Symfony 4 CRUD Samenvatting

- Date: 2020-07-16
- Door: Amsterdam UMC, Apotheek I&A, & Dion Dresschers
- License: CC BY-SA 4.0
- Versie: 2020-08-06 10:06:26

Symfony Officiele Documentatie

- Symfony releases, notifications and release checker
- Coding Standards (Symfony Docs)
- Symfony Documentation
- Symfony Recipes Server
- Installing & Setting up the Symfony Framework (Symfony Docs)

Composer

> Composer is "A Dependency Manager for PHP".

- Composer

1. Install Composer lokaal in the folder:

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('sha384', 'composer-setup.php') ===
'e5325b19b381bfd88ce90a5ddb7823406b2a38cff6bb704b0acc289a09c8128d4a8ce2bbafc
d1fcbdc38666422fe2806') { echo 'Installer verified'; } else { echo
'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

2. To use composer `php composer.phar`

PHP

1. Check the PHP version:
    - `php -v`

Symfony

Symfony - Installing

- Minimal version:
    - `composer create-project symfony/skeleton <my_project_name>`
- Web version:
    - `composer create-project symfony/website-skeleton <my_project_name>`
- Specific version (in this case 5.1):
    - `php composer.phar create-project symfony/skeleton <my_project_name> 5.1`
- Check the requirements
    - `php composer.phar require symfony/requirements-checker`
- If you want to run an older version, and there is no symfonty/skeleton for that version (for instance version 4.1), then specify it like this:

```
php composer.phar create-project symfony/skeleton:^4.1 <my_project_name>
```

Symfony - Folder Structure

- bin - unit tests
- config - configuration files
- public - the server servers from here
- src - here you program your application
- var - cache files and sessesions
- vendor - external php libraries

Symfony - Requrements Checker

1. After you've created a project, check in the project folder for the requirements. This will create a 'composer.json' and 'composer.lock' file (!?) and also the 'check.php' file in the 'public' directory:
    - `php ../composer.phar require symfony/requirements-checker`
2. Open the 'check.php' file in the browser to see all requirements.
3. After you've resolved all issues, remove the requirements checker again (wich deleted the 'check.php file again).
    - `php ../composer.phar remove symfony/requirements-checker`

Symfony - Configuration Files

- The morst important config file is the '.env' file in the project folder.

In the '.env' file you have the 'APP_ENV' variable which could have the value of:

- prod
- dev
- test

The environments correspond with the directory 'prod', 'dev' and 'test' inside different places of the 'conf' directory.

You also see inside the 'conf' directory the files:

- bundles.php
- routes.php
- services.yml

and you can (!?) add '_dev' or '_test' to that file if you want to use it for a specific environment. So you can add 'bundles_dev.php' to that folder.

Symfony - Flex

Symfony flex is a Composer component to install packages. You can find packages via: Symfony Recipes Server

Symfony - Flex - Twig

For instance you can install the Twig bundle via: symfony/twig-bundle - Packagist

From that you can copy the code `composer require symfony/twig-bundle`, so execute:

- `php ../composer.phar require symfony/twig-bundle`

You can due to Aliases also use:

- `php ../composer.phar require twig`

Composer will update the `composer.json` file.

You can see the `twig` directory has been added to the `vendor` directory.

Also the folder `templates` directory has been added and a `base.html.twig` template. And some `twig.yaml` files to the serveral `config` directories.

Syfmony - Flex - Doctrine

Install ORM-Pack for Doctrine:

```
php ../composer.phar require symfony/orm-pack
```

You can also see an entry for the database has been added in the `.env` file:

- `cat .env | grep ^DATABASE`

returns:

```
DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=5.7
```

You can also see that some additional Symfony console commands have been added via: `php bin/console | grep doctrine`

Symfony - Commando's

You can run: `php bin/console` to see all Symfony commands.

- See the top directories in a Symfony project:
  - `tree -d -L 1`
- Run a testserver from the Symfony project (where 'public' is the directory where to server from):
  - `php -S 127.0.0.1:8000 -t <public>`

Symfony - Routes

Symfony - Routers - Not! Annotiations

You can see with `cat config/routes.yaml` that the way of Routing is disabled, because the result is:

```
#index:
#    path: /
#    controller: App\Controller\DefaultController::index
```

- You can see that `:index` is the name of the Route.
- The URL (`path`) is the root folder.
- The `::index` is the index method of the controller.

If you want to create that, than uncomment all config.

In VI open the file `config.routes.yaml` and there give the VI command: `:s/^#//gc`

Or with SED: `sed -i 's/^# //' routes/routes.yaml`

So add the controller: `touch src/Controller/DefaultController.php` and make sure that the file looks like:

```
<?php

namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;

class DefaultController
{

    public function index()
    {
        return new Response("<h1>It works</h1>");
    }
}
```

Now it should work. Delete the file again: `rm src/Controller/DefaultController.php`

And add the comments again for the `config/routes.yaml` file via `sed -i "s/^/# /" config/routes.yaml`

Symfony - Routes - Via Annotation

Via maker:

```
composer require symfony/maker-bundle
```

Now we see there are a lot of new Symfony commands, you can see them via: `php bin/console | grep -e make:`

You can create a new controller via: `php bin/console make:controller DefaultController` this will create the files:

- `src/Controller/DefaultController.php`
- `templates/default/index.html.twig`

Check if the route is working via 'http://localhost:8000/default'

> A controller is a PHP function that reads information from the request object and created and returns a response object.

You see the Controller:

```
class DefaultController extends AbstractController
{
    /**
     * @Route("/default", name="default")
     */
    public function index()
    {
        return $this->render('default/index.html.twig', [
            'controller_name' => 'DefaultController',
        ]);
    }
}
```

Symfony - Routes - JSON Response

You can next to render a response also return an json response if you are building an API.

You can change the `return` object from above to JSON:

```
return $this->json(['username]=>'john.doe');
```

and then view the result inside a browser.

Symfony - Routes - Simple HTML Response

Use this is you want to only use a simple HTML response:

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

And than:

`return new Response("<h1>Hey</h1>");`

Symfony - Routes - Redirect

Use: `return $this->redirect('https://www.example.com);`

Or in most cases you redirect to other Routes:

```
class DefaultController extends AbstractController
{
    /**
     * @Route("/default/{name}", name="default")
     */
    public function index($name)
    {
        #return $this->render('default/index.html.twig', [
        #    'controller_name' => 'DefaultController',
        #]);
        #return $this->json(['username'=>'john.doe']);
        return $this->redirectToRoute('default2');
    }

    /**
    * @Route("/default2/", name="default2")
    */
    public function index2()
    {
        return new Response("I am from default2 route!");
    }

}
```

Symfony - Begin met de CRUD

1. Maak een nieuwe project aan:
    - `composer create-project symfony/skeleton OpenDIB2 ^4.4`

Controller en de bijbehorende packages

1. Probeer een nieuwe controller aan te maken:
    - `bin/console make:controller ToDoListController`
2. Negatief resulaat:

```
[error] Error thrown while running command "'make:controller'
ToDoListController". Message: "There are no commands defined in the "make"
```

```
namespace.

You may be looking for a command provided by the "MakerBundle" which is
currently not installed. Try running "composer require symfony/maker-bundle
--dev"."
```

3. Dus draai zoals geadviseerd:
    - `composer require symfony/maker-bundle --dev`
4. Probeer weer een nieuwe controller te maken:
    - `bin/console make:controller ToDoListController`
5. Het lukt nog steeds niet en resulteert in:

```
[ERROR] Missing package: to use the make:controller command, run:

        composer require doctrine/annotations
```

6. Draai dus:
    - `composer require doctrine/annotations`
7. Nu kan je wel succesvol draaien:
    - `bin/console make:controller ToDoListController`
8. Je ziet dat er een nieuwe controller file aangemaakt is:

```
created: src/Controller/ToDoListController.php

 Success!

Next: Open your new controller class and add some pages!
```

9. You can see the new annotation route:
    - `grep -e @Route src/Controller/ToDoListController.php`
10. This results in:
    - `* @Route("/to/do/list", name="to_do_list")`
11. Test the new Controller:
    - `firefox http://192.168.56.102:82/to/do/list`
12. This is not working as it uses Apache2 rewrite rules. Make sure this module is enabled:
    - `a2query -m | grep rewrite`
13. If this is enabled the result will be:
    - `rewrite (enabled by site administrator)`
14. You can check if the route is served:
    - `bin/console debug:router`
15. You can see the route `/to/do/list` is served:

```
---------------- -------- -------- ------ --------------------------
 Name             Method   Scheme   Host   Path
```

```
---------------- -------- -------- ------ --------------------------
 _preview_error   ANY      ANY      ANY    /_error/{code}.{_format}
 to_do_list       ANY      ANY      ANY    /to/do/list
---------------- -------- -------- ------ --------------------------
```

16. If it is not yet running, you can create the specific rewrite rules in a local `.htaccess` file that resided inside the `public`-directory.
    - `composer require symfony/apache-pack`
17. If it is still not working, copy the content of the `public\.htaccess` folder inside the Apache2 virtual host config file as described in Configuring a Web Server (Symfony 4.4 Docs)

Use Twig files

1. If it is working we want to use templating via Twig, so install this package:
    - `composer install twig`
2. This will create the folder `templates` and inside that folder the template Twig file: `base.html.twig`
3. Als je de text bekijkt met:
    - `cat templates/base.html.twig`
4. Dan ziet de template ziet er zo uit:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>{% block title %}Welcome!{% endblock %}</title>
        {% block stylesheets %}{% endblock %}
    </head>
    <body>
        {% block body %}{% endblock %}
        {% block javascripts %}{% endblock %}
    </body>
</html>
```

5. Maak een nieuwe template aan:
    - `touch templates/index.html.twig`
6. Zorg dat de nieuwe template de eerder gemaakt `base.html.twig` gaat gebruiken als basis. Het volgende commando:
    - `cat templates/index.html.twig`
7. Geeft als resultaat:

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

{% block title %}De titel van de specifieke pagina{% endblock title %}

{% block body %}
```

```
Dit is van de Controller genaamd: ToDoList,
{% endblock body %}
```

8. Zorg dat de controller nu een andere respons geeft en pas de route aan zodat dit:
    - `cat src/Controller/ToDoListController.php`
9. Het volgende resultaat oplevert:

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;

class ToDoListController extends AbstractController
{
    /**
    * @Route("/", name="to_do_list")
    */
    public function index()
    {
        return $this->render('index.html.twig');
    }
}
```

10. Test of route klopt:
    - `firefox http://http://192.168.56.102:82/`
11. Meer info over Twig is te vinden via Documentation - Twig - The flexible, fast, and secure PHP template engine
12. Twig is gebaseerd op Jinja — Jinja Documentation (2.11.x)

Bootstrap

1. Je kunt Bootstrap gebruiken om het uiterlijk te verbeteren zodat je zelf geen CSS hoeft te schrijven:
    - `composer require twbs/bootstrap:4.5.0`
2. Je kan zien dat alle files gedownload worden in de `vendor/twbs/bootstrap`-directory.
    - `tree vendor/twbs/bootstrap`
3. Copieer de `dist` directory met daarin `CSS` en `JS` directories naar `public`
    - `cp -r vendor/twbs/bootstrap/dist/ public/`
4. Hernoem de `public/dist/` filder naar `public/assets`
    - `mv public/dist public/assets`
5. Zie dat in de `public/assets`-directory alle CSS en JavaScript files zitten:
    - `tree public/assets`
6. Je kan refereren in je `base.html.twig` file naar de CSS Bootstap file via:
    - `<link rel="stylesheet" href="./assets/css/bootstrap.css">`

Maak en nieuwe Create controller (via POST)

1. In de ToDoListController maak deze functie/method aan:

```
/**
 * @Route("/create", name="create_task")
 */
public function create()
{
    exit("TO DO: Create a new task!");
}
```

2. In het HTML formulier, moet je de `name`, dus `create_task` van die route/annotation gebruiken als `action` zodat:
    - `cat templates/index.html.twig | grep "form action"`
3. Resulteert in:
    - `<form action="{{ path('create_task') }}">`
4. HTML formulier worden meesteal verstuurd via een `POST` method, dus dat veranderen we ook:
    - `<form action="{{ path('create_task') }}", method="POST">`
5. En ook in de annotation in `Controller\ToDoListController`:
    - `* @Route("/create", name="create_task", methods={"POST"} )`

Maak een nieuwe Update controller aan (via GET) met een `id`

1. Zorg dat deze functie/method gemaakt wordt in de ToDoListController:

```
 /**
 * @Route("/switch_status", name="switch_status")
 */
public function switch_status()
{
    exit("TO DO: Switch_status task!");
}
```

2. Echter als je een status van een reeds bestaande To Do item wilt verwijderen, heb je eerst het ID nodig van het To Do item dat je wilt veranderen.
3. Zorg dat de method er zo uit komt te zien:

```
/**
 * @Route("/switch_status/{id}", name="switch_status")
 */
public function switch_status($id)
```

```
{
    exit("TO DO: Switch_status task number id: " .$id);
}
```

4. In de Twig template geeft de URL aan in de `href` en maak een tijdelijk `id` aan voor `<a href>` aangezien we nog geen database hebben:

```
<li>
    <a href="{{ path('switch_status', {'id':1}) }}">
        <span class="task-list-item">Do shopping </span>
    </a>
    <a onclick="return confirm('Are you sure to delete?')" href="delete">
<span class="close">X</span></a>
</li>
```

5. Pas je `id` in de URL aan naar bijvoorbeeld `8`,dus `192.168.56.102:82/switch_status/8/` dan zie je dat de ID ook meegenomen wordt:
    - `fire 192.168.56.102:82/switch_status/8`

Maak een nieuwe Delete controller aan

1. Zorg dat de nieuwe Method in de Controller er zo uit ziet:

```
/**
 * @Route("delete/{id}", name="task_delete")
 */
public function delete($id)
{
    exit("TO DO: Delete task number id: " .$id);
}
```

2. Update de Twig file, zodat drukken op de `x` de juiste `href` heeft om het juiste `id` de deleten via `href="{{ path('task_delete, {'id':1}) }}"`:

```
  <li>
    <a href="{{ path('switch_status', {'id':1}) }}"> <span class="task-list-
item">Do shopping </span> </a
    ><a onclick="return confirm('Are you sure to delete?')" href="{{
path('task_delete, {'id':1}) }}"><span class="close">X</span></a>
  </li>
```

Database

Database - Maak de database aan

1. Installeer de package Doctrine voor de ORM:
    - `composer require doctrine`
2. In de output zie je dat je nog de `DATBASE_URL` configuratie moet aanpassen in den `.env` file:

```
Database Configuration

* Modify your DATABASE_URL config in .env

* Configure the driver (mysql) and
    server_version (5.7) in config/packages/doctrine.yaml
```

3. Probeer eerst of de standaard `root` gebruiker in `mysql` het door jou gebruikte wachtwoord gebruikt:
    - `mysql -h 127.0.0.1 -u root -p`
4. Is de login succesvol pas dan de `.env` file aan:
    - `vi .env`
5. Zodat de config voor `DATABAE_URL` aangepast is en ook de database:
    - `DATABASE_URL=mysql://root:{wachtwoord}@127.0.0.1:3306/db_todolist?serverVersion=5.7`
6. Nu kan je de database door Symfony laten aanmaken:
    - `bin/console doctrine:database:create`
7. Het resultaat is:

```
Created database `db_todolist` for connection named default`
```

Database - Maak Entities aan

1. Maak nu tabellen aan via Entity:
    - `bin/console make:entity`
2. Je komt in dit vraag/antwoord geval uit:

```
Class name of the entity to create or update (e.g. BraveKangaroo):
> Task

created: src/Entity/Task.php
created: src/Repository/TaskRepository.php

Entity generated! Now let's add some fields!
You can always add more fields later manually or by re-running this command.

New property name (press <return> to stop adding fields):
> title
```

```
Field type (enter ? to see all types) [string]:
> string

Field length [255]:
> 255

Can this field be null in the database (nullable) (yes/no) [no]:
> no

updated: src/Entity/Task.php

Add another property? Enter the property name (or press <return> to stop
adding fields):
> status

Field type (enter ? to see all types) [string]:
> boolean

Can this field be null in the database (nullable) (yes/no) [no]:
> yes

updated: src/Entity/Task.php

Add another property? Enter the property name (or press <return> to stop
adding fields):
>

Success!

Next: When you're ready, create a migration with php bin/console
make:migration
```

3. Je hebt dus nu gedefinieerd de MySQL tabel `Tasks`, waarin de MySQL kolommen staan `title` en `status`.

4. Je ziet dat de volgende files zijn aangemaakt:

```
created: src/Entity/Task.php
created: src/Repository/TaskRepository.php
```

5. Je ziet ook dat in de `src\Entity\Task.php`-file de kolommen staat, inclusief de autotmatische aangemaakte kolom `id`:

    - `cat src/Entity/Task.php | grep private`

6. Want het resultaat is:

```

```

```
private $id;
private $title;
private $status;
```

7. Je ziet ook dat automatische wat methods zijn aangemaakt om data op te halen uit de database en te schrijven:
    - `cat src/Entity/Task.php | grep function`
8. Wat resulteert in:

```
public function getId(): ?int
public function getTitle(): ?string
public function setTitle(string $title): self
public function getStatus(): ?bool
public function setStatus(?bool $status): self`
```

9. Nu moet je eerst een Migration file aanmaken zoals geadviseerd in de vraag/antwoord console `Next: When you're ready, create a migration with php bin/console make:migration`
    - `bin/console make:migration`
10. Waarbij het resultaat is:

```

Success!

Next: Review the new migration "migrations/Version20200731103451.php"
Then: Run the migration with php bin/console doctrine:migrations:migrate
See
https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html
```

11. Je ziet dat de versiebeheerde migration file is aangemaakt:
    - `ls migrations/`
12. Wat resulteert in:
    - `Version20200731103451.php`
13. Nu maak de tablelen aan zoals geadviseerd in de output `Then: Run the migration with php bin/console doctrine:migrations:migrate`
    - `bin/console doctrine:migrations:migrate`
14. Na het bevestigen via 'yes' of [ENTER]-key, zie je deze hele output:

```

```

```
WARNING! You are about to execute a database migration that could result in
schema changes and data loss. Are you sure you wish to continue? (yes/no)
[yes]:
> yes

[notice] Migrating up to DoctrineMigrations\Version20200731103451
[notice] finished in 101.1ms, used 12M memory, 1 migrations executed, 1 sql
queries
```

15. Nu zijn de tabllen aangemaakt, zoals je kan zien in mysql:

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 17
Server version: 5.7.31-0ubuntu0.18.04.1 (Ubuntu)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| db_todolist        |
| mysql              |
| new_db             |
| performance_schema |
| sys                |
| wordpress          |
+--------------------+
7 rows in set (0.00 sec)

mysql> use db_todolist;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+----------------------------+
| Tables_in_db_todolist      |
+----------------------------+
| doctrine_migration_versions |
| task                       |
```

```
+----------------------------+
2 rows in set (0.00 sec)

mysql> describe task;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| id     | int(11)      | NO   | PRI | NULL    | auto_increment |
| title  | varchar(255) | NO   |     | NULL    |                |
| status | tinyint(1)   | YES  |     | NULL    |                |
+--------+--------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> exit;
Bye
```

Update the Create method

1. First in the Twig file, privide a `name` for the `input` field:

```
  <form action="{{ path('create_task') }}" method="POST">
    <input name="title" type="text" id="myInput" placeholder="Task title..."
/>
    <button type="submit" class="addBtn">Add a task</button>
  </form>
```

2. A Request Method contains `GET` and `POST` data, so we are going to use this.

3. So make sure you import the Request method:

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

4. Then use the Request in the Create Method, and use the `title` which is the `name` of the HTML we've just adapted.

```
/**
 * @Route("/create", name="create_task", methods={"POST"} )
 */
public function create(Request $request)
{
    exit($request->request->get('title'))
}
```

5. You can check that this works and you push `Add a task`

    - `firefox http://192.168.56.102:82/`

6. Now you can add some validation code, to check the name of the `title`:

```
public function create(Request $request)
{
    $title = trim($request->request->get('title'));
    if(empty($title))
    return $this->redirectToRoute('to_do_list');
    exit($request->request->get('title'));
}
```

Database - Save entry in the database

1. Now we are going to save that new task in de database via the getManager() method.
2. Change the Contoller so that it looks like:

```
public function create(Request $request)
    {
        $title = trim($request->request->get('title'));
        if(empty($title))
        return $this->redirectToRoute('to_do_list');

        $entityManager = $this->getDoctrine()->getManager();

        $task = new Task; // make sure to include `use App\Entity\Task`
        $task->setTitle($title); // You can find these kind of Methods in
`Entity/Task.php`

        $entityManager->persist($task);
        $entityManager->flush();

        return $this->redirectToRoute('to_do_list');
    }
```

3. Make sure to import the new Entity Class called `Task` with:
    - `use App\Entity\Task`
4. After making a new task you can see that it was saved in the database:

```
mysql> select * from task;
+----+------------------+--------+
| id | title            | status |
+----+------------------+--------+
|  1 | Go sit on an egg |   NULL |
+----+------------------+--------+
1 row in set (0.00 sec)
```

Laat alle database entries zien

1. Om alle To Do items te laten zien in de index, haal alle entries van de database in de view:

```
public function index()
{
    $tasks = $this->getDoctrine()->getRepository(Task::class)->findAll();
    return $this->render('index.html.twig', ['tasks'=>$tasks] ); // the
`tasks` will be the tasks which is usable in Twig, notice [ and ], as it is
an array
}
```

2. Nu kan in de Twig template door tasks heen worden geloopt:

```
<ul>
    {% for task in tasks %}

        <li class="checked">
            <a href="{{ path('switch_status', {'id':1}) }}"> <span
class="task-list-item">Do shopping </span> </a
            ><a onclick="return confirm('Are you sure to delete?')" href="{{
path('task_delete', {'id':1}) }}"><span class="close">X</span></a>
        </li>

    {% endfor %}
</ul>
```

3. Controlleer of je het aantal entries ziet. Let Op dat de titel (nog) steeds hetzelfde is, namelijk `Do shopping`:
    - `firefox http://192.168.56.102:82/`
4. Pas nu de titel aan in de Twig template:
    - `<a href="{{ path('switch_status', {'id':1}) }}"> <span class="task-list-item"> {{ task.title }} </span> </a>`
5. Nu kan je ook het ID dynamisch aanpassen in de Twig template die we eerder vast op `1` hebben gezet:

```
        <li class="checked">
            <a href="{{ path('switch_status', {'id':task.id}) }}"> <span
class="task-list-item"> {{ task.title }} </span> </a>
            <a onclick="return confirm('Are you sure to delete?')" href="{{
path('task_delete', {'id':task.id }) }}"><span class="close">X</span></a>
        </li>
```

6. Check nu in de browser en hover over de 'x' om te checken of in de statusbar het juiste ID wordt meegegeven:
    - `firefox http://192.168.56.102:82/`
7. Voeg nu wat `line-through` CSS toe voor taksk die klaar zijn (let op de `not` die er straks nog uitgehaald wordt, want alle `status` is nog True):

```
        <li>
            <a href="{{ path('switch_status', {'id':task.id}) }}"
                {% if not task.status %}
                style="text-decoration-line: line-through"
                {% endif %}
                ><span class="task-list-item"> {{ task.title }} </span>
            </a>
            <a onclick="return confirm('Are you sure to delete?')" href="{{
path('task_delete', {'id':task.id }) }}"><span class="close">X</span></a>
        </li>
```

8. Ja kan ook soreteren, dus de nieweste Task bovenaan. Pas je `findAll()` method aan door een `findBy()`-method, waarbij de eerste Array de criterea aangeeft (deze laten we leeg, want we willen alle items en de tweede array de sortering is):
    - `$tasks = $this->getDoctrine()->getRepository(Task::class)->findBy([], ['id'=>'DESC']);`

Database - Update de `status`

1. Update de `switch_status` method/controller:

```
/**
 * @Route("/switch_status/{id}", name="switch_status")
 */
public function switch_status($id)
{
    $entityManager = $this->getDoctrine()->getManager(); // getManager for
saving to the database
    $task = $entityManager->getRepository(Task::class)->find($id);

    $task->setStatus( ! $task->getStatus() ); // Invert it, and the
`getStatus` and `setStatus` method is defined in the `Entity/Task.php` file

    $entityManager->flush();
    return $this->redirectToRoute('to_do_list');
}
```

2. Controlleer of het werkt. Klik op de naam om te kunnen switchen.

- `firefox http://192.168.56.102:82/`

Database - Delete een taak

1. Om meer te zien van Doctrine, lees Databases and the Doctrine ORM (Symfony 4.4 Docs)
2. Pas de `delete` method/controller zo aan dat je het specifieke `id` kan verwijderen:

```
/**
 * @Route("delete/{id}", name="task_delete")
 */
public function delete($id)
{
    $entityManager = $this->getDoctrine()->getManager();
    $task = $entityManager->getRepository(Task::class)->find($id);
    $entityManager->remove($task);
    $entityManager->flush();
    return $this->redirectToRoute('to_do_list');
}
```

3. Je kan ook via Doctrine direct een SQL query naar de datbase sturen:
    - `php bin/console doctrine:query:sql 'SELECT * FROM task'`
4. Resulteert in:

```
array(4) {
[0]=>
array(3) {
    ["id"]=>
    string(1) "1"
    ["title"]=>
    string(16) "Go sit on an egg"
    ["status"]=>
    string(1) "1"
}
[1]=>
array(3) {
    ["id"]=>
    string(1) "2"
    ["title"]=>
    string(16) "Go Sit on an egg"
    ["status"]=>
    string(1) "0"
}
[2]=>
array(3) {
    ["id"]=>
    string(1) "3"
```

```
    ["title"]=>
    string(20) "Ga een kuiken zuigen"
    ["status"]=>
    string(1) "1"
  }
  }
```