

# PRAKTEK LANGSUNG



*Symfony Framework*

# Praktek Langsung Framework Symphony

Belajar Framework Symphony Menggunakan Contoh

Muhamad Surya Iksanudin

Buku ini dijual di

<http://leanpub.com/Praktek-Langsung-Framework-Symfony>

Versi ini diterbitkan pada 2016-06-17



Leanpub

This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2016 Muhamad Surya Iksanudin

# Contents

- Kenapa Memilih Symfony . . . . . 1
  - I. Sekilas tentang Symfony . . . . . 1
  - II. Alasan memilih Symfony . . . . . 1
- Pengenalan *Controller* dan *Routing* pada Symfony . . . . . 9
  - I. *Controller* pada Symfony . . . . . 9
  - II. *Routing* pada Symfony . . . . . 10
  - III. *Route Parameter* . . . . . 13
  - IV. *HTTP Method* pada Symfony . . . . . 14
- Membuat Aplikasi Kontak Sederhana . . . . . 18
  - I. Pengantar . . . . . 18
  - II. Pembuatan *Entity* . . . . . 18
  - III. Pembuatan Form . . . . . 24
  - IV. Pembuatan *Controller* . . . . . 25
  - V. Pembuatan View . . . . . 35
  - VI. Menjalankan Aplikasi . . . . . 41
  - VII. Kesimpulan . . . . . 43
- Menambahkan Group pada Aplikasi Kontak . . . . . 44
  - I. Membuat CRUD untuk Group . . . . . 44
  - II. Limitasi . . . . . 44
  - III. Meng-*update form* dan *view* Contact . . . . . 45
  - IV. Menjalankan Aplikasi . . . . . 47

# Kenapa Memilih Symfony

## I. Sekilas tentang Symfony

Symfony adalah salah satu dari banyak framework yang ditulis dalam bahasa PHP. Selain symfony ada juga CodeIgniter, Yii, Cake, Laravel, Slim, Silex dan lain sebagainya. Symfony pertama kali diperkenalkan pada tahun 2005 oleh Fabien Potencier, seorang *programmer* asal Perancis serta pendiri SensioLabs.

Saat buku ini ditulis, Symfony telah mencapai versi 3.1.X serta 2.7.X dan 2.8.X untuk versi LTS (*Long Term Support*)

Saat ini, Symfony telah diadopsi dan digunakan oleh banyak perusahaan maupun project open source diantaranya EzPublish, Drupal, Joomla, Magento, Laravel, Composer, dan masih banyak lagi lainnya.

Symfony dimaintain oleh lebih dari 1.500 *developer* pada *core component* dan lebih dari 1.100 developer pada dokumentasi serta lebih dari 10.000 *third party bundle* yang siap digunakan dalam *project* Symfony Anda.

Saat ini, Symfony telah di-*download* lebih dari 12.500.000 kali hanya untuk versi 2 keatas sehingga merupakan pilihan yang tepat menggunakan Symfony sebagai solusi dari kebutuhan Sistem Anda.

## II. Alasan memilih Symfony

Seringkali sebelum memakai suatu *framework*, kita akan bertanya, kenapa *kok* harus pakai *framework* X, kenapa bukan *framework* Y saja? Untuk menjawab pertanyaan tersebut, maka disini saya

akan mencoba menjabarkan keunggulan Symfony dibandingkan *framework* PHP lainnya.

### a. Reputasi

Dengan lebih dari 12 juta *download*, reputasi Symfony tidak perlu diragukan lagi. Terlebih sebagai sebuah *framework*, Symfony telah bertahan lebih dari 10 tahun serta menjadi pondasi banyak project lain, sehingga tidak ada yang perlu diragukan lagi bahwa Symfony adalah *framework* dengan reputasi yang baik.

### b. Standar tinggi

Banyaknya *component* Symfony yang digunakan oleh *project* lain menunjukkan bagaimana kualitas dari Symfony itu sendiri. Symfony disusun mengikuti kaidah (*standard*) yang disepakati oleh komunitas PHP diseluruh dunia yaitu **PSR** (*PHP Standard Recommendations*)<sup>1</sup> sehingga untuk kualitas *code*, Symfony tidak perlu diragukan.

### c. *Support* banyak database

Untuk koneksi database, Symfony menggunakan Doctrine sebagai *provider*-nya. Doctrine sendiri adalah *Database Library* yang mendukung banyak data storage baik itu **RDMS** seperti MySQL, PostgreSQL, Oracle, Ms SQL, SQLite, dan MariaDB maupun **NoSQL database** seperti MongoDB, CouchDB dan *PHPCR* (*PHP Content Repository*).

Dengan dukungan *database* yang banyak tersebut, Symfony sangat mudah diimplementasikan untuk *Enterprise* yang memiliki *multiple data storage*. Bahkan dengan Symfony, Anda akan lebih mudah mengaplikasikan konsep **Polyglot**<sup>2</sup> pada aplikasi Anda.

---

<sup>1</sup><http://www.php-fig.org/psr>

<sup>2</sup><http://martinfowler.com/bliki/PolyglotPersistence.html>

#### **d. *Third party* melimpah**

Dengan lebih dari 10.000 *third party bundle*, Anda akan sangat dimanjakan ketika membangun aplikasi karena apa yang Anda butuhkan hampir semuanya telah di-*support* oleh komunitas. Berikut adalah beberapa *bundle* yang bisa menjadi referensi ketika Anda membangun aplikasi.



## Bundles Symfony

User Management ([FosUserBundle<sup>3</sup>](#))

Menu ([KnpMenuBundle<sup>4</sup>](#))

Pagination ([KnpPaginatorBundle<sup>5</sup>](#), [PagerfantaBundle<sup>6</sup>](#))

Database ([DoctrineBundle<sup>7</sup>](#), [DoctrineFixturesBundle<sup>8</sup>](#), [DoctrineMongoDBBundle<sup>9</sup>](#))

Cache ([DoctrineCacheBundle<sup>10</sup>](#), [FosHttpCacheBundle<sup>11</sup>](#))

Admin Generator ([SonataAdminBundle<sup>12</sup>](#), [SymfonyIdAdminBundle<sup>13</sup>](#))

Restful Api ([FosRestBundle<sup>14</sup>](#), [NelmioApiDocBundle<sup>15</sup>](#), [BazingaHateOasBundle<sup>16</sup>](#))

Serialization ([JmsSerializationBundle<sup>17</sup>](#))

Dan masih banyak lagi *bundle-bundle* lainnya yang tidak bisa saya sebutkan disini. Untuk melihat *bundle-bundle* apa saja yang tersedia, salah satu *channel* yang dapat dikunjungi adalah [KnpBundles<sup>18</sup>](#).

---

<sup>3</sup><http://symfony.com/doc/current/bundles/FOSUserBundle/index.html>

<sup>4</sup><http://symfony.com/doc/current/bundles/KnpMenuBundle/index.html>

<sup>5</sup><https://github.com/KnpLabs/KnpPaginatorBundle>

<sup>6</sup><https://github.com/whiteoctober/WhiteOctoberPagerfantaBundle>

<sup>7</sup><http://symfony.com/doc/master/bundles/DoctrineBundle/index.html>

<sup>8</sup><http://symfony.com/doc/master/bundles/DoctrineMongoDBBundle/index.html>

<sup>9</sup><http://symfony.com/doc/master/bundles/DoctrineMongoDBBundle/index.html>

<sup>10</sup><http://symfony.com/doc/master/bundles/DoctrineCacheBundle/index.html>

<sup>11</sup><https://github.com/FriendsOfSymfony/FOSHttpCacheBundle>

<sup>12</sup><http://symfony.com/doc/master/bundles/SonataAdminBundle/index.html>

<sup>13</sup><https://github.com/SymfonyId/SymfonyIdAdminBundle>

<sup>14</sup><http://symfony.com/doc/master/bundles/FOSRestBundle/index.html>

<sup>15</sup><http://symfony.com/doc/master/bundles/NelmioApiDocBundle/index.html>

<sup>16</sup><https://github.com/willdurand/BazingaHateoasBundle>

<sup>17</sup><http://jmsyst.com/bundles/JMSSerializerBundle>

<sup>18</sup><http://knpbundles.com>

### ***e. Built-in Email Library***

Symfony secara *default* sudah *include email library* yaitu Swift-Mailer sehingga Anda tidak perlu lagi meng-*install email library* untuk berkirim *email* melalui aplikasi Anda. Cukup *setting* dan konfigurasi aplikasi Symfony Anda maka Anda sudah dapat mengirim email dengan sangat mudah.

### ***f. Templating Engine***

Symfony menggunakan [Twig](http://twig.sensiolabs.org/)<sup>19</sup> sebuah *template engine* besutan Sensiolabs yang juga ditulis oleh penulis Framework Symfony yaitu Fabien Potencier. Dengan twig, Anda dapat dengan mudah berkolaborasi dengan *frontend developer* karena syntax-nya yang *friendly*.

*Template twig* di-*compile* kedalam *plain old php object (POPO)* sehingga untuk eksekusi jauh lebih cepat.

### ***g. Built-in Security***

*Security* di Symfony dibuat *Out of The Box* dimana sebuah aplikasi dan sistem keamanan dibuat seolah-olah terpisah. Maksud dari terpisah adalah, Anda dapat membuat aplikasi secara utuh hingga benar-benar berjalan tanpa perlu memikirkan *security* terlebih dahulu.

Kemudian setelah aplikasi selesai, Anda dapat meng-*inject security* dengan mudah tanpa perlu merubah *code* program sama sekali.

Dengan konsep ini, maka Anda akan terbebas permasalahan *User Management* yang biasanya sering terjadi perubahan-perubahan yang cukup radikal sehingga sistem atau aplikasi yang dibuat harus menyesuaikan dan banyak *code* yang harus dirubah pula.

---

<sup>19</sup><http://twig.sensiolabs.org/>



*Security* di Symfony dibuat berlapis sehingga menjamin keamanan dari aplikasi yang Anda buat. *Security* di Symfony menggunakan konsep *Authentication* dan *Authorization* sehingga seorang *user* harus “*lolos*” pengecekan *authentication* dan *authorization* terlebih dahulu ketika mengakses aplikasi kita.

#### **h. Annotation**

Salah satu fitur yang membedakan Symfony dan framework lainnya adalah adanya fitur *annotation* di Symfony.

*Annotation* adalah sebuah *block doc* (blok komentar) php yang akan di-*parsing* dan dibaca sebagai program oleh Symfony. Dengan *annotation*, kita dapat merubah *flow* aplikasi, menambahkan *security*, melakukan manipulasi *parameter*, dll dengan cara yang sangat mudah dan sekali tanpa merubah *code* program sama sekali.

#### **i. Human friendly configuration**

Meski konfigurasi Symfony dapat berupa php atau xml namun yang direkomendasikan oleh Symfony adalah menggunakan yml. Yml adalah kependekan dari *Yahoo Markup Language* yaitu sebuah *markup language* yang *human friendly* karena tidak menggunakan *tag* seperti html maupun xml.

Syntax yml sangat mudah dibaca dan dimengerti oleh siapapun termasuk oleh mereka yang tidak memiliki latar belakang pemrograman sama sekali. Berikut adalah contoh *syntax* yml.

```
1 parameters:
2     database_host: localhost
3     database_user: root
```

Bagaimana sangat mudah dibaca, bukan?

## j. Satu untuk Semua

Banyaknya *project* lain yang menggunakan Symfony memberikan benefit tersendiri bagi kita. Tanpa disadari dengan mempelajari Symfony kita juga secara tidak langsung telah mempelajari *project-project* tersebut dari sisi *library*-nya yaitu *Symfony Component*. Itu artinya jika nantinya kita diharuskan menggunakan *project-project* tersebut maka kita akan lebih familiar.

## k. Lisensi MIT

Lisensi MIT adalah lisensi yang paling aman untuk bisnis dan juga untuk *developer* karena kita boleh mendistribusikan ulang, memodifikasi, termasuk melisensi ulang *software* yang kita buat. Untuk lebih jelas tentang lisensi MIT, silakan baca [link ini](#)<sup>20</sup>.

## l. *Backward Compability*

Bagi sebuah perusahaan maupun *developer*, dukungan *backward compability* adalah sesuatu yang penting. Dengan dukungan *backward compability*, seorang *developer* maupun perusahaan lebih percaya diri ketika meng-*upgrade* program tidak akan terjadi masalah baik *bug* ataupun *error*.

## m. *Cache Everything*

Berbeda dengan *framework* kebanyakan, Symfony memiliki pola eksekusi yang berbeda antara *development* dan *production*. Pada frase *development*, setiap perubahan dalam konfigurasi, *annotation*, dan lain sebagainya akan di-*compile* dan diperbaharui. Namun ketika memasuki frase *production* maka perubahan-perubahan yang

---

<sup>20</sup><http://symfony.com/doc/current/contributing/code/license.html>

kita lakukan pada konfigurasi maupun *annotation* tidak akan di-*compile*.

Hal tersebut terjadi karena Symfony meng-*compile* semua konfigurasi, *annotation*, dan lain sebagainya kedalam *cache file* sehingga proses eksekusi pada *production* akan jauh lebih cepat dibandingkan dengan *development*.

Dengan alasan-alasan diatas, sudah tepat jika Anda memilih Symfony untuk membangun aplikasi Anda.

# Pengenalan *Controller* dan *Routing* pada Symfony

## I. *Controller* pada Symfony

Secara mudah, *controller* di Symfony adalah semua tipe *callable*<sup>21</sup>, namun secara spesifik *controller* adalah semua *class* yang *extends* *class* `Symfony\Bundle\FrameworkBundle\Controller\Controller` dan memiliki *method/action* yang berakhiran *Action*. Untuk lebih jelas, mari kita lihat *code controller* yang ada di hasil instalasi kita.

```
1  <?php
2
3  namespace AppBundle\Controller;
4
5  use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
6  use Symfony\Bundle\FrameworkBundle\Controller\Controller;
7  use Symfony\Component\HttpFoundation\Request;
8
9  class DefaultController extends Controller
10 {
11     /**
12      * @Route("/", name="homepage")
13      */
14     public function indexAction(Request $request)
15     {
16         // replace this example code with whatever you \
```

---

<sup>21</sup><http://php.net/manual/en/language.types.callable.php>

```
19 need
20     return $this->render('default/index.html.twig', \
21     [
22         'base_dir' => realpath($this->getParameter(\
23 'kernel.root_dir').'/..'),
24     ]);
25     }
26 }
```

Pada *code* diatas, *controller* `DefaultController` meng-*extends* `Controller` dan memiliki *action* `indexAction`. *Controller* di Symfony secara *default* tidak memiliki *constructor*. Sebagai gantinya, dengan meng-*extends* `Symfony\Bundle\FrameworkBundle\Controller\Controller` Anda dapat mengakses semua fitur yang ada di Symfony.

Karena fokus kita pada pembahasan *controller*, maka baris *code* yang lainnya tidak kita bahas terlebih dahulu agar tidak melebar pembahasannya.

Setiap *controller* di Symfony wajib mengembalikan *Response object*.

## II. *Routing* pada Symfony

Untuk konfigurasi *routing* di Symfony ada empat cara yaitu menggunakan\*\* `php`, `yml`, `xml` dan *annotation*\*. Namun pada buku ini hanya akan dibahas \**routing* menggunakan *annotation* sesuai dengan *best practice*<sup>22</sup> Symfony.

Kembali pada contoh diatas, pada baris *code* berikut:

---

<sup>22</sup>[http://symfony.com/doc/current/best\\_practices/controllers.html](http://symfony.com/doc/current/best_practices/controllers.html)

```

1      /**
2      * @Route("/", name="homepage")
3      */
    
```

Pada *code* diatas, adalah definisi dari *route* pada Symfony. *Route* diatas adalah / (*root* aplikasi) dengan nama *homepage*.

Untuk definisi dari *route*, sebenarnya dimulai dari *routing.yml* yang ada di *folder* *app/config* sebagai berikut:

```

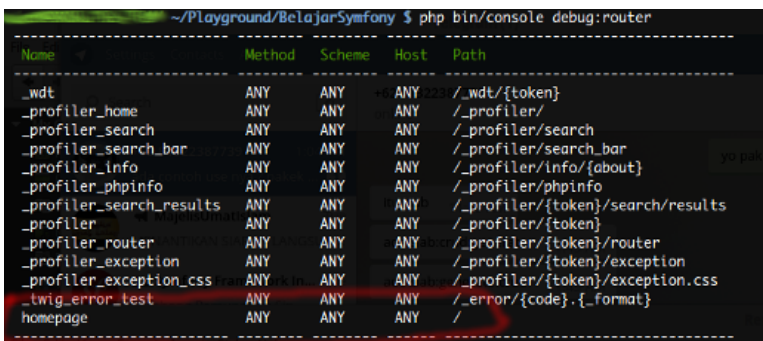
1  app:
2      resource: "@AppBundle/Controller/"
3      type:     annotation
    
```

Pada *code* diatas terlihat bahwa *routing* yang kita pakai adalah *annotation* dengan *controller* merujuk pada *folder* *@AppBundle/Controller* yaitu *folder* *src/AppBundle/Controller*.

Untuk mengetahui daftar *route* yang terregistrasi di Symfony, kita dapat menjalankan perintah berikut:

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

Maka akan muncul tampilan sebagai berikut:



Name	Method	Scheme	Host	Path
_wdt	ANY	ANY	ANY	/{token}
_profiler_home	ANY	ANY	ANY	/_profiler/
_profiler_search	ANY	ANY	ANY	/_profiler/search
_profiler_search_bar	ANY	ANY	ANY	/_profiler/search_bar
_profiler_info	ANY	ANY	ANY	/_profiler/info/{about}
_profiler_phpinfo	ANY	ANY	ANY	/_profiler/phpinfo
_profiler_search_results	ANY	ANY	ANY	/_profiler/{token}/search/results
_profiler	ANY	ANY	ANY	/_profiler/{token}
_profiler_router	ANY	ANY	ANY	/_profiler/{token}/router
_profiler_exception	ANY	ANY	ANY	/_profiler/{token}/exception
_profiler_exception_css	ANY	ANY	ANY	/_profiler/{token}/exception.css
_twig_error_test	ANY	ANY	ANY	/_error/{code}.{_format}
homepage	ANY	ANY	ANY	/

### Route

Kita juga bisa menambahkan *prefix* pada *routing* sebagai berikut:

```

1 app:
2     resource: "@AppBundle/Controller/"
3     type:     annotation
4     prefix:   /admin
    
```

Maka secara otomatis, semua *routing* pada `@AppBundle/Controller` akan dimulai dengan *prefix* `/admin`. Dan bila kita menjalankan perintah `debug:router` maka tampilannya akan menjadi seperti ini:

~/.Playground/BelajarSymfony \$ php bin/console debug:router

Name	Method	Scheme	Host	Path
_wdt	ANY	ANY	ANY	/_wdt/{token}
_profiler_home	ANY	ANY	ANY	/_profiler/
_profiler_search	ANY	ANY	ANY	/_profiler/search
_profiler_search_bar	ANY	ANY	ANY	/_profiler/search_bar
_profiler_info	ANY	ANY	ANY	/_profiler/info/{about}
_profiler_phpinfo	ANY	ANY	ANY	/_profiler/phpinfo
_profiler_search_results	ANY	ANY	ANY	/_profiler/{token}/search/results
_profiler	ANY	ANY	ANY	/_profiler/{token}
_profiler_router	ANY	ANY	ANY	/_profiler/{token}/router
_profiler_exception	ANY	ANY	ANY	/_profiler/{token}/exception
_profiler_exception_css	ANY	ANY	ANY	/_profiler/{token}/exception.css
_twig_error_test	ANY	ANY	ANY	/_error/{code}.{_format}
homepage	ANY	ANY	ANY	/

~/.Playground/BelajarSymfony \$ php bin/console debug:router

Name	Method	Scheme	Host	Path
_wdt	ANY	ANY	ANY	/_wdt/{token}
_profiler_home	ANY	ANY	ANY	/_profiler/
_profiler_search	ANY	ANY	ANY	/_profiler/search
_profiler_search_bar	ANY	ANY	ANY	/_profiler/search_bar
_profiler_info	ANY	ANY	ANY	/_profiler/info/{about}
_profiler_phpinfo	ANY	ANY	ANY	/_profiler/phpinfo
_profiler_search_results	ANY	ANY	ANY	/_profiler/{token}/search/results
_profiler	ANY	ANY	ANY	/_profiler/{token}
_profiler_router	ANY	ANY	ANY	/_profiler/{token}/router
_profiler_exception	ANY	ANY	ANY	/_profiler/{token}/exception
_profiler_exception_css	ANY	ANY	ANY	/_profiler/{token}/exception.css
_twig_error_test	ANY	ANY	ANY	/_error/{code}.{_format}
homepage	ANY	ANY	ANY	/admin/

Route Prefix

### III. *Route Parameter*

Untuk mendefinisikan *route parameter* di Symfony sangatlah mudah, Anda cukup mendefinisikan sebagai berikut:

```

1      /**
2      * @Route("/{id}", name="homepage")
3      */
4      public function detailAction(Request $request, $id)
5      {
6          //do your logic
7      }
```

Penamaan untuk *route parameter* dan *method parameter* harus sama. Pada contoh diatas *route parameter* adalah {id} maka *method parameter*-nya harus sama yaitu \$id.

Sedangkan untuk *parameter* \$request adalah *optional*. Jika tidak didefinisikan maka *parameter* pertama harus diisi *route parameter*. Sehingga *code*-nya akan menjadi seperti berikut:

```

1      /**
2      * @Route("/{id}", name="homepage")
3      */
4      public function detailAction($id)
5      {
6          //do your logic
7      }
```





## Route Annotation

*Route annotation* pada Symfony berasal dari *class* `Sensio\Bundle\FrameworkExtraBundle\Configuration\Route` sehingga untuk menggunakan annotation `@Route` maka Anda harus terlebih dahulu meng-*import class* tersebut menggunakan *keyword use*

`@Route` juga bisa kita taruh pada *class Controller* sebagai *prefix route* untuk *Controller* tersebut.

Untuk lebih lengkap tentang *routing* dapat membaca [Dokumentasi](#)<sup>23</sup>

## IV. HTTP Method pada Symfony

Secara *default*, *route* di Symfony dapat diakses baik dengan method GET, POST, PUT, DELETE, ataupun PATCH. Untuk membatasi akses tersebut, maka kita harus mendefinisikan *method* tersebut secara spesifik pada *route* kita.

Untuk mendefinisikan *HTTP Method*, kita dapat menggunakan *annotation* `@Method` yang berasal dari `Sensio\Bundle\FrameworkExtraBundle\Configuration` sehingga *controller* kita akan menjadi seperti berikut:

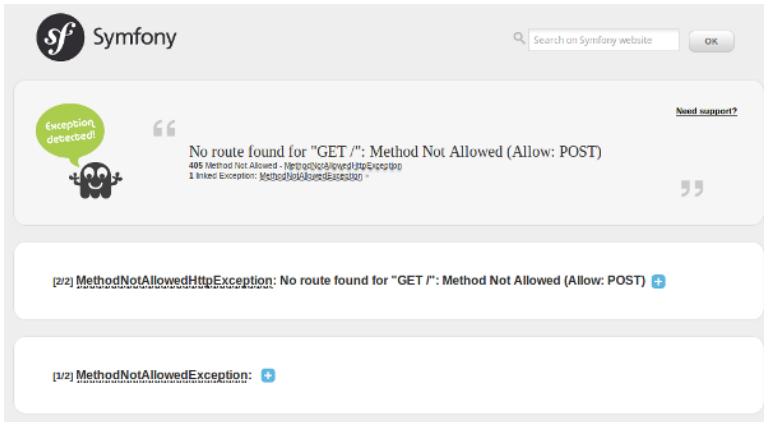
```

1  <?php
2
3  namespace AppBundle\Controller;
4
5  use Sensio\Bundle\FrameworkExtraBundle\Configuration\Me\
6  thod;
7  use Sensio\Bundle\FrameworkExtraBundle\Configuration\Ro\
8  ute;
9  use Symfony\Bundle\FrameworkBundle\Controller\Controller\
    
```

<sup>23</sup><http://symfony.com/doc/current/book/routing.html>

```
10 r;
11 use Symfony\Component\HttpFoundation\Request;
12
13 class DefaultController extends Controller
14 {
15     /**
16      * @Route("/", name="homepage")
17      * @Method("POST")
18      */
19     public function indexAction(Request $request)
20     {
21         // replace this example code with whatever you \
22 need
23         return $this->render('default/index.html.twig',\
24 [
25     'base_dir' => realpath($this->getParameter(\
26 'kernel.root_dir').'/..'),
27     ]);
28     }
29 }
```

Pada *code* diatas, kita hanya membolehkan *method* POST pada *route* homepage sehingga ketika kita mengaksesnya menggunakan *browser* akan muncul pesan *error* bahwa *method* GET tidak diperbolehkan sebagai berikut:



### Method Not Allowed

Kita juga dapat mendefinisikan lebih dari satu *method* pada *route* sebagai berikut:

```

1  <?php
2
3  namespace AppBundle\Controller;
4
5  use Sensio\Bundle\FrameworkExtraBundle\Configuration\Me\
6  thod;
7  use Sensio\Bundle\FrameworkExtraBundle\Configuration\Ro\
8  ute;
9  use Symfony\Bundle\FrameworkBundle\Controller\Controll
10 e;
11 use Symfony\Component\HttpFoundation\Request;
12
13 class DefaultController extends Controller
14 {
15     /**
16      * @Route("/", name="homepage")
17      * @Method({"POST", "GET"})
18      */
    
```

```

19     public function indexAction(Request $request)
20     {
21         // replace this example code with whatever you \
22 need
23         return $this->render('default/index.html.twig', \
24     [
25         'base_dir' => realpath($this->getParameter(\
26 'kernel.root_dir').'../'),
27     ]);
28     }
29 }
```

Dan karena kita menambahkan *method* GET pada route homepage maka ketika kita mengakses halaman homepage, kita tidak akan mendapat pesan *error* lagi.



## Selalu Definisikan *Method*

Untuk keamanan aplikasi, sebaiknya selalu mendefinisikan *method* pada setiap *route* untuk membatasi akses bagi *client*

# Membuat Aplikasi Kontak Sederhana

## I. Pengantar

Setelah kita belajar tentang apa *Controller*, *Routing*, *Entity*, *Form* serta *View*, maka sekarang saatnya untuk mengintegrasikan semuanya menjadi sebuah aplikasi sederhana yang bermanfaat. Sebagai studi kasus, kita akan membuat sebuah aplikasi Kontak sederhana.

Pada aplikasi ini kita hanya punya satu tabel saja yaitu tabel kontak tanpa ada relasi dengan tabel lain sama sekali. Hal ini bertujuan agar kita lebih memahami apa yang telah kita pelajari terlebih dahulu.

## II. Pembuatan *Entity*

Pada aplikasi Kontak Sederhana ini, kita akan membuat sebuah *entity* bernama *Contact* dengan *fields* *id*, *name*, *phoneNumber* dan *email* sebagai berikut:

```
1  <?php
2
3  //filename: AppBundle\Entity/Contact.php
4
5  namespace AppBundle\Entity;
6
7  use Doctrine\ORM\Mapping as ORM;
8
9  /**
10   * @ORM\Entity
11   * @ORM\Table(name="contacts")
12   */
13  class Contact
14  {
15      /**
16       * @ORM\Id
17       * @ORM\Column(name="id", type="integer")
18       * @ORM\GeneratedValue(strategy="AUTO")
19       */
20      private $id;
21
22      /**
23       * @ORM\Column(name="name", type="string", length=7\
24  7)
25       */
26      private $name;
27
28      /**
29       * @ORM\Column(name="phone_number", type="string", \
30  length=17)
31       */
32      private $phoneNumber;
33
34      /**
35       * @ORM\Column(name="email", type="string", length=\
```

```
36 255)
37     */
38     private $email;
39 }
```

Selanjutnya kita akan membuat *setter* dan *getter method* dengan menjalankan perintah `php bin/console doctrine:generate:entities AppBundle:Contact` sehingga hasilnya akan menjadi seperti ini:

```
1  <?php
2
3  //filename: AppBundle/Entity/Contact.php
4
5  namespace AppBundle\Entity;
6
7  use Doctrine\ORM\Mapping as ORM;
8
9  /**
10   * @ORM\Entity
11   * @ORM\Table(name="contacts")
12   */
13  class Contact
14  {
15      /**
16       * @ORM\Id
17       * @ORM\Column(name="id", type="integer")
18       * @ORM\GeneratedValue(strategy="AUTO")
19       */
20      private $id;
21
22      /**
23       * @ORM\Column(name="name", type="string", length=7\
24  7)
25       */
```

```
26     private $name;
27
28     /**
29      * @ORM\Column(name="phone_number", type="string", \
30 length=17)
31      */
32     private $phoneNumber;
33
34     /**
35      * @ORM\Column(name="email", type="string", length=\
36 255)
37      */
38     private $email;
39
40     /**
41      * Get id
42      *
43      * @return integer
44      */
45     public function getId()
46     {
47         return $this->id;
48     }
49
50     /**
51      * Set name
52      *
53      * @param string $name
54      *
55      * @return Contact
56      */
57     public function setName($name)
58     {
59         $this->name = $name;
60
```



```
61         return $this;
62     }
63
64     /**
65      * Get name
66      *
67      * @return string
68      */
69     public function getName()
70     {
71         return $this->name;
72     }
73
74     /**
75      * Set phoneNumber
76      *
77      * @param string $phoneNumber
78      *
79      * @return Contact
80      */
81     public function setPhoneNumber($phoneNumber)
82     {
83         $this->phoneNumber = $phoneNumber;
84
85         return $this;
86     }
87
88     /**
89      * Get phoneNumber
90      *
91      * @return string
92      */
93     public function getPhoneNumber()
94     {
95         return $this->phoneNumber;
```

```
96     }
97
98     /**
99      * Set email
100     *
101     * @param string $email
102     *
103     * @return Contact
104     */
105     public function setEmail($email)
106     {
107         $this->email = $email;
108
109         return $this;
110     }
111
112     /**
113     * Get email
114     *
115     * @return string
116     */
117     public function getEmail()
118     {
119         return $this->email;
120     }
121 }
```

Setelah men-*generate setter* dan *getter method*, kita akan membuat tabel pada *database* dengan menjalankan perintah `php bin/console doctrine:schema:update --force`.

Sampai disini, kita telah selesai untuk membuat *entity* lengkap dengan tabel *database*-nya.

### III. Pembuatan Form

Untuk *form* yang akan kita gunakan untuk memanipulasi *entity* Contact, kita akan membuatnya seperti berikut:

```
1  <?php
2
3  //filename: AppBundle/Form/Type/ContactType.php
4
5  namespace AppBundle\Form\Type;
6
7  use AppBundle\Entity\Contact;
8  use Symfony\Component\Form\AbstractType;
9  use Symfony\Component\Form\Extension\Core\Type\EmailType\
10 e;
11 use Symfony\Component\Form\Extension\Core\Type\TextType;
12 use Symfony\Component\Form\FormBuilderInterface;
13 use Symfony\Component\OptionsResolver\OptionsResolver;
14
15 class ContactType extends AbstractType
16 {
17     /**
18      * @param FormBuilderInterface $builder
19      * @param array $options
20      */
21     public function buildForm(FormBuilderInterface $bui\
22 lder, array $options)
23     {
24         $builder
25             ->add('name', TextType::class)
26             ->add('phoneNumber', TextType::class)
27             ->add('email', EmailType::class)
28         ;
29     }
```

```
30
31     /**
32      * @param OptionsResolver $resolver
33      */
34     public function configureOptions(OptionsResolver $r\
35 esolver)
36     {
37         $resolver->setDefaults(array(
38             'data_class' => Contact::class
39         ));
40     }
41 }
```

Pada pembahasan kali ini, kita menggunakan satu *form type* baru yaitu `EmailType`. `EmailType` adalah *form type* untuk didalamnya telah di-*built-in* *HTML5 email validator*, sehingga untuk penggunaan normal, kita tidak perlu melakukan pengecekan *input*-an sama sekali.

## IV. Pembuatan Controller

*Controller* yang akan kita gunakan untuk memanipulasi *entity* `Contact` adalah sebagai berikut:

```
1  <?php
2
3  //filename: AppBundle/Controller/ContactController.php
4
5  namespace AppBundle\Controller;
6
7  use Symfony\Component\HttpFoundation\Request;
8  use Symfony\Bundle\FrameworkBundle\Controller\Controller;
9  r;
```

```
10 use Sensio\Bundle\FrameworkExtraBundle\Configuration\Me\
11 thod;
12 use Sensio\Bundle\FrameworkExtraBundle\Configuration\Ro\
13 ute;
14 use AppBundle\Entity\Contact;
15 use AppBundle\Form\Type\ContactType;
16 use Symfony\Component\HttpFoundation\Exception\NotFoundHttp\
17 Exception;
18
19 /**
20  * Contact controller.
21  *
22  * @Route("/contact")
23  */
24 class ContactController extends Controller
25 {
26     /**
27      * Lists all Contact entities.
28      *
29      * @Route("/", name="contact_index")
30      * @Method("GET")
31      */
32     public function indexAction()
33     {
34         $em = $this->getDoctrine()->getManager();
35
36         $contacts = $em->getRepository('AppBundle:Conta\
37 ct')->findAll();
38
39         return $this->render('AppBundle:contact:index.h\
40 tml.twig', array(
41             'contacts' => $contacts,
42         ));
43     }
44 }
```

```
45      /**
46       * Creates a new Contact entity.
47       *
48       * @Route("/new", name="contact_new")
49       * @Method({"GET", "POST"})
50       */
51     public function newAction(Request $request)
52     {
53         $contact = new Contact();
54         $form = $this->createForm(ContactType::class, $contact);
55         $form->handleRequest($request);
56
57         if ($form->isSubmitted() && $form->isValid()) {
58             $em = $this->getDoctrine()->getManager();
59             $em->persist($contact);
60             $em->flush();
61
62             return $this->redirectToRoute('contact_show',
63                 array('id' => $contact->getId()));
64         }
65
66         return $this->render('AppBundle:contact:new.html.twig', array(
67             'form' => $form->createView(),
68         ));
69     }
70
71     /**
72     * Finds and displays a Contact entity.
73     *
74     * @Route("/{id}", name="contact_show")
75     * @Method("GET")
76     */
77     public function showAction($id)
```

```
80     {
81         $contact = $this->getDoctrine()->getRepository(\
82 'AppBundle:Contact')->find($id);
83         if (!$contact) {
84             throw new NotFoundHttpException(sprintf('Ko\
85 ntak dengan id %d tidak ditemukan', $id));
86         }
87
88         $deleteForm = $this->createDeleteForm($contact);
89
90         return $this->render('AppBundle:contact:show.ht\
91 ml.twig', array(
92             'contact' => $contact,
93             'delete_form' => $deleteForm->createView(),
94         ));
95     }
96
97     /**
98      * Displays a form to edit an existing Contact enti\
99 ty.
100
101      *
102      * @Route("/{id}/edit", name="contact_edit")
103      * @Method({"GET", "POST"})
104      */
105     public function editAction(Request $request, $id)
106     {
107         $contact = $this->getDoctrine()->getRepository(\
108 'AppBundle:Contact')->find($id);
109         if (!$contact) {
110             throw new NotFoundHttpException(sprintf('Ko\
111 ntak dengan id %d tidak ditemukan', $id));
112         }
113
114         $deleteForm = $this->createDeleteForm($contact);
115         $editForm = $this->createForm(ContactType::clas\
```

```
115 s, $contact);
116     $editForm->handleRequest($request);
117
118     if ($editForm->isSubmitted() && $editForm->isValid\
119 lid()) {
120         $em = $this->getDoctrine()->getManager();
121         $em->persist($contact);
122         $em->flush();
123
124         return $this->redirectToRoute('contact_edit\
125 ', array('id' => $contact->getId()));
126     }
127
128     return $this->render('AppBundle:contact:edit.ht\
129 ml.twig', array(
130         'edit_form' => $editForm->createView(),
131         'delete_form' => $deleteForm->createView(),
132     ));
133 }
134
135 /**
136  * Deletes a Contact entity.
137  *
138  * @Route("/{id}", name="contact_delete")
139  * @Method("DELETE")
140  */
141 public function deleteAction(Request $request, $id)
142 {
143     $contact = $this->getDoctrine()->getRepository(\
144 'AppBundle:Contact')->find($id);
145     if (!$contact) {
146         throw new NotFoundException(sprintf('Ko\
147 ntak dengan id %d tidak ditemukan', $id));
148     }
149 }
```



```

150         $form = $this->createDeleteForm($contact);
151         $form->handleRequest($request);
152
153         if ($form->isSubmitted() && $form->isValid()) {
154             $em = $this->getDoctrine()->getManager();
155             $em->remove($contact);
156             $em->flush();
157         }
158
159         return $this->redirectToRoute('contact_index');
160     }
161
162     /**
163      * Creates a form to delete a Contact entity.
164      *
165      * @param Contact $contact The Contact entity
166      *
167      * @return \Symfony\Component\Form\Form The form
168      */
169     private function createDeleteForm(Contact $contact)
170     {
171         return $this->createFormBuilder()
172             ->setAction($this->generateUrl('contact_delete', array('id' => $contact->getId())))
173             ->setMethod('DELETE')
174             ->getForm()
175         ;
176     }
177 }
178 }

```

Penjelasan dari setiap *method* dalam *controller* adalah sebagai berikut:

#### a. indexAction

```

1      public function indexAction()
2      {
3          $em = $this->getDoctrine()->getManager();
4
5          $contacts = $em->getRepository('AppBundle:Conta\
6 ct')->findAll();
7
8          return $this->render('AppBundle:contact:index.h\
9 tml.twig', array(
10             'contacts' => $contacts,
11             ));
12     }

```

- Pada baris `$em = $this->getDoctrine()->getManager();`, kita memanggil `EntityManager`.
- Pada baris `$contacts = $em->getRepository('AppBundle:Contact')->findAll();`, kita mengambil semua data kontak dari `Contact repository`.
- Pada baris selanjutnya, kita *me-render view* `AppBundle:contact:index.html` dan memasukkan `$contacts` dalam *view* sebagai `contacts`.

**\*\* b. newAction \*\***

```

1      public function newAction(Request $request)
2      {
3          $contact = new Contact();
4          $form = $this->createForm(ContactType::class, $\
5 contact);
6          $form->handleRequest($request);
7
8          if ($form->isSubmitted() && $form->isValid()) {
9              $em = $this->getDoctrine()->getManager();
10             $em->persist($contact);

```

```
11         $em->flush();
12
13         return $this->redirectToRoute('contact_show\
14 ', array('id' => $contact->getId()));
15     }
16
17     return $this->render('AppBundle:contact:new.htm\
18 l.twig', array(
19         'form' => $form->createView(),
20     ));
21 }
```

- Pada baris `$contact = new Contact()`, kita membuat *object* baru untuk kontak.
- Pada baris `$form = $this->createForm(ContactType::class, $contact)`, kita membuat *object form* dan memasukkan `$contact` sebagai *default* data-nya
- Pada baris `$form->handleRequest($request)`, *form* meng-*handle request* yang masuk dari *client*. Disini, request yang masuk dari client di-*mapping* sesuai data\_class pada *form* `ContactType`.
- Pada baris `if ($form->isSubmitted() && $form->isValid())`, kita mengecek apakah *form* tersebut di-*submit* (*client* mengirim menggunakan *method* POST) dan apakah *form* tersebut *valid* inputannya.
- Tiga baris selanjutnya, saya anggap Anda sudah paham. Jika belum, silahkan baca kembali bab sebelumnya tentang Doctrine
- Pada baris `return $this->redirectToRoute('contact_show', array('id' => $contact->getId()))`, jika proses *insert* ke *database* berhasil maka *client* akan di-*redirect* ke *route* dengan nama `contact_show`
- Baris terakhir, secara *default* (*client* tidak melakukan *submit* data atau data yang di-*input* tidak valid), maka akan

menampilkan `view AppBundle:contact:new.html.twig` yang berisi *form*.

### c. `showAction`

```

1      public function showAction($id)
2      {
3          $contact = $this->getDoctrine()->getRepository(\
4      'AppBundle:Contact')->find($id);
5          if (!$contact) {
6              throw new NotFoundHttpException(sprintf('Kontak
7      ntak dengan id %d tidak ditemukan', $id));
8          }
9
10         $deleteForm = $this->createDeleteForm($contact);
11
12         return $this->render('AppBundle:contact:show.html.twig', array(
13             'contact' => $contact,
14             'delete_form' => $deleteForm->createView(),
15         ));
16     }
17 
```

- Pada baris `$contact = $this->getDoctrine()->getRepository('AppBundle:Contact')->find($id)`, kita mencoba mengambil data kontak berdasarkan `$id`
- Pada baris `throw new NotFoundHttpException(sprintf('Kontak dengan id %d tidak ditemukan', $id))`, akan menampilkan pesan *error* jika kontak dengan id `$id` tidak ditemukan.
- Pada baris `$deleteForm = $this->createDeleteForm($contact)`, kita membuat form untuk keperluan operasi delete. Sebuah *form* sederhana yang berisi *route* untuk operasi *delete*.

### d. `editAction`

```
1     public function editAction(Request $request, $id)
2     {
3         $contact = $this->getDoctrine()->getRepository(\
4 'AppBundle:Contact')->find($id);
5         if (!$contact) {
6             throw new NotFoundHttpException(sprintf('Ko\
7 ntak dengan id %d tidak ditemukan', $id));
8         }
9
10        $deleteForm = $this->createDeleteForm($contact);
11        $editForm = $this->createForm(ContactType::clas\
12 s, $contact);
13        $editForm->handleRequest($request);
14
15        if ($editForm->isSubmitted() && $editForm->isVa\
16 lid()) {
17            $em = $this->getDoctrine()->getManager();
18            $em->persist($contact);
19            $em->flush();
20
21            return $this->redirectToRoute('contact_edit\
22 ', array('id' => $contact->getId()));
23        }
24
25        return $this->render('AppBundle:contact:edit.ht\
26 ml.twig', array(
27            'edit_form' => $editForm->createView(),
28            'delete_form' => $deleteForm->createView(),
29        ));
30    }
```

Saya anggap Anda sudah mengerti maksud dari baris *code* diatas. Karena semua penjelasannya sudah ada pada penjelasan sebelumnya.

**\*\* e. deleteAction \*\***

```
1      public function deleteAction(Request $request, $id)
2      {
3          $contact = $this->getDoctrine()->getRepository(\
4      'AppBundle:Contact')->find($id);
5          if (!$contact) {
6              throw new NotFoundHttpException(sprintf('Ko\
7      ntak dengan id %d tidak ditemukan', $id));
8          }
9
10         $form = $this->createDeleteForm($contact);
11         $form->handleRequest($request);
12
13         if ($form->isSubmitted() && $form->isValid()) {
14             $em = $this->getDoctrine()->getManager();
15             $em->remove($contact);
16             $em->flush();
17         }
18
19         return $this->redirectToRoute('contact_index');
20     }
```

Pada code diatas pun sama, saya anggap Anda sudah paham.

## V. Pembuatan View

**a . new.html.twig**

```

1  {% extends 'base.html.twig' %}
2
3  {% block body %}
4      <h1>Contact creation</h1>
5
6      {{ form_start(form) }}
7          {{ form_widget(form) }}
8          <input type="submit" value="Create" />
9      {{ form_end(form) }}
10
11     <ul>
12         <li>
13             <a href="{{ path('contact_index') }}">Back \
14 to the list</a>
15         </li>
16     </ul>
17 {% endblock %}

```

- Pada baris `{% extends 'base.html.twig' %}`, kita meng-*extends* template `base.html.twig` yang ada di `app/Resources/views`.
- Pada baris `{% block body %}` hingga `{% endblock %}`, kita mendefinisikan ulang block **body** yang ada di `base.html.twig`.
- Pada baris `{{ path('contact_index') }}`, kita men-*generate* url untuk *route* dengan nama `contact_index`

#### b. edit.html.twig

```
1  {% extends 'base.html.twig' %}
2
3  {% block body %}
4      <h1>Contact edit</h1>
5
6      {{ form_start(edit_form) }}
7          {{ form_widget(edit_form) }}
8          <input type="submit" value="Edit" />
9      {{ form_end(edit_form) }}
10
11     <ul>
12         <li>
13             <a href="{{ path('contact_index') }}">Back \
14 to the list</a>
15         </li>
16         <li>
17             {{ form_start(delete_form) }}
18                 <input type="submit" value="Delete">
19             {{ form_end(delete_form) }}
20         </li>
21     </ul>
22 {% endblock %}
```

Saya anggap Anda sudah paham.

### c. show.html.twig



```
1  {% extends 'base.html.twig' %}
2
3  {% block body %}
4      <h1>Contact</h1>
5
6      <table>
7          <tbody>
8              <tr>
9                  <th>Id</th>
10                 <td>{{ contact.id }}</td>
11             </tr>
12             <tr>
13                 <th>Name</th>
14                 <td>{{ contact.name }}</td>
15             </tr>
16             <tr>
17                 <th>Phonenumber</th>
18                 <td>{{ contact.phoneNumber }}</td>
19             </tr>
20             <tr>
21                 <th>Email</th>
22                 <td>{{ contact.email }}</td>
23             </tr>
24         </tbody>
25     </table>
26
27     <ul>
28         <li>
29             <a href="{{ path('contact_index') }}">Back \
30 to the list</a>
31         </li>
32         <li>
33             <a href="{{ path('contact_edit', { 'id': co\
34 ntact.id }) }}">Edit</a>
35         </li>
```

```

36         <li>
37             {{ form_start(delete_form) }}
38             <input type="submit" value="Delete">
39             {{ form_end(delete_form) }}
40         </li>
41     </ul>
42 {% endblock %}

```

- Pada baris `{{ contact.id }}`, sama saja dengan `$contact->getId()` karena pada twig notasi dot (.) dapat bermakna *array key* atau *method calling*. Namun agar tidak terjadi kebingungan, sebaiknya untuk pemanggilan *array* menggunakan cara seperti pada PHP yaitu dengan `$array['index']` atau jika dalam twig menjadi `{{ array['index'] }}`.
- Pada baris `{{ path('contact_edit', { 'id': contact.id }) }}`, terutama pada `{ 'id': contact.id }`. Ini adalah cara *mem-passing route param* pada twig, tidak jauh berbeda dengan *mem-passing route param* pada *controller*.

**\*\*d. index.html.twig**

```

1  {% extends 'base.html.twig' %}
2
3  {% block body %}
4      <h1>Contact list</h1>
5
6      <table>
7          <thead>
8              <tr>
9                  <th>Id</th>
10                 <th>Name</th>
11                 <th>Phonenumber</th>
12                 <th>Email</th>

```

```

13         <th>Actions</th>
14     </tr>
15 </thead>
16 <tbody>
17     {% for contact in contacts %}
18     <tr>
19         <td><a href="{{ path('contact_show', { \
20 'id': contact.id }) }}">{{ contact.id }}</a></td>
21         <td>{{ contact.name }}</td>
22         <td>{{ contact.phoneNumber }}</td>
23         <td>{{ contact.email }}</td>
24         <td>
25             <ul>
26                 <li>
27                     <a href="{{ path('contact_s\
28 how', { 'id': contact.id }) }}">show</a>
29                 </li>
30                 <li>
31                     <a href="{{ path('contact_e\
32 dit', { 'id': contact.id }) }}">edit</a>
33                 </li>
34             </ul>
35         </td>
36     </tr>
37     {% endfor %}
38 </tbody>
39 </table>
40
41 <ul>
42     <li>
43         <a href="{{ path('contact_new') }}">Create \
44 a new entry</a>
45     </li>
46 </ul>
47 {% endblock %}

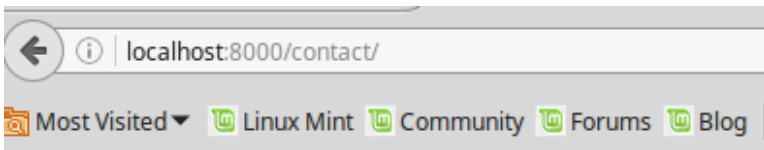
```

- Pada baris `{% for contact in contacts %}` hingga `{% endfor %}`, ini adalah cara looping di twig.

## VI. Menjalankan Aplikasi

Setelah semuanya selesai, maka sekarang saatnya kita mencoba untuk menjalankan aplikasi kita. Setelah kita menjalankan *web server* dengan mengetikan perintah `php bin/console server:run`, selanjutnya kita mengetikan di *browser* alamat `localhost:8000/contact`.

Bila tidak ada *error*, Anda akan medapati halaman sebagai berikut:



# Contact list

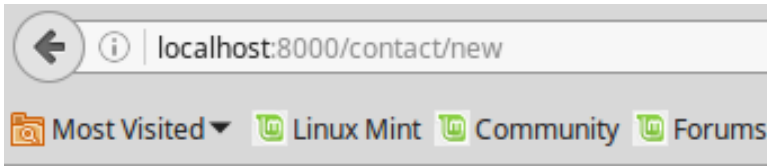
	<b>Id</b>	<b>Name</b>	<b>Phonenumber</b>	<b>Email</b>	<b>Actions</b>
--	-----------	-------------	--------------------	--------------	----------------

	<u>2</u>	Surya	087800093915	a@b.c	<ul style="list-style-type: none"><li>• <a href="#">show</a></li><li>• <a href="#">edit</a></li></ul>
--	----------	-------	--------------	-------	---

- [Create a new entry](#)

List Kontak

Kemudian jika kita mengeklik link **Create a new entry** maka akan tampil sebagai berikut:



## Contact creation

Name

Phone number

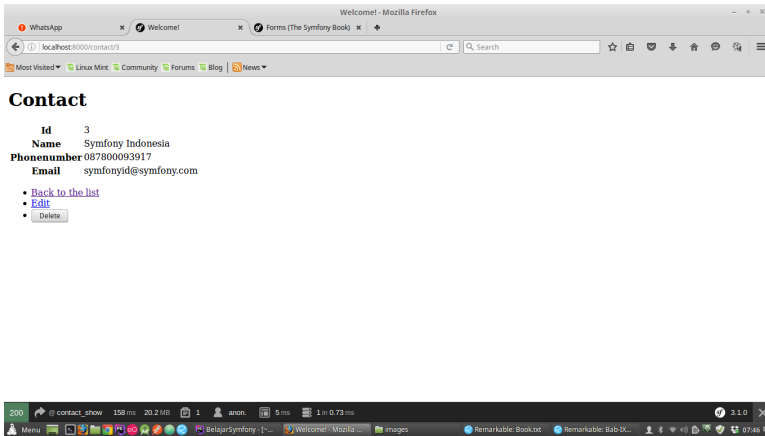
Email

- [Back to the list](#)

### Kontak Baru

Pada gambar terlihat jika saya memasukkan alamat *email* tidak sesuai dengan format *email* yang benar maka akan muncul *error* dan *form* tidak dapat di *submit*.

Setelah kita *input* data dengan benar, dan kita *submit*, maka akan muncul tampilan berikut:



### Kontak Disimpan

Kita bisa meneruskan percobaan tersebut dan mengetes semua fungsi yang telah kita buat.

## VII. Kesimpulan

Untuk membuat operasi *CRUD* sederhana dengan Symfony sangatlah mudah dan cepat. Di Symfony kita diajarkan untuk fokus pada *code* yang kita bangun, sedangkan *database* diserahkan *management*-nya kepada Doctrine sehingga kita tidak perlu membuat *database* dan merancang tabel seperti yang biasa kita lakukan.

Untuk membuat operasi *CRUD* sederhana diatas, Doctrine sebenarnya dapat membuatnya dengan sangat mudah. Anda cukup menjalankan perintah `php bin/console doctrine:generate:crud AppBundle:Contact` maka Anda akan dibuatkan *Form*, *Controller* dan *View* lengkap dengan *Unit Testing*-nya oleh Doctrine.

Saya tidak memperkenalkan perintah diatas, agar Anda lebih memahami *code* yang Anda tulis terlebih dahulu.

# Menambahkan Group pada Aplikasi Kontak

## I. Membuat CRUD untuk Group

Untuk membuat CRUD Group, kita akan mencoba menggunakan pendekatan yang berbeda dengan sebelumnya. Kita akan mencoba menggunakan *CRUD Generator* dari Symfony.

Seperti yang sudah disinggung pada bab sebelumnya, bahwa di Symfony telah ada *CRUD Generator*, namun saya sendiri tidak menggunakan *CRUD Generator* tersebut untuk keperluan *project* saya. Namun bila Anda dikejar *deadline* dan ingin membangun aplikasi secara *instant*, mungkin Anda layak untuk mencobanya.

Untuk menggunakan fitur tersebut, kita cukup menjalankan perintah berikut:

```
1 php bin/console doctrine:generate:crud AppBundle:Group \  
2 --overwrite
```

Kemudian ikut dan jawab pertanyaan singkatnya, maka CRUD untuk *entity* Group pun telah selesai dibuat. Kita menambahkan *option --overwrite* karena pada pembahasan sebelumnya, kita telah memiliki *controller* dengan nama *GroupController* sehingga perlu di-*overwrite*.

## II. Limitasi

Perlu Anda pahami, bahwa dengan menggunakan *CRUD Generator* maka harus tahu tentang keterbatasan *CRUD Generator* tersebut.

Bila kita menggunakan *CRUD Generator* maka kita akan menemui perbedaan letak *folder* dengan *CRUD* yang kita buat sebelumnya secara manual.

Bila secara manual *view* yang kita buat disimpan dalam folder `AppBundle/Resources/views` maka dengan menggunakan *generator*, *view* kita akan disimpan pada *folder* `app/Resources/views`.

Selain itu, *form* kita yang sebelumnya kita simpan dalam *folder* `AppBundle/Form/Type`, jika menggunakan *generator* maka *form* akan disimpan pada *folder* `AppBundle/Form`.

Selain letak *form*-nya berbeda, perbedaan lain pada *form* juga terdapat pada pendefinisian *form field* dimana jika menggunakan *generator* maka pendefinisian *form field* menggunakan *type guesser* seperti berikut:

```
1         $builder
2             ->add('name')
3         ;
```

Dengan cara seperti diatas, *Symfony* secara otomatis akan mengambil *field type* dari *Doctrine mapping*. Dan karena pada *mapping field name* bertipe *string* seperti berikut:

```
1 @ORM\Column(name="name", type="string", length=77)
```

Maka secara otomatis, *Symfony* akan menganggapnya sebagai *TextType*.

### III. Meng-update *form* dan *view* Contact

Langkah terakhir untuk menyempurnakan aplikasi Kontak Sederhana kita, kita perlu untuk menambahkan *Group* kedalam *form* dan juga *view* *Contact* kita.

Pada *form* kita cukup menambahkan *field group* sebagai berikut:



```

1      public function buildForm(FormBuilderInterface $builder\
2      lder, array $options)
3      {
4          $builder
5              ->add('group', EntityType::class, array(
6                  'class' => Group::class,
7                  'choice_label' => 'name',
8              ))
9              ->add('name', TextType::class)
10             ->add('phoneNumber', TextType::class)
11             ->add('email', EmailType::class)
12         ;
13     }

```

Dan pada *view* kita cukup memanggil group seperti berikut:

a. Pada `index.html.twig`

```

1      <td><a href="{{ path('contact_show', { \
2      'id': contact.id }) }}">{{ contact.id }}</a></td>
3      <td>{{ contact.group.name }}</td>
4      <td>{{ contact.name }}</td>
5      <td>{{ contact.phoneNumber }}</td>
6      <td>{{ contact.email }}</td>

```

a. Pada `show.html.twig`

```
1         <tr>
2             <th>Id</th>
3             <td>{{ contact.id }}</td>
4         </tr>
5         <tr>
6             <th>Group</th>
7             <td>{{ contact.group.name }}</td>
8         </tr>
9         <tr>
10            <th>Name</th>
11            <td>{{ contact.name }}</td>
12        </tr>
```

## IV. Menjalankan Aplikasi

Perlu Anda ketahui, karena Contact dan Group sudah terhubung, baik dari *view* maupun *form*-nya. Maka Anda harus menambahkan minimal 1 Group sebelum Anda membuka halaman *New Contact* atau *Edit Contact*. Hal tersebut perlu dilakukan karena pada *form* langsung terhubung dengan tabel Group pada *database*.