Getting started

1.1 Create a new application

1.1.1 Development environment

During all this course, I am going to work on an Ubuntu server 14.04 installed on a virtual machine. You are free to use any other operating system, you may need to adjust the system commands given here to your environment. If you use any Debian based Linux distribution you should not have any problems executing all the commands mentioned in this book.

If you are not familiar with installing and configuring packages on a linux operating system, you can use a machine pre-configured with the tools we need for running the applications we will develop here. You can download the VirtualBox machine from http://ilyeskooli.com/ubuntu-symfony.tar.gz

The virtual machine has the following hostnames mappings in /etc/hosts

```
127.0.0.1       localhost
127.0.1.1       ubuntu
127.0.0.1       symfony.local
127.0.0.1       store.local
```

Configured interfaces in /etc/network/interfaces

```
auto eth1
iface eth1 inet dhcp

auto eth0
iface eth0 inet static
address 10.10.10.1
netmask 255.255.255.0
```

In VirtualBox it has one bridged network interface with internet access and one host only interface with the IP 10.10.10.100

On my development machine I have the following mappings in my /etc/hosts

```
10.10.10.1      symfony.local
10.10.10.1      store.local
10.10.10.1      fr.store.local
10.10.10.1      us.store.local
```

The following apache virtual hosts are defined

```
/etc/apache2/sites-available/symfony.local.conf
<VirtualHost *:80>
    ServerName symfony.local

    DocumentRoot /home/ubuntu/web/symfony-tutorial/web
    <Directory /home/ubuntu/web/symfony-tutorial/web>
        Options Indexes FollowSymLinks MultiViews
    AllowOverride all
        allow from all
        Require all granted
        <IfModule mod_rewrite.c>
            RewriteEngine On
            RewriteCond %{REQUEST_FILENAME} !-f [OR]
            RewriteCond %{REQUEST_FILENAME} !=^app_dev.php
            RewriteRule ^(.*)$ /app.php [L]
        </IfModule>
    </Directory>
    ErrorLog /var/log/apache2/symfony_error.log
    CustomLog /var/log/apache2/symfony_access.log combined
</VirtualHost>
```

```
/etc/apache2/sites-available/store.local.conf
<VirtualHost *:80>
    ServerName store.local
    ServerAlias *.store.local

    DocumentRoot /home/ubuntu/web/store/web
    <Directory /home/ubuntu/web/store/web>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride all
        allow from all
        Require all granted
        <IfModule mod_rewrite.c>
            RewriteEngine On
            RewriteCond %{REQUEST_FILENAME} !-f [OR]
            RewriteCond %{REQUEST_FILENAME} !=^app_dev.php
            RewriteRule ^(.*)$ /app.php [L]
        </IfModule>
    </Directory>

    ErrorLog /var/log/apache2/store_error.log
    CustomLog /var/log/apache2/store_access.log combined
</VirtualHost>
```

1.1.2 Install Composer

Composer is a dependency management tool for PHP applications. To install Composer, simply run:

```
$ curl -sS https://getcomposer.org/installer|sudo php -- --install-dir=/bin \
 --filename=composer
#!/usr/bin/env php
All settings correct for using Composer
Downloading...

Composer successfully installed to: /bin/composer
Use it: php /bin/composer
```

You can check the composer version by typing

```
$ composer --version
Composer version 1.0-dev
(4eb8ecff9cd136c4c2829164db1a55ad9d2de616) 2015-05-10 19:56:28
```

For more details about other installation alternatives, check Composer's online documenation.

1.1.3 Create a new Symfony application

```
$ composer create-project symfony/framework-standard-edition symfony-tutorial "2.6.7"
Installing symfony/framework-standard-edition (v2.6.7)
  - Installing symfony/framework-standard-edition (v2.6.7)
    Downloading: 100%

Created project in symfony-tutorial
> SymfonyStandard\Composer::hookRootPackageInstall
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
  - Installing doctrine/lexer (v1.0.1)
  .
  .. All the required packages
  ...
  .....
Generating autoload files
Would you like to install Acme demo bundle? [y/N] N
> Incenteev\ParameterHandler\ScriptHandler::buildParameters
Creating the "app/config/parameters.yml" file
Some parameters are missing. Please provide them.
database_driver (pdo_mysql):
database_host (127.0.0.1):
database_port (null):
database_name (symfony):
database_user (root):
database_password (null): mysql
mailer_transport (smtp):
mailer_host (127.0.0.1):
mailer_user (null):
mailer_password (null):
locale (en):
secret (ThisTokenIsNotSoSecretChangeIt):
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::buildBootstrap
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::clearCache
Clearing the cache for the dev environment with debug true
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::installAssets
Trying to install assets as symbolic links.
Installing assets for Symfony\Bundle\FrameworkBundle into web/bundles/framework
The assets were installed using symbolic links.
Installing assets for Sensio\Bundle\DistributionBundle into web/bundles/sensiodistribution
The assets were installed using symbolic links.
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::installRequirementsFile
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::removeSymfonyStandardFiles
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::prepareDeploymentTarget
```

Composer generated the directory structure of your Symfony application.

- **app** contains the application loader and global configuration files.
- **app**/**cache** contains the application's cache.
- **app**/**logs** contains the application's logs.
- **src** contains the source code of the application.
- **vendor** contains code of external packages required by the application
- **web** is the web directory that your web server should use as DocumentRoot

To check if the application is created correctly, visit http://symfony.local you should get a 404 not found HTTP error. We will come back to this later.

1.2 Git

Git is a distributed VCS (Version Control System). If you are not using an VCS in your company, stop reading right now and go to install one. I would recommand Git, a comparison of VCS tools or a detailed documentation about Git are out of the scope of this book. If you are already working with a different VCS (SVN, Mercurial or any other tool) and you are not willing to give Git a try that's perfectly fine, you can skip the current and the next section. If you are not using a VCS yet, this can be a great opportunity to start working with Git.

If you didn't yet, you can install Git from here. For detailed informations on how to install Git on different platforms, check the online documentation.

If you didn't worked with github before, you may need to learn how to set up an SSH key

If this is the first time you will use Git on your development environment, don't forget to setup your identity.

```
$ git config --global user.name "Your name"
$ git config --global user.email "your_email@example.com"
```

You can also setup aliases for Git, which are simply shortcuts to commands you use often. Here is the list of my aliases that I found very useful:

```
$ git config --global alias.co checkout
$ git config --global alias.br branch
$ git config --global alias.ci commit
$ git config --global alias.st status
$ git config --global alias.poh "push origin HEAD"
$ git config --global alias.pom "pull origin master"
$ git config --global alias.dif "diff HEAD..origin/master"
$ git config --global alias.last 'log -1 HEAD'
```

Let's create a Git repository

```
$ git init
Initialized empty Git repository in /home/ubuntu/web/symfony-tutorial/.git/
```

```
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .gitignore
    LICENSE
    README.md
    UPGRADE-2.2.md
    UPGRADE-2.3.md
    UPGRADE-2.4.md
    UPGRADE.md
    app/
    composer.json
    composer.lock
    src/
    web/

nothing added to commit but untracked files present (use "git add" to track)
```

Symfony standard edition comes with a .gitignore file that excludes all autogenerated files from your repository.

To start tracking a file, we will use `git add filename`. Since this is a new project, we can run `git add .`. Now if you run `git status` again you won't see *Untracked files* anymore, instead you will see a list of *Changes to be commited*.

Let's commit the changes.

```
$ git commit -m "Symfony tutorial"
[master (root-commit) 227f5a9] Symfony tutorial
 41 files changed, 3754 insertions(+)
 create mode 100644 .gitignore
 ...
 create mode 100644 web/robots.txt
```

To see the last commits, run `git log`.

```
$ git log
commit 227f5a9febc86dcec2c846287a01384179b98069
Author: skafandri <you@example.com>
Date:   Sun Aug 23 00:53:09 2015 +0300

    Symfony tutorial
```

Git generates a *unique* hash for each commit (10343840c09043b26df51ef9806091a8dd0c646b), so most probably you will see a different hash in your repository.

> Git generates an SHA-1 hash computed from your repository metadata, your username and the current timestamp. This data combined together guarantees there is a very small probability of colliding with other commits.

Symfony standard edition comes with a **composer.json** file that holds informations about the repository, components dependencies, among other parameters. The second dependency is `"symfony/symfony": "2.6.*"`. That means to require the latest available Symfony version in the 2.6 series. We just realized that Symfony 2.7 is available, so let's update our application.

Change `"symfony/symfony": "2.6.*"` to `"symfony/symfony": "2.7.*"` and save the file.

```
$ composer update symfony/symfony
Loading composer repositories with package information
Updating dependencies (including require-dev)
  - Removing symfony/symfony (v2.6.11)
  - Installing symfony/symfony (v2.7.3)
    Loading from cache

Writing lock file
Generating autoload files
> Incenteev\ParameterHandler\ScriptHandler::buildParameters
Updating the "app/config/parameters.yml" file
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::buildBootstrap
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::clearCache
Clearing the cache for the dev environment with debug true
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::installAssets
Trying to install assets as symbolic links.
Installing assets for Symfony\Bundle\FrameworkBundle into web/bundles/framework
The assets were installed using symbolic links.
Installing assets for Sensio\Bundle\DistributionBundle into web/bundles/sensiodistribution
The assets were installed using symbolic links.
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::installRequirementsFile
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::removeSymfonyStandardFiles
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::prepareDeploymentTarget
```

Let's commit our update.

```
$ git add composer.json
$ git commit -m "Updated Symfony version to 2.7.*"
[master 596064a] Updated Symfony version to 2.7.*
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Let's check our repository status again.

```
 $ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   composer.lock

no changes added to commit (use "git add" and/or "git commit -a")
```

Seems that composer changed another file **composer.lock**. Let's see what changes it did. Using git diff, we can see the changes that are made to a file between 2 commits. Not specifying the commits endpoints will just result in comparing the file from the current status to the HEAD.

> HEAD is a Git shortcut that points to the current commit in your repository.

To see the changes made to **composer.lock** run `git diff composer.lock`. The changes shows the update from Symfony 2.6 to 2.7 plus some more details, notably the **reference** key. After computing packages dependencies, Composer compiles a list of commit hashes to clone for each package. When you run `composer install`, Composer doesn't compute any dependencies and just clones the list of commits from **composer.lock**. For this reason `composer install` runs much faster than `composer update`, and also guarantees you have the same packages versions since the last time you updated your packages.

I recommend to never run `composer update` on a production environment since a new package version could bring a backward compatibility break or new bug (feature?) that may break your code.

Let's commit the change to **composer.lock**

```
$ git commit -a -m 'Updated Symfony version to 2.7.*'
[master bd0a116] Updated Symfony version to 2.7.*
 1 file changed, 24 insertions(+), 19 deletions(-)
```

Let's check our repository history again.

```
 $ git log
commit bd0a1162b85c31591b28a11d03d891fd204a2026
Author: skafandri <you@example.com>
Date:   Sun Aug 23 01:43:14 2015 +0300

    Updated Symfony version to 2.7.*

commit 596064a7098a0ebdab7906a834c5de3178dc8e93
Author: skafandri <you@example.com>
Date:   Sun Aug 23 01:26:03 2015 +0300

    Updated Symfony version to 2.7.*

commit 227f5a9febc86dcec2c846287a01384179b98069
Author: skafandri <you@example.com>
Date:   Sun Aug 23 00:53:09 2015 +0300

    Symfony tutorial
```

We have now 2 commits that are related to the same feature. This breaks the main purpose of a CVS, is hard to return to a point in history or to see the exact changes it made to the application if it is spread across many commits, and to make it even worst, with different commit messages.

So let's merge the last 2 commits into one single commit.

```
 git rebase -i 227f5a9febc86dcec2c846287a01384179b98069
```

Git will open your default text editor with the following

```
pick 596064a Updated Symfony version to 2.7.*
pick bd0a116 Updated Symfony version to 2.7.*

# Rebase 227f5a9..bd0a116 onto 227f5a9
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

Change `pick bd0a116 Updated Symfony version to 2.7.*` to `s bd0a116 Updated Symfony version to 2.7.*`
Save the file and exit, Git will open your text editor again with

```
# This is a combination of 2 commits.
# The first commit's message is:

Updated Symfony version to 2.7.*

# This is the 2nd commit message:

Updated Symfony version to 2.7.*

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# rebase in progress; onto 227f5a9
# You are currently editing a commit while rebasing branch 'master' on '227f5a9'.
#
# Changes to be committed:
#       modified:   composer.json
#       modified:   composer.lock
#
```

Remove one of the duplicate **Updated Symfony version to 2.7.***, save the file and exit. You should see

```
$ git rebase -i 227f5a9febc86dcec2c846287a01384179b98069
[detached HEAD f32f00b] Updated Symfony version to 2.7.*
 2 files changed, 25 insertions(+), 20 deletions(-)
Successfully rebased and updated refs/heads/master.
```

```
$ git log
commit f32f00b2691285fe93946e0c30c53fab47edf012
Author: skafandri <you@example.com>
Date:   Sun Aug 23 01:26:03 2015 +0300

    Updated Symfony version to 2.7.*

commit 227f5a9febc86dcec2c846287a01384179b98069
Author: skafandri <you@example.com>
Date:   Sun Aug 23 00:53:09 2015 +0300

    Symfony tutorial
```

Looks better, to see exactly what happened in a given commit, use `git show f32f00b2691285fe93946e0c30c53fab47edf012`
You will see the changes made to **composer.json** and **composer.lock**

We are in section (1.2 Git) from this book, let's create a branch for each section, so we will start with **1.2**

```
$ git checkout -b 1.2
Switched to a new branch '1.2'
```

> You can find all the source code of this book on github https://github.com/symfony-tutorial/tutorial and every section is in a branch with the section number. To get the code at this point, you need to checkout the branch 1.2

To get the code for the next section

```
git clone git@github.com:symfony-tutorial/tutorial.git --branch 1.3
```

Or, if you already cloned the repository, `git checkout 1.3`. To see all the branches, `git branch -r`.

So far, we worked only on a local repository. To share code between developers, they must synchronize their work with a shared repository. There are free platforms like Github and Gitlab, you can also install git server within your private infrastructure.

I am using Github and the example URLs will point to my github repository. No matter which platform you use, the commands are the same. On my profile, I create a new repository and name it **my-symfony-tutorial**

```
$ git remote add origin git@github.com:skafandri/my-symfony-tutorial.git
```

We just created a new remote that we called **origin**. Now we can **push** our changes to the remote repository.

```
$ git push origin 1.2
Counting objects: 57, done.
Compressing objects: 100% (51/51), done.
Writing objects: 100% (57/57), 45.97 KiB | 0 bytes/s, done.
Total 57 (delta 3), reused 0 (delta 0)
To git@github.com:skafandri/my-symfony-tutorial.git
 * [new branch]      1.2 -> 1.2
```

1.3 Configuration

Let's first take a look routing configuration file is located at app/config/routing.yml

```
app:
    resource: "@AppBundle/Controller/"
    type:     annotation
```

This is a YAML configuration file.
resource indicates to import an external configuration file.
type is the type of configuration to load.

Symfony supports 4 types of configuration files, PHP, YAML, XML, Annotation. They are almost the same except some few features that only XML and YAML formats supports (by now). Like support for class and global constants as parameters.

All the configuration is compiled as serialized PHP objects and saved in cache files in app/cache. This said, the choice of a configuration type has no impact on the application's performance. I don't have any recommendadtion regarding which format to use, I think is a matter of taste. Personally, I prefer YAML because I find it more readable, except for doctrine entities and documents where I prefer annotations. The most important thing with configuration files is not to treat them with less care than you do with source code. You should maintain a logical organization. Otherwise, you may end up with hundreds of files, with similar and different names and things spread all over, or one fat file that your editor struggles to search something into. I don't know which is worse, I don't like either situations.

Seems that the imported resource is a controller class, let's check it out **src/AppBundle/Controller/DefaultController.php**, particularly the **indexAction** method (the only one)

```
/**
* @Route("/app/example", name="homepage")
*/
public function indexAction()
{
  return $this->render('default/index.html.twig');
}
```

The method's annotation defines a route with the name **homepage** and the path **/app/example**. Now we know why we got a 404 error when we tried to browse http://symfony.local, let's try again with http://symfony.local/app/example you should see a white page with just **Homepage.**. In fact, all this action does is render the **default/index.html.twig** template. Let's check it out at **app/Resources/views/default/index.html.twig**

```
{% extends 'base.html.twig' %}

{% block body %}
    Homepage.
{% endblock %}
```

Like the file extension suggests, that's a Twig template. It extends from **app/Resources/views/base.html.twig** and defines a block called body with just Homepage. inside it, just what you saw in your browser.

So this was another way to configure Symfony through annotations. In the case of routes however, I don't like using annotations. If I have one controller with (only) 10 actions 20 lines each, I have to scroll up and down more than 200 lines to see all the routes. Alternatively, if I have a configuration file, XML or YAML, all the routes will fit into my laptop's screen. You must choose the option you feel the most comfortable with, in the following of this course I will configure routes using YAML format.

First, let's remove the annotation from **indexAction**. Then we create a new routing file **src/AppBundle/Resources/config/routing.yml**

```
app:
    path: "/app/example"
    defaults: {_controller: AppBundle:Default:index}
```

We update **app/config/routing.yml**, we change `@AppBundle/Controller/` to `@AppBundle/Resources/config/routing.yml` (the new routing configuration file we just created), and we remove the **type** key.

Visit http://symfony.local/app/example to make sure everything works as before and we didn't break anything. We should not in fact, we just changed the configuration format, the values are the same.

1.4 Tracer Bullet Development

The idea behind TBD is very simple, implement just enough code to have something visible (not necessarly graphically). One of the greatest advantages of TBD is that it forces you not to couple your code with third party vendors like libraries, SAAS, APIs.. Pushing those conserns as far as possible in the development process will help you implement a more independent application. It will be easier for you in the future to switch from one technology to another (database server, web server, mailing server) since most of your code will be platform agnostic.

I won't dive too much into TBD now. We will just adopt it as we develop our application. Better to see it in action. I recommend the following book *Ship It! A Practical Guide to Successful Software Projects [ISBN 0-9745140-4-7]*. The authors dedicated a full chapter to Tracer Bullet Development. The rest of the book is not less valuable, is a helpful guide through good software development practices. With very interesting use cases from real projects.

We will start implementing a simple product catalog for our application. Initially, we will delivery the following features: list categories, edit a category, save a category.

1.4.1 Routing

Routing is about telling Symfony which action to invoke depending on the request properties. The mostly used property being the request URL, is not the only one. You can configure routes based on hostname, HTTP method, headers, custom computed conditions...

Let's create the routes for the category functionality.

Edit **app/config/routing.yml** discard the existing content and put

```
category:
    prefix: /category
    resource: "@AppBundle/Resources/config/routing/category.yml"
```

`prefix: /category` tells the router that every **path** defined in the imported file will be prefixed by /**category**. So the real path of `/some_path` will be `/category/some_path`.

- Create **src/AppBundle/Resources/config/routing/category.yml**

```
category_list:
    path: /list
    defaults: { _controller: AppBundle:Category:listCategories}
category_edit:
    path: /edit/{categoryId}
    defaults: { _controller: AppBundle:Category:editCategory, categoryId:0}
    methods:  [GET]
category_save:
    path: /edit/{categoryId}
    defaults: { _controller: AppBundle:Category:saveCategory, categoryId:0}
    methods:  [POST]
```

`{ _controller: AppBundle:Category:listCategories}` tells the router component about which controller method to call to return a response when this route is matched. In this case it will be `AppBundle\Controller\CategoryController::listCategoriesAction()`

1.4.2 Views

Symfony has a flexible templating component that supports multiple templating engines. In this course we will use Twig.

> **Symfony best practice:** Use Twig templating format for your templates.

- **app/Resources/views/category/list.html.twig**

```
{% extends 'base.html.twig' %}
{% block body %}
    <p>Categories</p>
    <table border="1">
        <tr>
            <th>#</th><th>Category</th><th colspan="2">Actions</th>
        </tr>
        {% for category in categories %}
            {% set editUrl = path('category_edit', {'categoryId': category.id})%}
            <tr>
                <td>{{ category.id }}</td>
                <td>{{ category.label }}</td>
                <td><a href="{{ editUrl }}">Edit</a></td>
            </tr>
        {% endfor %}
    </table>
{% endblock %}
```

`set editUrl = path('category_edit',{'categoryId': category.id})` is how you assign a value to a variable in Twig. This time we used the **path** function with a second argument `{'categoryId': category.id}` . This argument will be passed to the Controller that will render that route.

- **app/Resources/views/category/edit.html.twig**

```
{% extends 'base.html.twig' %}
{% block body %}
    <form method="post">
        <input type="hidden" value="{{ category.id }}" />
        <table border="1">
            <tr>
                <td>ID</td>
                <td>{{ category.id }}</td>
            </tr>
            <tr>
                <td>Label</td>
                <td>
                    <input type="text" name="label" value="{{ category.label }}" />
                </td>
            </tr>
            <tr>
                <td>Parent</td>
                <td>
                    <select name="parent">
                        <option value="0">No parent</option>
                        {% for parentCategory in categories
                                            if category.id != parentCategory.id %}
                        {% set selected = '' %}

                        {% if (category.parent is not null)
                                    and
                                (category.parent.id == parentCategory.id) %}
                        {% set selected = 'selected' %}
                        {% endif %}
                        <option {{ selected }} value="{{ parentCategory.id }}">
                            {{ parentCategory.label }}
                        </option>
                        {% endfor %}
                    </select>
                </td>
            </tr>
        </table>
        <input type="submit" value="save"/>
    </form>
    <a href="{{ path('category_list') }}">Back to list</a>
{% endblock %}
```

1.4.3 Controller

Now we need to implement the controller actions we referred in the routes.

- Create **src/AppBundle/Controller/CategoryController.php**

```
<?php
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class CategoryController extends Controller
{
    public function listCategoriesAction()
    {
        $arguments = array('categories' => $this->getCategories());
        return $this->render('AppBundle:category:list.html.twig', $arguments);
    }
    public function editCategoryAction($categoryId)
    {
        $arguments = array(
            'category' => $this->getCategory($categoryId),
            'categories' => $this->getCategories()
        );
        return $this->render('AppBundle:category:edit.html.twig', $arguments);
    }
    public function saveCategoryAction($categoryId)
    {
        return $this->redirectToRoute('category_list');
    }
    private function getCategories()
    {
        return array(
            1 => array('id' => 1, 'label' => 'Phones', 'parent' => null),
            2 => array('id' => 2, 'label' => 'Computers', 'parent' => null),
            3 => array('id' => 3, 'label' => 'Tablets', 'parent' => null),
            4 => array('id' => 4, 'label' => 'Desktop', 'parent' => array(
                    'id' => 2,
                    'label' => 'Computers')
            ),
            5 => array('id' => 5, 'label' => 'Laptop', 'parent' => array(
                    'id' => 2,
                    'label' => 'Computers')
            ),
        );
    }
    private function getCategory($categoryId)
    {
        $categories = $this->getCategories();
        return $categories[$categoryId];
    }
}
```

At this point we are done, let's check if everything is working as expected.

- http://symfony.local/category/list should show the list of 5 categories with edit links.
- http://symfony.local/category/edit/1 should show a form to edit the label and the parent of the category Phones. After submitting the form, you should get back to the category listing page.

> When developing, you usually want to run your application in **dev** mode. All you need is to prefix any URL with **app_dev.php**. The dev mode is very useful when working on cacheable resources like templates, routes, doctrine meta-data, configurations, parameters.. Because those resources get parsed on every request. When running in prod mode, you need to clear your application's cache whenever you make a change to a cacheable resource. Another good reason to run the application in dev mode when working on it, is the debugging informations available about routes, translations, templates, queries...

1.5 Exercises

1.5.1 Return an HTTP 404 response when CategoryController::editCategoryAction is called with an invalid categoryId.

1.5.2 Add a new route **category_list_json** that

- Has the same path as **category_list** route
- Returns a JSON response
- Is matched when the request content-type is **application/json**

1.5.3 Create ProductController with the following actions: **listProducts editProduct saveProduct**. A product will have the following members **id**, **title**, **code** and **decription**.
ProductController will have a method `getProducts` that returns a hard-coded array of few products.

1.5.4 Add a new member to the product structure, name it **categories**. It will contain the categories to which the product belongs. Update the controllers and the product list view to show the new change.

Product item example:

```
array(
  'id' => 1,
  'title' => 'Padfone2',
  'code' => 'padfone2',
  'description' => 'an awesome padfone',
  'categories' => array(
    0 => array(
      'id' => 1,
      'label' => 'phones',
      'parent' => null
    ),
    1 => array(
      'id' => 3,
      'label' => 'tablets',
      'parent' => null
    )
  )
)
```

Product list example:

#	Code	Title	Description	Categories	Actions
1	padfone2	Padfone2	an awesome padfone	Phones Tablets	Edit
2	gforce750xdf15	Gforce 750x df-15	an awesome graphic card	Computers	Edit

To keep consistency between CategoryController and ProductController without introducing code duplication, move the `getCategory` and `getCategories` from the controller to an other class that can be shared between other controllers.