боломжтой.

```php
// app/AppCache.php
use Symfony\Bundle\FrameworkBundle\HttpCache\HttpCache;

class AppCache extends HttpCache
{
    protected function getOptions()
    {
        return array(
            'debug' => false,
            'default_ttl' => 0,
            'private_headers' => array('Authorization', 'Cookie'),
            'allow_reload' => false,
            'allow_revalidate' => false,
            'stale_while_revalidate' => 2,
            'stale_if_error' => 60,
        );
    }
}
```

## HTTP кэйшийн тухай

Кэйш давхаргыг ашиглахын тулд таны програм кэйш хийгдэх боломжтой response-ууд, болон хуучирсан кэйшийг хэрхэн зохицуулахыг шийдэх дүрэмүүдтэй харилцах боломжтой байх ёстой. Ингэхийн тулд response дээр HTTP cache header –ийг тохируулах хэрэгтэй.

HTTP кэйш нь клейнт болон серверийг кэштэй холбоотой мэдээллийг өөрчилөх боломж олгоно.

HTTP кэйш нь дараах 4 төрлийн response cache header-ийг тодорхойлно.

- Cache-Control
- Expires
- Etag
- Last-Modified

Хамгийн чухал, өргөн хүрээнд хэрэглэгддэг header бол Cache-Control header юм. Энэ нь олон төрлийн кэйшийн мэдээллийн цуглуулга юм.

### Cache-Control Header

Cache-Control Header нь кэйш хийгдэх боломжтой олон тооны мэдээллийн хэсгүүдийг агуулдаг. Эдгээр мэдээллийн хэсэг бүр нь таслалаар тусгаарлагдсан байна.

```
Cache-Control: private, max-age=0, must-revalidate

Cache-Control: max-age=3600, must-revalidate
```

Symfony нь Cache-Control header –ийг илүү зохион байгуулалттайгаар үүсгэх боломж олгоно.

```php
// ...

```

```php
use Symfony\Component\HttpFoundation\Response;

$response = new Response();

// mark the response as either public or private
$response->setPublic();
$response->setPrivate();

// set the private or shared max age
$response->setMaxAge(600);
$response->setSharedMaxAge(600);

// set a custom Cache-Control directive
$response->headers->addCacheControlDirective('must-revalidate', true);
```

### Public ба private response

**gateway** болон **proxy cache** хоёулаа хадгалсан кэйшийн агуулгыг бусад хэрэглэгчидтэй хуваалцах боломжтой байдаг. Хэрэв хэрэглэгчийн тодорхойлсон response –ийг shared cache ашиглан хадгалсан бол энэ нь дараа бусад хэрэглэгч рүү буцаагдах болно. Хэрэв та хэрэглэгчийн нэвтрэх мэдээлээ энд кэйшилсэн бол өөр хэн нэг нь нэвтрэх хуудсыг дуудах үед тэр мэдээллийг тань буцаана.

Энэ байдлыг үүсгэхгүйн тулд response бүрийг public эсвэл private –ээр тохируулах хэрэгтэй.

- *Public:* response нь private болон shared кэйш хоёулангаар нь кэйшлэгдэх боломжтойг заана.
- *Private:* Зөвхөн нэг хэрэглэгчид зориулан response мессежийг бүхэлд нь эсвэл нэг хэсэгийг заах ба shared кэйшээр кэйш хийх ёсгүй.

Symfony-д default-аар response нь private байдаг. Shared кэйшийг ашиглахын тулд response-ийг public болгох хэрэгтэй юм.

### Safe Method

HTTP кэйш нь зөвхөн “аюулгүй” HTTP method-той ажиллана. Аюулгүй байна гэдэг нь request боловсруулж байх үед сервер дээрх програмын төлөвийг өөрчилөхгүй байна гэсэн үг юм.

- GET эсвэл HEAD request-д хариу үзүүлэх үед програмын төлөв өөрчилөгдөх ёсгүй.
- PUT, POST, DELETE method –уудыг кэйш хийж болохгүй. Эдгээр method-ууд нь програмын төлөвийг өөрчилдөг (blog post-ийг устгах гэх мэт).

### Rule ба Default кэйш

HTTP 1.1 нь default-аар ямар ч кэйш хийх боломжтой ба энэ нь Cache-Control header –ийг агуулдаг. Практик дээр, ихэнх кэйшүүд cookie, authorization header, non-safe method (PUT, POST, DELETE) ашигласан үед ямар нэг үйлдэл хийдэггүй.

Хөгжүүлэгч нь дараах дүрмээр тохиргоо хийгээгүй үед Symfony2 нь автоматаар Cache-Control header-ийг тохируулдаг.

- Хэрэв cache header (**Cache-Control, Expires, Etag, Last-Modified**) тодорхойлоогүй тохиолдолд Cache Control нь no-cache байх ба энэ нь response-ийг кэйшлэхгүй гэсэн үг юм.
- Хэрэв Cache-Control хоосон байвал энэ нь **private, must-revalidate** утгатай байна.
- Харин хамгийн багадаа нэг Cache-Control тохируулсан бол Symfony нь түүнд private –ийг автоматаар олгоно.

## Edge Side Include ашиглах

Gateway кэйш нь вэб сайтын ажиллагааг сайжруулах хамгийн сайн арга юм. Гэвч энэ нь хуудсыг бүхэлд нь кэйшилдэг. Харин карим тохиолдолд хуудсыг бүхэлд нь бус зарим динамик хэсгүүдийг л кэйшлэх хэрэг гардаг. Тиймээс Symfony2 нь **ESI буюу Edge Side Include** гэх технологи ашиглан энэ асуудлыг шийдвэрлэнэ.

ESI tag-ууд нь gateway кэйштэй холбоотой хуудсанд ашиглагдана. Symfony2-т ганцхан **include** tag ашиглана.

```
<!DOCTYPE html>
<html>
    <body>
        <!-- ... some content -->

        <!-- Embed the content of another page here -->
        <esi:include src="http://..." />

        <!-- ... more content -->
    </body>
</html>
```

### Symfony2-т ESI ашиглах

Эхлээд, ESI ашиглахын тулд програмын тохиргоонд түүнийг идэвхтэй болгох хэрэгтэй.

```
# app/config/config.yml
framework:
    # ...
    esi: { enabled: true }
```

Жишээ нь, танд нэг статик хуудас байгаа бөгөөд энэ нь контентийн доод талд мэдээ гүйлгэгчийг харуулна. ESI ашиглан тухайн статик хуудаснаас мэдээ гүйлгэгчийг түсд нь кэйшлэх боломжтой.

```
public function indexAction()
{
    $response = $this->render('MyBundle:MyController:index.html.twig');
    // set the shared max age - which also marks the response as public
    $response->setSharedMaxAge(600);

    return $response;
}
```

Энэ жишээнд хуудсыг бүхэлд нь 10 минутын турш кэйшилнэ. Дараа нь, темплэйтэд Action ашигласнаар мэдээ гүйлгэгчийг оруулж ирнэ. Үүнийг render функц ашиглан хийнэ.

Өөр хуудаснаас агуулгыг оруулж ирэхдээ Symfony2-т ESI tag-д render функцыг ашиглана.

```
{# you can use a controller reference #}
{{ render_esi(controller('...:news', { 'max': 5 })) }}

{# ... or a URL #}
{{ render_esi(url('latest_news', { 'max': 5 })) }}
```

render_esi –г ашигласнаар тухайн Action-ийг ESI tag-аар боловсруулах ёстой гэдэгийг заана.

Render функц ашиглаж байгаа үед Symfony2 нь клейнт рүү response илгээхийн өмнө үндсэн агуулгатай гаднаас орж ирсэн агуулгыг нэгтгэнэ.

Харин action нь өөрийн кэйшинг дүрмийг ашиглан мастер хуудсыг бүхэлд нь тусгаарлана.

```
public function newsAction($max)
{
    // ...
    $response->setSharedMaxAge(60);
}
```

ESI ашиглан 600 секунд хуудсыг бүхэлд нь кэйшилнэ. Гэхдээ мэдээний агуулга нь сүүлийн 60 секунд л кэйшлэгдэнэ.

Controller заагч ашиглах тохиолдолд ESI tag нь URL-аар хандах action-ийг заах ёстой. Иймээс gateway кэйш нь хуудсыг тусд нь авна. Symfony2 нь controller руу заах дахин давтагдахгүй URL үүсгэх ба тохиргооны файлд **FragmentListener**[44]-ийг тохируулна.

```
# app/config/config.yml
framework:
    # ...
    fragments: { path: /_fragment }
```

[44] http://api.symfony.com/master/Symfony/Component/HttpKernel/EventListener/FragmentListener.html

Бүлэг 15

# Орчуулга

Энэ бүлэгт Symfony2-т Translator component-ийг хэрхэн ашиглахыг үзэх болно.

### Тохируулга

Програмын орчуулгыг хийхийн тулд хэрэглэгчийн **locale** (хэл ба орон)–ийг хайх, орчуулсан мессежийг буцаах үүрэгтэй **translator** сервесийг ашиглана. Үүнийг ашиглахын өмнө тохиргооны файлд **translator** –ийг идэвхтэй болгох хэрэгтэй.

```
#app/config/config.yml
framework:
    translator: { fallback: en }
```

**fallback** –ийн тухай дараа нь үзэх болно.

Орчуулгад ашиглагдах locale нь request-д хадгалагдсан байна. Энэ нь ерөнхийдөө **_locale** гэсэн аттрибутаар тохируулагдсан байна.

## Үндсэн орчуулга

Текстийн орчуулгыг **translator**[45] service ашиглан гүйцэтгэнэ. Текстийг орчуулахдаа **trans()**[46] method-ийг ашиглана. Жишээ нь: controller дотор энгийн мессеж орчуулж үзье.

```
// ...
use Symfony\Component\HttpFoundation\Response;

public function indexAction()
{
    $translated = $this->get('translator')->trans('Symfony2 is great');
    return new Response($translated);
}
```

[45] http://api.symfony.com/master/Symfony/Component/Translation/Translator.html
[46] http://api.symfony.com/master/Symfony/Component/Translation/Translator.html#trans()

Энэ код ажиллах үед Symfony2 нь хэрэглэгчийн locale дээр тулгуурлан "Symfony2 is great" мессежийг орчуулах оролдлого хийнэ. Инэгхийн тулд өгсөн locale-ийн дагуу орчуулгын жагсаалтуудыг агуулж байгаа файлыг ашиглан тухайн мессежийг хэрхэн орчуулах зааврыг Symfony2 –т зааж өгнө. Энэ нь нэг ёсондоо хэд хэдэн төрлийн форматтайгаар үүсгэж болох толь бичиг бөгөөд танд харин XLIFF форматыг ашиглахыг зөвлөж байна.

```
<!-- messages.fr.xliff -->
<?xml version="1.0"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
    <file source-language="en" datatype="plaintext" original="file.ext">
        <body>
            <trans-unit id="1">
                <source>Symfony2 is great</source>
                <target>J'aime Symfony2</target>
            </trans-unit>
        </body>
    </file>
</xliff>
```

Энэ файлыг хаана үүсгэхийг “Орчуулгын файлын нэр ба байрлал” гэсэн сэдвээс үзнэ үү.

Одоо, харин хэрэглэгчийн locale хэл нь Франц хэл (**fr_FR** эсвэл **fr_BE**) бол мессежийг **J'aime Symfony2** гэж орчуулах болно.  Мөн темплэйт дотор мессеж орчуулах боломжтой.

### Орчуулах үйл ажиллагаа

Symfony2 –т мессежийг дараах алхамуудаар орчуулна.

- Request-д хадгалагдаж байгаа хэрэглэгчийн locale-ийг тодорхойлох
- Орчуулсан мессежийн каталогийг local (жнь: fr_FR) дээр тулгуурлан тодорхойлсон файлаас уншина. fallback locale –ээс мөн мессежүүдийг унших ба хэрэв тухайн мессеж каталогит байхгүй бол түүн рүү мессежийг нэмнэ. Үүний үр дүнд их хэмжээний толь бичиг бий болно.
- Хэрэв тухайн мессеж каталогит байвал түүний орчуулгыг буцаана. Харин каталогит байхгүй бол анхны мессежийг буцаана.

## Message Placeholder

Заримдаа, хувьсагчийн агуулж байгаа мессежийг орчуулах хэрэгтэй байдаг.

```
use Symfony\Component\HttpFoundation\Response;

public function indexAction($name)
{
    $translated = $this->get('translator')->trans('Hello '.$name);

    return new Response($translated);
}
```

Гэхдээ энэ String-д зориулсан орчуулгыг үүсгэх боломжгүй учир translator нь тохирох мессежийг олохыг хичээх болно.

## Pluralization

Өөр нэг хүндхэн асуудал бол цифр болон үгэн тоог орчуулах.

```
1  There is one apple.
2  There are 5 apples.
```

Үүнийг хийхдээ **transChoice()**[47] method эсвэл темплэйтэд **transchoice** tag ашиглан хийнэ.

## Темплэйт дотор орчуулга хийх

Ихэнх тохиолдолд орчуулгыг темплэйт дотор хийж өгдөг.

### Twig темплэйт

Symfony2 нь Twig tag (**trans** болон **transchoice**) -ийн тусламжтайгаар текстийг орчуулдаг.

```
1  {% trans %}Hello %name%{% endtrans %}
2
3  {% transchoice count %}
4      {0} There are no apples|{1} There is one apple|]1,Inf] There are
   %count% apples
5  {% endtranschoice %}
```

**transchoice** tag нь автоматаар **%count%** хувьсагчийг авч translator- луу дамжуулна. Энэ механизм нь **%var%** загварын дагуу байгаа үед л ажиллана.

Twig темплэйтэд tag ашиглан орчуулах үед %var% гэсэн энэ тэмдэглэгээг шаардана.

Хэрэв тухайн текстэнд хувь тэмдэг (%) ашиглах бол давхар (%) тэмдэг ашиглана.

**{% trans %}Percent: %percent%%%{% endtrans %}**

Мөн тухайн мессежид нэр оноон өгч нэмэлт хувьсагчуудыг дамжуулж болно.

```
1  {% trans with {'%name%': 'Fabien'} from "app" %}Hello %name%{% endtrans %}
2
3  {% trans with {'%name%': 'Fabien'} from "app" into "fr" %}Hello %name%{%
   endtrans %}
4
5  {% transchoice count with {'%name%': 'Fabien'} from "app" %}
6      {0} %name%, there are no apples|{1} %name%, there is one apple|]1,Inf]
   %name%, there
7  are %count% apples
8  {% endtranschoice %}
```

[47] http://api.symfony.com/master/Symfony/Component/Translation/Translator.html#transChoice()

**trans** and **transchoice** filter –үүд нь хувьсагчын мессежүүдийг орчуулахад ашиглагдана.

```
{{ message|trans }}

{{ message|transchoice(5) }}

{{ message|trans({'%name%': 'Fabien'}, "app") }}

{{ message|transchoice(5, {'%name%': 'Fabien'}, 'app') }}
```

Хэрэв орчуулсан мессеждээ ямар нэг шүүлтгүйгээр (output escape) гаргахыг хүсвэл translation filter-ийн дараа **raw** filter ашиглах хэрэгтэй.

Listing

```
{# text translated between tags is never escaped #}
{% trans %}
    <h3>foo</h3>
{% endtrans %}

{% set message = '<h3>foo</h3>' %}

{# strings and variables translated via a filter are escaped by default #}
{{ message|trans|raw }}
{{ '<h3>bar</h3>'|trans|raw }}
```

### PHP темплэйт

Translator сервесийг мөн PHP темплэйтэд ашиглах боломжтой.

```
<?php echo $view['translator']->trans('Symfony2 is great') ?>

<?php echo $view['translator']->transChoice(
'{0} There are no apples|{1} There is one apple|]1,Inf[ There are %count% apples',
10,
array('%count%' => 10)
) ?>
```

## Орчуулгын файлын нэр ба байрлал

Symfony2 нь дараах байрлалуудаас мессежийн файлуудыг хайдаг.

- **app/Resources/translations** хавтас
- **app/Resources/<bundle name>/translations** хавтас
- Bundle-ийн **Resources/translations/** хавтас

Энд хамгийн их зэрэгтэйгээр нь дарааллуулан жагсаасан болно. Иймд дээд талын 2 хавтасны аль нэгээс bundle-ийн орчуулгын мессежийг оруулж ирэх боломжтой.

Энэ механизм нь key level ашиглан ажиллана. Эдгээр үгүүд нь өндөр зэрэгтэй мессеж файлд байх хэрэгтэй. Мессеж файлд тухайн key олдоогүй тохиолдолд translator нь бага зэрэгтэй мессеж файлыг буцаана.

Орчуулгын файлын нэр нь мөн маш их чухал ач холбогдолтой. Мессеж файл нь дараах дүрмийн дагуу нэрлэгдсэн байх ёстой. **domain.locale.loader:**

- domain: Мессежүүдийг бүлэглэн зохион байгуулах нэмэлт арга
- locale: орчуулгыг заах locale (**en_GB, en** гэх мэт)
- loader: Symfony2 нь ямар файлыг дуудан ажиллуулахыг заана.. (**xliff, php, yml** гэх мэт)

Мөн орчуулгыг өгөгдлийн санд эсвэл класс үүсгэн орчуулах бас боломжтой байдаг. Хэрэв класс үүсгэх бол тэр класс нь LoaderInterface –ээс удамшсан байх шаардлагатай.

## Fallback Translation Locale

Хэрэв хэрэглэгчийн locale нь **fr_FR** байгаа тохиолдолд "**Symfony2 is great**" гэсэн key ашиглан орчуулах хэрэгтэй байлаа гэж төсөөлөө. Францаар орчуулахын тулд доох locale-үүдийг шалгаж үзнэ.

1. Эхлээд Symfony нь **fr_FR** гэсэн local-тэй орчуулгын файлыг хайна. (жнь: **messages.fr_FR.xliff**)
2. Хэрэв **fr_FR** олдохгүй бол **fr** –ийг хайна. (жнь: **messages.fr.xliff**)
3. Энэ 2 аргаар мөн л олдохгүй бол **fallback** тохиргооны параметерийг ашиглан default –аар **en** locale-ийг ашиглана.

## Хэрэглэгчийн locale-ийг боловсруулах

Тухайн хэрэглэгчийн locale нь request-д хадгалагдсан байдаг бөгөөд түүн рүү **request** обектоор хандана.

```
1  use Symfony\Component\HttpFoundation\Request;
2
3  public function indexAction(Request $request)
4  {
5      $locale = $request->getLocale();
6
7      $request->setLocale('en_US');
8  }
```

### Locale ба URL

Хэрэглэгчийн locale-ийг мөн session-д хадгалж болно. Ингэснээр нэг ижил URL –ээр хэрэглэгчийн locale-ээс хамаараад өөр өөр хэлээр сайтыг харуулах боломжтой юм. Жишээ нь: **http://www.example.com/contact** гэсэн URL-аар нэг хэрэглэгч Англи хэлээр үзэх ба өөр нэг хэрэглэгч бас Франц хэлээр үзэж болно. Харамсалтай нь энэ арга нь вэбийн үндсэн зарчимтай зөрчилдөнө. Тухайлбал тодорхой URL ижил зүйлийг буцаадаг.

Хамгийн сайн арга бол URL-д locale-ийг оруулах. Энэ нь routing системд тусгай төрлийн **_locale** параметер ашиглан биелэгдэнэ.

```yaml
contact:
    path: /{_locale}/contact
    defaults: { _controller: AcmeDemoBundle:Contact:index }
    requirements:
        _locale: en|fr|de
```

Энэ тохиолдолд Request дээр тухайн locale-ийг автоматаар тохируулах ба **getLocale()** method –оор түүнийг авах боломжтой. Өөрөөр хэлбэл, хэрэв хэрэглэгч **/fr/contact** гэсэн URL-аар хандвал **fr** locale нь тухайн ирж байгаа request дээр тохируулагдана.

### default Locale тохируулах

Хэрэв хэрэглгчийн locale тодорхойгүй бол яах вэ? Ийм тохиолдолд тухайн locale-ийг **default_locale** тодорхойлох хэрэгтэй.

```yaml
# app/config/config.yml
framework:
    default_locale: en
```

## Constraint Message орчуулах

Хэрэв формд Validation constraint ашиглаж байгаа бол алдааны мессежийг орчуулж болно. **validators** гэсэн domain ашиглаад орчуулгын файлыг үүсгэнэ.

Жишээ нь, програмынхаа ашиглагдах шаардлагатай нэг PHP обект байна.

```php
// src/Acme/BlogBundle/Entity/Author.php
namespace Acme\BlogBundle\Entity;

class Author
{
    public $name;
}
```

Одоо түүнд constraint нэмээд орчуулах тескстээ message тохиргоонд зааж өгнө. Жишээ нь: дараах кодонд **$name** property хоосон утга авч болохгүй гэдэгийг заасан байна.

```yaml
# src/Acme/BlogBundle/Resources/config/validation.yml
Acme\BlogBundle\Entity\Author:
    properties:
        name:
            - NotBlank: { message: "author.name.not_blank" }
```

Дараа нь constraint мессежид зориулан validator каталог дотор орчуулгын файлаа үүсгэнэ. Энэ нь ерөнхийдөө bundle-ийн **Resources/translations/** хавтсанд байрладаг.

```xml
<!-- validators.en.xliff -->
<?xml version="1.0"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
    <file source-language="en" datatype="plaintext" original="file.ext">
        <body>
            <trans-unit id="1">
```

```
                <source>author.name.not_blank</source>
                <target>Please enter an author name.</target>
            </trans-unit>
        </body>
    </file>
</xliff>
```

## Өгөгдлийн сангийн агуулгыг орчуулах

Өгөгдлийн сан дахь агуулгыг орчуулахын тулд Translatable Extension эсвэл Translatable Behavior сангуудыг Doctrine –тай ашиглах хэрэгтэй. Эдгээр сангуудын тухай та өөрөө дэлгэрүүлж судлаарай.

## Орчуулгад алдаа илрүүлэх

**translation:debug** командыг Symfony 2.5-д танилцуулсан.

**translation:debug** командыг ашиглан хэрэглэгдэхгүй байгаа эсвэл эвдэрсэн мессежүүдийг олох боломжтой. Энэ нь өгсөн locale-ийн дагуу гарсан үр дүнг болон fallback ашиглан гарсан үр дүн зэргийг хүснэгтэн хэлбэрээр харуулна.

Twig тмеплэйтэд translation tag эсвэл filter ашиглан алдааг илрүүлнэ.

```
{% trans %}Symfony2 is great{% endtrans %}

{{ 'Symfony2 is great'|trans }}

{{ 'Symfony2 is great'|transchoice(1) }}

{% transchoice 1 %}Symfony2 is great{% endtranschoice %}
```

Мөн PHP темплэйтэд translator ашиглан илрүүлнэ.

```
$view['translator']->trans("Symfony2 is great");

$view['translator']->trans('Symfony2 is great');
```

Таны програмын **default_locale** нь **fr** бөгөөд fallback locale-р **en** гэж тохируулсан гэж төсөөлөе. Мөн **AcmeDemoBundle** bundle дотор **fr** locale –ийн орчуулгыг тохируулж өгсөн байг.

```
<!--src/Acme/AcmeDemoBundle/Resources/translations/messages.fr.xliff-->
<?xml version="1.0"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
    <file source-language="en" datatype="plaintext" original="file.ext">
        <body>
            <trans-unit id="1">
                <source>Symfony2 is great</source>
```

```
                <target>J'aime Symfony2</target>
            </trans-unit>
        </body>
    </file>
</xliff>
```

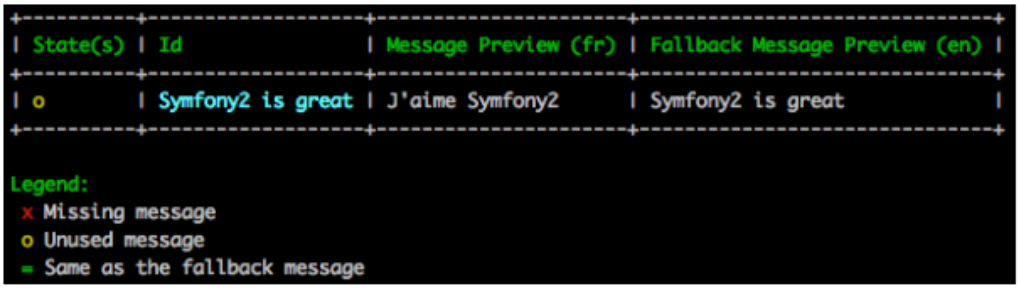
**en** locale нь

```
<!--src/Acme/AcmeDemoBundle/Resources/translations/messages.en.xliff-->
<?xml version="1.0"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
    <file source-language="en" datatype="plaintext" original="file.ext">
        <body>
            <trans-unit id="1">
                <source>Symfony2 is great</source>
                <target>Symfony2 is great</target>
            </trans-unit>
        </body>
    </file>
</xliff>
```

Тэгвэл AcmeDemoBundle –ийн **fr** locale-ийн бүх мессежийг шалгахдаа дараах командыг ажиллуулна.

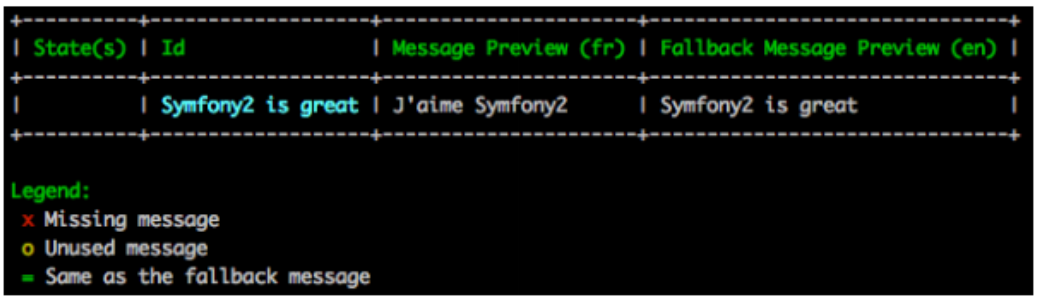
```
$ php app/console translation:debug fr AcmeDemoBundle
```

Энэ командыг ажиллуулсанаар дараах үр дүн гарах болно.



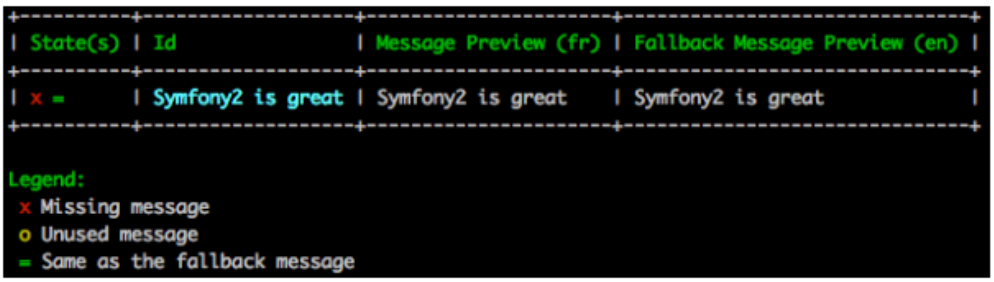

Энд заасан **Symfony2 is great** гэсэн мессеж нь ашиглагдаагүй (unused) гэж гарч байгаагын учир нь хэдийгээр орчуулагдсан боловч хаа нэгтээ ашиглагдагүй байгааг илтгэнэ.

Одоо харин нэг темплэйтэд тухайн мессежийг орчуулж хийвэл дараах үр дүн гарна.



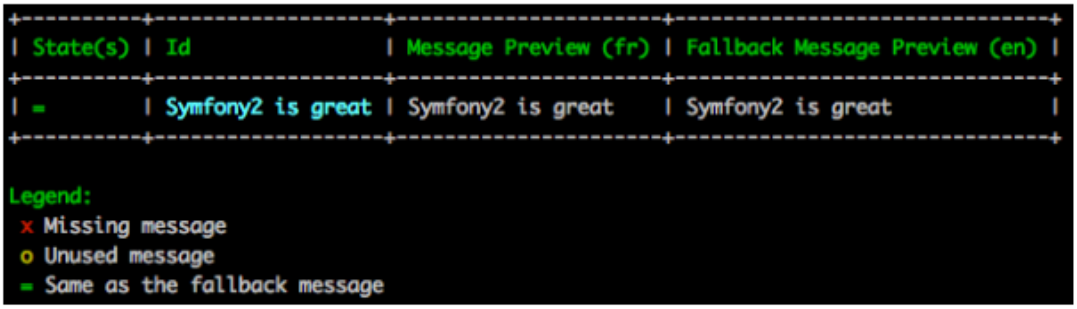

State багана хоосон байна. Учир нь **fr** locale –ийн мессеж орчуулагдсан бөгөөд нэг эсвэл хэд хэдэн темплэйтэд ашиглагдаж байгааг заана.

Хэрэв **fr** locale –ийн орчуулгын файлаас **Symfony2 is great** гэсэн мессежийг устгаад командаа ажиллуулбал дараах үр дүн гарна.




State нь тухайн мессежийг байхгүй гэж заасан байгаа нь **fr** locale –д тухайн орчуулсан мессеж байхгүй хэдий ч темплэйтэд ашигласаар байгааг харуулж байна. Мөн түүнчлэн, **fr** locale –ийн мессеж **en** locale –ийн мессежтэй ижил байна. Энэ нь **en** locale –ийн орчуулгын id нь орчуулагдаагүй мессежийн id-тай адил байгаа учир ингэж зааж байна.

Хэрэв **en** locale –ийн орчуулгын файл дахь агуулгуудыг **fr** locale-ийн орчуулгын файлд хуулаад командаа ажиллуулбал дараах үр дүнг харуулна.




Энэ тохиолдолд **fr** and **en** locale-ийн мессежүүд адил байх бөгөөд магадгүй та орчуулахаа мартсан байж болзошгүй юм гэсэн үр дүнг харуулж байгаа хэрэг юм.

Default-аар бүх domain-уудыг шалгадаг боловч аль нэг domain-ийг сонгон ажиллуулж бас болно.

```
$ php app/console translation:debug en AcmeDemoBundle --domain=messages
```

Таны bundle дотор маш их хэмжээний мессеж байгаа тохиолдолд зөвхөн ашиглагдаагүй болон байхгүй болсон мессежүүдийг харуулах боломжтой.

```
$ php app/console translation:debug en AcmeDemoBundle --only-unused
$ php app/console translation:debug en AcmeDemoBundle --only-missing
```

16-р бүлэг

# Service Container

Орчин үеийн PHP програмууд олон тооны обектуудаас бүрдэнэ. Нэг обект нь мэйл илгээж байхад нөгөө обект нь өгөгдлийн сан руу мэдээлэл хадгалах жишээтэй.

Энэ бүлэгт таны програмд ашиглагдаж буй обектуудтай ажиллан тэдгээрийг зохион байгуулах үүрэг бүхий Symfony2 –н онцгой нэг PHP обектын тухай үзэх болно. Энэ обектийг **service container** гэж нэрлэнэ. Энэ нь кодын ажиллагааг хурдан болгож, тухайн кодын дахин ашиглах боломжыг дээшлүүлж өгөхөд их тустай. Иймд Symfony2-ийн цөм классууд service container –ийг ашигладаг.

Энэ бүлэг сэдвийг судалсанаар та өөрийн обектуудаа container ашиглан үүсгэж, ямар нэг third-party bundle –оос обект авч түүнийгээ зохион байгуулаж ашиглаж сурах болно.

## Service гэж юу вэ?

Service бол энгийн PHP обект юм. Энэ нь компьютерийн шинжлэх ухаанд тусгай зорилгоор үүсгэсэн обектийн (мэйл илгээх гэх мэт) ерөнхий нэр юм. Та сервесийг ашиглахын тулд тусгай үйлдэл гүйцэтгэх шаардлагагүй. Зүгээр л тусгай үйлдлийг биелүүлэх энгийн PHP класс үүсгэнэ.

## Service Container гэж юу вэ?

Service Container бол сервесүүдийг зохион байгуулах энгийн PHP обект юм.

Жишээ нь: мэйл хүлээж авах зорилготой энгийн PHP обект байлаа гэж төсөөлөө. Service Container ашиглахгүйгээр хэрэгтэй обектоо гар аргаар үүсгэх шаардлагатай.

```
1  use Acme\HelloBundle\Mailer;
2
3  $mailer = new Mailer('sendmail');
4  $mailer->send('ryan@foobar.net', ...);
```

**Mailer** класст мэйл илгээхэд ашиглагдах method-ийг үүсгэж өгсөн байг. Гэвч хэрэв та өөр нэг газар мэйл илгээх шаардлагатай бол яах вэ? Мэдээж та **Mailer** обектийг дахин давтан үүсгэхийг хүсэхгүй байх.

## Container-т сервесийг үүсгэн тохируулах

Дээрх асуудлыг шийдвэрлэхийн тулд service container ашиглан **Mailer** обектийг үүсгэх хэрэгтэй. Ингэхийн тулд эхлээд тохиргооны файлд тохиргоо хийх хэрэгтэй.

```yaml
# app/config/config.yml
services:
    my_mailer:
        class: Acme\HelloBundle\Mailer
        arguments: [sendmail]
```

Жишээ болгон хийж байгаа **Acme\HelloBundle\Mailer** обект маань одоо service container ашиглан ажиллах боломжтой боллоо. Controller дотор **get()** method ашиглан container доторх сервесүүд рүү хаанаас ч хандах боломжтой юм.

```php
class HelloController extends Controller
{
    // ...

    public function sendEmailAction()
    {
    // ...
        $mailer = $this->get('my_mailer');
        $mailer->send('ryan@foobar.net', ...);
    }
}
```

container –с **my_mailer** service-ийг дуудах үед container нь тухайн обектийг үүсгэн буцаана. service container-ийн өөр нэг чухал давуу тал бий. Тэр юу вэ гэвэл, сервес нь хэрэгтэй болсон үедээ л үүсдэг. Хэрэв та сервесийг тодорхойлсон түүнийг ашиглаагүй л бол тэр обект үүсэхгүй гэсэн үг юм. Энэ нь санах ойн нөөцийг хэмнэж, програмын ажиллах хурдыг нэмэгдүүлнэ.

Иймд **Mailer** service обектийг нэг удаа үүсгээд түүнийг олон газар ашиглаж болно.

## Service параметерүүд

Сервест параметер ашиглах нь түүнийг илүү уян хатан, зохион байгуулалт сайтай болгоно.

```yaml
# app/config/config.yml
parameters:
    my_mailer.class: Acme\HelloBundle\Mailer
    my_mailer.transport: sendmail

services:
    my_mailer:
        class: "%my_mailer.class%"
        arguments: ["%my_mailer.transport%"]
```

Үр дүн өмнөхтэй яг адил боловч сервесийг хэрхэн тодорхойлохоороо л ялгаатай. **my_mailer.class** болон **my_mailer.transport** –уудийг (%) тэмдэгээр хүрээлж өгсөн нь тухайн container эдгээр нэрээр

параметерийг авна гэдэгийг заа байгаа юм. Container-ийг бүтээж байх үед энэ нь параметер бүрийн утгыг хайх бөгөөд түүнийгээ сервесийг тодорхойлход ашиглана.

Хэрэв та параметерийн утгыг @ тэмдэгээр эхлүүлэх хэрэгтэй бол давхар @ тэмдэг ашиглах хэрэгтэй. Гэхдээ энэ нь зөвхөн YAML файлд боломжтой.

```
# app/config/parameters.yml
parameters:
    # This will be parsed as string "@securepass"
    mailer_password: "@@securepass"
```

Параметерийн зорилго бол сервесийг мэдээллээр хангах. Мэдээж параметер ашиглахгүйгээр сервесийг тодорхойлох нь буруу зүйл биш. Гэхдээ параметер ашигласнаар хэд хэдэн давуу талыг бий болгоно.

- parameter гэсэн нэг түлхүүр үгээр бүх сервесийн тусгаар байдал болон зохион байгуулалтыг хангана.
- Параметерийн утгууд нь олон сервес тодорхойлоход ашиглагдаж болно.
- Bundle –д сервес үүсгэхэд параметерүүд нь тухайн сервесийг ашиглан програмын зохион байгуулалтыг сайжруулах боломжыг олгоно.

Параметер ашиглах эсэх нь таны сонголт. Өндөр чанарын third-party bundle-ууд нь container дахь сервесүүдэд үргэлж параметер ашигласан байдаг.

## Container-ийн бусад тохиргоог ашиглах

**service container**-ийг нэг л тохиргооны файл ашиглан байгуулна. (default –аар **app/config/config.yml**). Сервесийн бусад бүх тохиргоонуудыг энэ файл руу оруулж ирэн ажиллах хэрэгтэй. Энэ нь таны програмыг маш уян хатан болгоно.

### Imports түлхүүр үг ашиглах

Өмнө нь **my_mailer** service container –ийн тодорхойлолтыг програмын тохиргооны файлд шууд хийж өгсөн. (app/config/config.yml). **Mailer** класс өөрөө **AcmeHelloBundle** дотор байгаа. Одоо my_mailer container-ийн тодорхойлолтыг bundle дотор хийж өгнө.

Эхлээд, **AcmeHelloBundle** дотор countainer-ийг тодорхойлох шинэ файл үүсгэн түүнд **my_mailer** container –ийн тодорхойлотыг хуулна. Хэрэв **Resources** эсвэл **Resources/config** хавтас үүсээгүй байвал эхлээд үүсгэх хэрэгтэй.

```
# src/Acme/HelloBundle/Resources/config/services.yml
parameters:
    my_mailer.class: Acme\HelloBundle\Mailer
    my_mailer.transport: sendmail

services:
    my_mailer:
        class: "%my_mailer.class%"
        arguments: ["%my_mailer.transport%"]
```

Одоо харин **imports** түлхүүр үг ашиглан програмын тохиргоонд дээрх шинэ файлыг оруулж ирэх хэрэгтэй.

```
# app/config/config.yml
```

```
2  imports:
3      - { resource: "@AcmeHelloBundle/Resources/config/services.yml" }
```

**Imports** нь өөр газраас (ихэнх тохиолдолд bundle-аас) service container-ийн тохиргоог агуулж буй файлыг программд ашиглах боломжыг олгоно. **resource** нь орж ирж байгаа файлын абсолют замыг авна. **@AcmeHello** гэдэг нь **AcmeHelloBundle** –ийн зам гэдэгийг заана.

## Referencing (Injecting) Services

Container-т нэг эсвэл хэд хэдэн сервесүүдээс хамааралтай сервес үүсгэх тохиолдолд container-ийн жинхэнэ хүчин чадал харагдана.

Жишээ нь: **NewsletterManager** нэртэй шинэ сервес үүсгэе. Энэ нь олон хаяг руу мэйл илгээх, зохион байгуулахад туслах үүрэгтэй. Тэгэхээр **NewsletterManager** дотор яг мессеж илгээх үүрэгтэй **my_mailer** –ээ ашиглая.

```
1  // src/Acme/HelloBundle/Newsletter/NewsletterManager.php
2  namespace Acme\HelloBundle\Newsletter;
3
4  use Acme\HelloBundle\Mailer;
5
6  class NewsletterManager
7  {
8  protected $mailer;
9      public function __construct(Mailer $mailer)
10     {
11         $this->mailer = $mailer;
12     }
13     // ...
14 }
```

service container ашиглахгүйгээр controller ашиглан шинээр **NewsletterManager** үүсгэх боломжтой.

```
1  use Acme\HelloBundle\Newsletter\NewsletterManager;
2
3  // ...
4
5  public function sendNewsletterAction()
6  {
7      $mailer = $this->get('my_mailer');
8      $newsletter = new NewsletterManager($mailer);
9      // ...
10 }
```

Энэ арга нь их амарч гэсэн дараа NewsletterManager классын 2, 3 дахь constructor аргументүүдийг ашиглах шаардлагатай бол яах вэ? Мөн кодоо шинэчлэх, классын нэрийг өөрчилөх хэрэг гарвал яах вэ? зэрэг асуудалууд гарч ирнэ. Энэ 2 тохиолдолд NewsletterManager-ийг хаа байгаа газар нь очиж засвар хийх хэрэгтэй болно. Мэдээж, service container нь дараах аргаар асуудлыг шийдэх боломж олгоно.

```
1  # src/Acme/HelloBundle/Resources/config/services.yml
2  parameters:
3      # ...
4      newsletter_manager.class: Acme\HelloBundle\Newsletter\NewsletterManager
5  services:
6      my_mailer:
```

```
7           # ...
8       newsletter_manager:
9           class: "%newsletter_manager.class%"
10          arguments: ["@my_mailer"]
```

YAML файлд **@my_mailer** нь **my_mailer** нэртэй сервесийг хайж олоод тухайн олсон обектоо бөгөөд **NewsletterManager** –ийн constructor –руу дамжуулна гэдэгийг container –т зааж өгч байгаа юм. Энэ тохиолдолд **my_mailer** сервес өмнө нь үүссэн байх хэрэгтэй. Хэрэв энэ обект байхгүй бол Exception заана.

## Core Symfony болон Third-Party Bundle сервесүүд

Symfony2 болон third-party bundle –ийн сервесүүдийг container ашиглан тохируулж, ажиллана. Тэдгээр сервесүүд рүү хандах, эсвэл өөрийн сервесд ашиглах боломжтой. Symfony2 –т default-аар controller –ийг сервес маягаар заавал тодорхойлох шаардлагагүй байдаг. Үүнээс гадна Symfony2 нь controller-руу service container-ийг бүхэлд нь оруулж ирдэг. Жишээ нь, хэрэглэгчийн session –д мэдээлэл хадгалах бол Symfony2-т controller-с хандах боломжтой **session** сервес байдаг.

```
1  public function indexAction($bar)
2  {
3      $session = $this->get('session');
4      $session->set('foo', $bar);
5      // ...
6  }
```

Symfony2-т темплэйтийг боловсруулах (**templating**), мэйл илгээх (**mailer**), request дээрх мэдээлэл рүү хандах (**request**) гэх мэт third-party bundle эсвэл Symfony core-р хангагдах сервесүүд байнга ашиглагддаг.

Дараагийн алхамд программдаа үүсгэсэн сервесүүд дотор эдгээр сервесийг ашиглах болно. Symfony2 **mailer** service-ийг ашиглахдаа **NewsletterManager** –ийг өөрчилнө. Мөн темплэйтэд мэйлийн агуулгыг боловсруулан гаргахдаа **NewsletterManager** –рүү templating engine service –ийг дамжуулна.

```
1  namespace Acme\HelloBundle\Newsletter;
2
3  use Symfony\Component\Templating\EngineInterface;
4
5  class NewsletterManager
6  {
7      protected $mailer;
8      protected $templating;
9      public function __construct(
10         \Swift_Mailer $mailer,
11         EngineInterface $templating
12     ) {
13         $this->mailer = $mailer;
14         $this->templating = $templating;
15     }
16     // ...
17 }
```

service container-ийг тохируулах амархан.

```
services:
    newsletter_manager:
        class: "%newsletter_manager.class%"
        arguments: ["@mailer", "@templating"]
```

**newsletter_manager** service нь одоо core **mailer** болон **templating** сервес рүү хандах боломжтой боллоо.

## Сервесүүдийн алдааг шалгах

Та console ашиглан container-т бүртгүүлсэн сервесүүдийг харах боломжтой. Бүх сервес болон сервес бүрийн классыг харахдаа дараах командыг ажиллуулна.

```
$ php app/console container:debug
```

Default-аар зөвхөн public сервесүүд харагдах боловч private сервесийг харж бас болно.

```
$ php app/console container:debug --show-private
```

Мөн та нэр зааж тухайн нэг сервесийн дэлгэрэнгүй мэдээллийг авч болно.

```
$ php app/console container:debug my_mailer
```

17-р бүлэг

# Сайжруулалт

Symfony2 хурдан ажилладаг. Гэхдээ хурдыг бүр илүү болгох олон арга байдаг. Энэ бүлэгт Symfony2-ийг илүү хурдан болгох ерөнхий аргуудын талаар судлах болно.

## Byte code cache ашиглах

Програмынхаа ажиллагааг сайжруулах шилдэг аргуудын нэг бол "byte code cache" ашиглах явдал юм. Byte code cache нь PHP эх кодыг дахин дахин хөрвүүлэхээс зайлсхийх зорилготой. Нээлттэй эхийн олон тооны **byte code caches**[48] -үүд байдаг. Хамгийн түгээмэл ашиглагддаг byte code cache бол **APC**[49] юм.

byte code cache бол үнэхээр сул тал байхгүй бөгөөд Symfony2 нь ийм үйлдлийг үнэхээр сайн гүйцэтгэхээр зохион байгуулагдсан юм.

### Нэмэлт сайжруулалт

Byte code cache нь голдуу эх файлууд өөрчлөгдөхөд хяналт тавьж ажиллана. Хэрэв эх файлд өөрчилөлт орвол byte code автоматаар дахин хөрвүүлэгдэнэ гэсэн үг юм. Энэ нь их тохиромжтой арга ч гэсэн мэдээж их ачаалал өгнө.

Ийм шалтгаанаар зарим төрлийн byte code cache-үүдийн хяналтыг хаах боломжтой байдаг. Ингэж хааснаар ямар нэг файлд өөрчлөлт орход тухайн кэйш цэвэрлэгдсэн байх болно.

Жишээ нь, APC-д хяналтыг хаах бол **php.ini** тохиргооны файлд **apc.stat=0** гэсэн тохиргоог нэмнэ.

[48] http://en.wikipedia.org/wiki/List_of_PHP_accelerators
[49] http://php.net/manual/en/book.apc.php

## Composer's Class Map ашиглах

Default-аар Symfony2 standard edition нь **autoload.php**[50] файл дахь Composer's autoloader-ийг ашигладаг. Энэ autoloader нь бүртгэгдсэн хавтсанд байрлах ямар нэг шинээр үүсгэсэн классыг автоматаар хайна.

Гэвч энэ нь тодорхой файлыг олохын тулд бүртгэгдсэн бүх namespace-үүдийг дахин дахин шалгаж файлаа хайж дуусах хүртэл file_exists-ийг дуудан ажиллуулсаар байдаг учир их төвөгтэй байдаг.

Үүнийг шийдэхийн тулд Composer-ийг үүсгэхдээ "class map"-ийг түүнд зааж өгнө. Үүнийг дараах командаар биелүүлнэ.

```
$ php composer.phar dump-autoload --optimize
```

## APC-тэй хамт autoloader ашиглан кэйшлэх

Өөр нэг шийдэл бол класс бүрийн байрлалыг хадгалж авах юм. Symfony нь энэ үйлдлийг гүйцэтгэх үүрэгтэй ApcClassLoader[51] классыг өөртөө агуулж байдаг. Үүнийг ашиглахын тулд fornt controller файлд тохиргоо хийх хэрэгтэй. Хэрэв та Symfony-ийн стандарт тархацыг ашиглаж байгаа бол энэ тохиргоог тайлбар маягаар бичсэн байдаг.

```
// app.php
// ...

$loader = require_once __DIR__.'/../app/bootstrap.php.cache';

// Use APC for autoloading to improve performance
// Change 'sf2' by the prefix you want in order
// to prevent key conflict with another application
/*
$loader = new ApcClassLoader('sf2', $loader);
$loader->register(true);
*/

// ...
```

## Bootstrap файл ашиглах

Кодын үян хатан байдал болон кодын дахин ашиглалтыг сайжруулахын тулд Symfony2 програм нь олон төрлийн класс болон 3rd Party компонентүүдтай ажиллана. Гэхдээ эдгээр бүх классыг дуудах явцад багагүй ачаалал өгдөг. Энэ ачааллыг багасгахын тулд Symfony2 Standard Edition нь **bootstrap file**[52] гэж нэрлэгдэх Script-ийг агуулах бөгөөд энэ файл нь олон тооны классын тодорхойлолтуудаас бүрдэнэ. Энэ файлыг дуудан ашигласнаар Symfony нь эдгээр классуудыг агуулж байгаа эх файлуудыг заавал оруулж ирэх шаардлагагүй болно.

[50] https://github.com/symfony/symfony-standard/blob/master/app/autoload.php
[51] http://api.symfony.com/master/Symfony/Component/ClassLoader/ApcClassLoader.html
[52] https://github.com/sensio/SensioDistributionBundle/blob/master/Composer/ScriptHandler.php

Symfony2 Standard Edition ашиглаж байгаа бол bootstrap file –ийг ашиглаж байгаа гэсэн үг. Үүнийг шалгахыг хүсвэл front controller (ихэнхдээ **app.php**) файлаа нээгээд дараах мөр кодыг хараарай.

```
1  require_once __DIR__.'/../app/bootstrap.php.cache';
```

Гэхдээ **bootstrap file** нь дараах 2 сул талтай байдаг.

- Эх файлууд өөрчилөгдөхөд тухайн файлыг дахин үүсгэх хэрэгтэй. (жнь: Symfony2 source эсвэл vendor сангуудад өөрчлөлт хийхэд);
- Debug хийхдээ bootstrap file-д эхлэхийн цэгийг тавих хэрэгтэй.

Symfony2 Standard Edition-ийг ашиглаж байгаа үед **php composer.phar install** командаар vendor сангуудад өөрчилөлт хийсний дараа bootstrap file нь автоматаар дахин үүсдэг.

18-р бүлэг

# Дотоод бүтэц

Энэ бүлэгт Symfony2 хэрхэн ажилладаг болох ба түүнийг хэрхэн өргөтгөх боломжтой талаар үзэх болно.

## Ерөнхий бүтэц

Symfony2 код нь хэд хэдэн тусдаа давхаргаас бүрдэнэ. Давхарга бүр нь өмнөх дээрээх байрлана.

### HttpFoundation компонент

Хамгийн цаад талын давхарга бол HttpFoundation компонент юм. HttpFoundation нь HTTP-тэй харилцахад шаардлагатай гол обектийг агуулж байдага. Энэ нь обект хандалдат PHP функц болон хувьсагчууд юм.

- **Request** класс нь **$_GET, $_POST, $_COOKIE, $_FILES,** болон **$_SERVER** гэсэн глобал хувьсагчуудыг агуулна.
- **Response** класс нь **header(), setcookie(), echo** зэрэг PHP функцүүдийг агуулна.
- **Session** класс болон **SessionStorageInterface** интерфэйс нь session-ийг удирдах **session_*()** функцийг агуулна.

### HttpKernel компонент

**HttpFoundation**-ийн дээр **HttpKernel** компонент байрлана. HttpKernel нь HTTP протоколын динамик хэсгийг боловсруулна. Энэ нь  Request болон Response классуудын дээр байрлан стандарт аргаар request-ийг боловсруулна.

### FrameworkBundle

**FrameworkBundle** гэдэг нь bundle бөгөөд MVC framework-ийг илүү хөнгөхөн, хурдан болгох зорилготой сангуудыг үндсэн компонентүүдтай холбоно.

# Kernel

**HttpKernel** класс нь Symfony2 –ийн гол класс бөгөөд клейнтийн илгээж байгаа request-ийг боловсруулна. Үүний үндсэн үүрэг бол **Request** обектоос **Response** обектийг хөрвүүлэх юм.

Symfony2 Kernel нь **HttpKernelInterface** –ээс implement хийгдэнэ.

```
1  function handle(Request $request, $type = self::MASTER_REQUEST, $catch = true)
```

### Controller

Request-ээс Response руу хөрвүүлэхдээ Kernel нь Controller дээр тулгуурлан ажиллана.

```
1  public function getController(Request $request);
2
3  public function getArguments(Request $request, $controller);
```

**getController()** method нь өгсөн Request-тэй хамааралтай Controller-ийг буцаана. **getArguments()** method нь controller руу дамжиж байгаа аргументийн массивыг буцаана.

### Request-ийг боловсруулах

**handle()** method нь Request-ийг авч Response болгон буцаана. Request-ийг хөрвүүлэхдээ, handle() method нь Event notification болон Resolver дээр тулгуурлан ажиллана.

1. Ямар нэг үйлдэл гүйцэтгэхийн өмнө **Kernel.request** event нь зарлагдсан байна. Хэрэв listener-үүдийн нэг нь Response буцааж байвал шууд 8 –р алхам руу үсрэнэ.
2. Resolver нь тухайн үед ажиллах ёстой Controller-ийг тодорхойлно.
3. **kernel.controller** event-ийн listener нь тохирох controller-ийг гаргаж ирнэ.
4. Kernel нь тухайн controller-ийг зөв PHP функц байгааг шалгана.
5. Resolver нь controller руу дамжуулах аргументүүдийг тодорхойлно.
6. Kernel нь тухайн controller-ийг дуудна.
7. Хэрэв controller нь Response буцаахгүй байвал kernel.view event-ийн listener-үүд Controller –ийн Response руу буцаах утгыг хөрвүүлэх боломжтой.
8. **kernel.response** event –ийн listener-үүд нь Response –ийг үүсгэнэ.
9. Response-ийг буцаана.
10. **kernel.terminate** event –ийн listener нь Response биелэгдсэний дараах үйлдлийг гүйцэтгэнэ.

Хэрэв request-ийг боловсруулж байх явцад алдаа гарвал **kernel.exception** зарлагдаж, listener-үүд нь тухайн Exception-ийг Response болгон хөрвүүлнэ.

Хэрэв Exception барихыг хүсэхгүй бол **kernel.exception** event –ийг идэвхгүй болгохын тулд **handle()** method –ийн 3 дахь аргументэд **false** утга дамжуулна.

### Internal Request

Request боловсруулж байх явцад sub-request –ийг боловсруулах боломжтой байдаг. Ингэхийн тулд **handle()** method руу тухайн request-ээ дамжуулна. (энэ нь 2 дахь аргумент байна).

- HttpKernelInterface::MASTER_REQUEST;

- HttpKernelInterface::SUB_REQUEST

### Event

Event бол KernelEvent –ийн дэд класс юм. Энэ нь event бүр үндсэн мэдээлэл рүү хандах боломжтой гэсэн үг юм.

- getRequestType() – request-ийн төрлийг буцаана. (**HttpKernelInterface::MASTER_REQUEST** эсвэл **HttpKernelInterface::SUB_REQUEST**)
- getKernel() – Kernel request-ийг боловсруулна.
- getRequest() – Боловсруулсан request-ийг буцаана.

### getRequestType()

**getRequestType()** method нь listener-үүдэд request-ийн төрлийг заана. Жишээ нь: хэрэв listener нь зөвхөн master request-ийг идэвхтэй байлгах ёстой бол listener method-ийн эхэнд дараах кодыг бичиж өгнө.

```
1  use Symfony\Component\HttpKernel\HttpKernelInterface;
2
3  if (HttpKernelInterface::MASTER_REQUEST !== $event->getRequestType()) {
4      // return immediately
5      return;
6  }
```

### `kernel.request` Event

Энэ event-ийн гол зарчим бол **Response** обектийг шууд буцаах эсвэл тухайн event –ийн дараа дуудагдаж болох controller –ийн хувьсагчийг тохируулна. Аливаа listener event дээрх setResponse() method ашиглан Response обектийг буцаадаг. Энэ тохиолдолд бусад бүх listener дуудагдахгүй.

Энэ event нь FrameworkBundle –аар ашиглагдах ба RouterListener –ийг ашиглан _controller Request аттрибутыг суулгана. RequestListener нь RouterInterface обектийг ашиглан Controller-ийн нэрийг тодорхойлж, Request-ийг тохируулна.

### `kernel.controller` Event

Энэ event нь FrameworkBundle-аар ашиглагдахгүй боловч тухайн биелэгдэх ёстой controller-ийг өөрчилөхөд ашиглагдана.

```
1  use Symfony\Component\HttpKernel\Event\FilterControllerEvent;
2
3  public function onKernelController(FilterControllerEvent $event)
4  {
5      $controller = $event->getController();
6      // ...
7
8      // the controller can be changed to any PHP callable
9      $event->setController($controller);
10 }
```

### kernel.view Event

Энэ event нь мөн FrameworkBundle-аар ашиглагдахгүй ч sub-system-ийг хархад ашиглагдах болно. Энэ event-ийг зөвхөн Controller нь Response обект буцаахгүй тохиолдолд л дуудагдана. Зорилго Response руу хөрвүүлэгдсэн бусад утгыг буцаах юм.

Controller-оос буцааж байгаа утга руу getControllerResult method ашиглан хандах боломжтой.

```
use Symfony\Component\HttpKernel\Event\GetResponseForControllerResultEvent;
use Symfony\Component\HttpFoundation\Response;

public function onKernelView(GetResponseForControllerResultEvent $event)
{
    $val = $event->getControllerResult();
    $response = new Response();

    // ... some how customize the Response from the return value

    $event->setResponse($response);
}
```

### kernel.response Event

Энэ event-ийн зорилго бол Response обектийг үүсгэсний дараа бусад системүүд түүнийг засах, өөрчилөх боломж олгоно.

```
public function onKernelResponse(FilterResponseEvent $event)
{
    $response = $event->getResponse();
    // ... modify the response object
}
```

### kernel.finish_request Event

Энэ event нь тухайн request-ийг боловсруулсаны дараа биелүүлэх үйлдлийг гүйцэтгэнэ.

### kernel.terminate Event

Энэ event-ийн зорилго бол клейнт рүү Response-ийг буцаасны дараа үйлдэл гүйцэтгэдэг.

### kernel.exception Event

FrameworkBundle нь өгсөн Controller руу request-ийг шилжүүлэх ExceptionListener-ийг бүртгэсэн байдаг. Энэ event дээрх listener нь Response обектийг болон Exception обектийг үүсгэн тохируулана.

```
use Symfony\Component\HttpKernel\Event\GetResponseForExceptionEvent;
use Symfony\Component\HttpFoundation\Response;

public function onKernelException(GetResponseForExceptionEvent $event)
{
    $exception = $event->getException();
    $response = new Response();
```

```
8       // setup the Response object based on the caught exception
9       $event->setResponse($response);
10
11      // you can alternatively set a new Exception
12      // $exception = new \Exception('Some special exception');
13      // $event->setException($exception);
14  }
```

## Profiler

Энэ нь идэвхтэй байгаа үед Symfony2-ийн profiler нь программд хийгдэж буй request бүрийн тухай мэдээллийг цуглуулдаг ба дараа шинжилгээ хийхийн тулд тэдгээр мэдээллийг хадгална. Profiler нь development орчинд таны кодыг debug хийх ба ажиллагааг сайжруулхад ашиглагдана. Production орчинд асуудлыг судлахад ашиглагдана.

Хэрэв та Symfony2 Standard Edition ашиглаж байгаа бол profiler, web debug toolbar, web profiler зэрэг хэрэгсэлүүдийг тохируулсан байдаг.

### Web Debug Toolbar ашиглах

Development орчинд web debug toolbar нь бүх хуудасны доод талд байрлана. Энэ нь ямар нэгэн зүйл ажиллахгүй байвал тэр тухай хэрэгцээт мэдээллийг харуулна.

Хэрэв web debug toolbar-ийн гаргасан дүгнэлт хангалтгүй байвал token link дээр дарж Web Profiler руу хандана.

### Web Profiler ашиглан profiling data-д анализ хийх

Web Profiler нь development орчинд таны кодыг debug хийх ба ажиллагааг сайжруулхад ашиглагддаг хэрэгсэл юм. Харин энэ нь production орчинд тохиолдох асуудлыг хайж олход ашиглагдана. Энэ нь profiler-ийн цуглуулсан бүх мэдээллийг харуулна.

### Profiling мэдээлэлд хандах

Request-ийн тухай мэдээллийг хадгалж байгаа profiler нь token-тай холбоотой байна. Token нь Response-ийн X-Debug-Token HTTP header –т байрлана.

```
1  $profile = $container->get('profiler')->loadProfileFromResponse($response);
2
3  $profile = $container->get('profiler')->loadProfile($token);
```

Хэд хэдэн шалгуур мэдээлэл дээр үндэслээд token руу хандахдаа **find()** method-ийг ашиглана.

```
1  // get the latest 10 tokens
2  $tokens = $container->get('profiler')->find('', '', 10, '', '');
3  // get the latest 10 tokens for all URL containing /admin/
4  $tokens = $container->get('profiler')->find('', '/admin/', 10, '', '');
5  // get the latest 10 tokens for local requests
```

```
$tokens = $container->get('profiler')->find('127.0.0.1', '', 10, '', '');
// get the latest 10 tokens for requests that happened between 2 and 4 days ago
$tokens = $container->get('profiler')->find('', '', 10, '4 days ago', '2 days ago');
```

Хэрэв та өөр өөр орчинд profiling data –г авах бол export() болон import() method-уудыг ашиглана.

```
// on the production machine
$profile = $container->get('profiler')->loadProfile($token);
$data = $profiler->export($profile);
// on the development machine
$profiler->import($data);
```