

## Day 1

### Setting up the working development environment

First of all, you need to check that your computer has a friendly working environment for web development. We will use Ubuntu 14.04 LTS Server installed in a VMware Player virtual machine. At a minimum, you need a web server (Apache, for instance), a database engine (MySQL) and PHP 5.3.3 or later.

1. Install `Apache`, your web server:

```
sudo apt-get install apache2
```

and enable Apache `mod-rewrite`:

```
sudo a2enmod rewrite
```

2, Install the `MySQL Server`:

```
sudo apt-get install mysql-server mysql-client
```

3. Install `PHP`, the server scripting language

```
sudo apt-get install php5 libapache2-mod-php5 php5-mysql
```

4. Install `Intl` extension:

```
sudo apt-get install php5-intl
```

5. Now, you need to restart Apache service

```
sudo service apache2 restart
```

### Download and install Symfony 2.3.2

The first thing to do is to prepare a directory on your web server where you want to install the new project. Let's call it `Gebeya`: `/var/www/Gebeya`.

```
mkdir /var/www/Gebeya
```

We have a directory prepared, but what to put in it? Go to <http://symfony.com/download>, choose `Symfony Standard 2.3.2` without

`vendors` and download it. Now, unzip the files inside the `Symfony` directory to your prepared directory, `Gebeya`.

## Updating Vendors

At this point, you've downloaded a fully-functional Symfony project in which you'll start to develop your own application. A Symfony project depends on a number of external libraries. These are downloaded into the `vendor/` directory of your project via a library called `Composer`.

Composer is a dependency management library for PHP, which you can use to download the Symfony 2.3.2 Standard Edition. Start by downloading Composer onto your `Gebeya` directory:

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

If you don't have `curl` extension installed, you can install it using this command:

```
apt-get install curl
```

Next, type the following command to start downloading all the necessary vendor libraries:

```
php composer.phar install
```

## Web Server Configuration

A good web practice is to put under the web root directory only the files that need to be accessed by a web browser, like stylesheets, JavaScripts and images. By default, it's recommended to store these files under the `web/` sub-directory of a symfony project.

To configure Apache for your new project, you will create a virtual host. In order to do that, go to your terminal and type in the next command :

```
sudo nano /etc/apache2/sites-available/gebeya.local
```

Now, a file named `gebeya.local` is created. Put the following inside that file, then hit Control - O and Enter to save it, then Control - X to exit the editor.

```
<VirtualHost *:80>
    ServerName gebeya.local
    DocumentRoot /var/www/Gebeya/web
    DirectoryIndex app.php
    ErrorLog /var/log/apache2/gebeya-error.log
```

```
CustomLog /var/log/apache2/gebeya-access.log combined
<Directory "/var/www/Gebeya/web">
    AllowOverride All
    Allow from All
</Directory>
</VirtualHost>
```

The domain name `gebeya.local` used in the Apache configuration has to be declared locally. If you run a Linux system, it has to be done in the `/etc/hosts` file. If you run Windows, this file is located in the `C:\Windows\System32\drivers\etc` directory. Add the following line:

```
127.0.0.1 gebeya.local
```

Replace `127.0.0.1` with the IP of your web server machine in case you are working on a remote server.

If you want this to work, you need to enable the newly created virtual host and restart your Apache. So go to your terminal and type:

```
sudo a2ensite gebeya.local
sudo service apache2 restart
```

Symfony comes with a visual server configuration tester to help make sure your Web server and PHP are correctly configured to use Symfony. Use the following URL to check your configuration:

<http://gebeya.local/config.php>

pic.

If you don't run this from your localhost, you should locate and open `web/config.php` file and comment the lines that restrict the access outside localhost:

```
if (!isset($_SERVER['HTTP_HOST'])) {
    exit('This script cannot be run from the CLI. Run it from a browser.');
```

```
}

/*
if (!in_array(@$_SERVER['REMOTE_ADDR'], array(
    '127.0.0.1',
    '::1',
))) {
```

```

    header('HTTP/1.0 403 Forbidden');
    exit('This script is only accessible from localhost.');
```

Do the same for `web/app_dev.php`:

```

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Debug\Debug;

// If you don't want to setup permissions the proper way, just uncomment the following PHP line
// read http://symfony.com/doc/current/book/installation.html#configuration-and-setup for more
information
//umask(0000);

// This check prevents access to debug front controllers that are deployed by accident to
production servers.
// Feel free to remove this, extend it, or make something more sophisticated.
/*
if (isset($_SERVER['HTTP_CLIENT_IP'])
    || isset($_SERVER['HTTP_X_FORWARDED_FOR'])
    || !in_array(@$_SERVER['REMOTE_ADDR'], array('127.0.0.1', 'fe80::1', '::1'))
) {
    header('HTTP/1.0 403 Forbidden');
    exit('You are not allowed to access this file. Check '.basename(__FILE__).' for more
information.');
```

Probably, you will get all kind of requirements when you go to `config.php`. Below, is a list of things to do for not getting all those “warnings”.

1. Change the permissions of `app/cache` and `app/logs`:

```
sudo chmod -R 777 app/cache
sudo chmod -R 777 app/logs
sudo setfacl -dR -m u::rwX app/cache app/logs
```

Install ACL if you don't have it yet:

```
sudo apt-get install acl
```

2. Set the `date.timezone` setting in `php.ini`

```
date.timezone = Europe/Bucharest
```

```
sudo nano /etc/php5/apache2/php.ini
```

Find the `date.timezone` setting for `[date]` section and set it to your timezone. After that, erase “;”, placed at the beginning of the line.

3. Set the `short_open_tag` setting to `off` in the same `php.ini` file

```
short_open_tag
Default Value: Off
```

4. Install and enable a PHP Accelerator (APC recommended)

```
sudo apt-get install php-apc
sudo service apache2 restart
```

After restarting Apache, open a browser window and type in [http://Gebeya.local/app\\_dev.php](http://Gebeya.local/app_dev.php). You should see the following page:

Pic.

## Symfony2 Console

Symfony2 comes with the console component tool that you will use for different tasks. To see a list of things it can do for you type at the command prompt:

```
php app/console list
Creating the Application Bundle
```

## What exactly is a bundle?

Is similar to a plugin in other software, but even better. The key difference is that everything is a bundle in Symfony 2.3.2, including both core framework functionality and the code written for your application.

A bundle is a structured set of files within a directory that implement a single feature.

Tips: A bundle can live anywhere as long as it can be autoloaded (`app/autoload.php`).

You can read more here: [http://symfony.com/doc/current/book/page\\_creation.html#the-bundle-system](http://symfony.com/doc/current/book/page_creation.html#the-bundle-system) - The Bundle System.

## Creating a basic bundle skeleton

Run the following command to start the Symfony's bundle generator:

```
php app/console generate:bundle --namespace=Ju/GebeyaBundle
```

The generator will ask you some questions before generating the bundle. Here are the questions and answers (all, except one, are the default answers):

```
Bundle name [JuGebeyaBundle]: JuGebeyaBundle
Target directory [/var/www/Gebeya/src]: /var/www/Gebeya/src
Configuration format (yml, xml, php, or annotation) [yml]: yml
Do you want to generate the whole directory structure [no]? yes
Do you confirm generation [yes]? yes
Confirm automatic update of your Kernel [yes]? yes
Confirm automatic update of the Routing [yes]? yes
```

Clear the cache after generating the new bundle with:

```
php app/console cache:clear --env=prod
php app/console cache:clear --env=dev
```

The new Gebeya bundle can be now found in the `src` directory of your project: `src/Ju/GebeyaBundle`. The bundle generator made a `DefaultController` with an `index` action. You can access this in your browser: <http://Gebeya.local/hello/Gebeya> or [http://Gebeya.local/app\\_dev.php/hello/Gebeya](http://Gebeya.local/app_dev.php/hello/Gebeya).

## How to remove the AcmeDemoBundle

The Symfony 2.3.2 Standard Edition comes with a complete demo that lives inside a bundle called `AcmeDemoBundle`. It is a great boilerplate to refer to while starting a project, but you'll probably want to eventually remove it.

1. Type the command to delete `Acme` directory:

```
rm -rf /var/www/Gebeya/src/Acme
```

2. Go to: `/var/www/Gebeya/app/AppKernel.php` and delete:

```
// ...  
  
$bundles[] = new AcmeDemoBundleAcmeDemoBundle();  
  
// ...
```

and now delete from `app/config/routing_dev.yml`:

```
# ...  
  
# AcmeDemoBundle routes (to be removed)  
_acme_demo:  
    resource: "@AcmeDemoBundle/Resources/config/routing.yml"
```

3. Finally, clear the cache.

## The Environments

Symfony 2.3.2 has different environments. If you look in the project's web directory, you will see two `php` files: `app.php` and `app_dev.php`. These files are called front controllers; all requests to the application are made through them. The `app.php` file is for production environment and `app_dev.php` is used by web developers when they work on the application in the development environment. The development environment will prove very handy because it will show you all the errors and warnings and the Web Debug Toolbar – the developer's best friend.

## Day 2: The project

This day is about the project specifications. Before diving into the code head-first, let's describe the project a bit more. The following sections describe the features we want to implement in the first version/iteration of the project with some simple stories.

### Users

There are three types of users in the e-commerce application. Let us see them one by one

- **Admin**: owns, configure and manages the page
- **User**: visits the site and purchase items online

- **Store man:** post items on the site

### Story 1: On the homepage, the user sees all the items

When a user comes to Gebeya website, he sees all the items available. The items are sorted by category and then by posted date – newer items first. For each item, only the photo, price, brand, the quantity available and the company are displayed.

For each category, the list shows the first 10 items and a link that allows to list all the items for a given category (Story 2).

On the homepage, the user can refine the Item list (Story 3) or post a new Item (Story 5).

### Story 2: A user can ask for all the items in a given category

When a user clicks on a category name or on a “more items” link on the homepage, he sees all the items for this category sorted by date.

The list is paginated with 20 items per page.

### Story 3: A user refines the list with some keywords

The user can enter some keywords to refine his search. Keywords can be words found in the price, the brand, the category or the company fields.

### Story 4: A user clicks on an item to see more detailed information

The user can select an item from a list to see more detailed information.

### Story 5: A user/store man posts an item

A user can post an item. An item is made of several pieces of information. The process has only two steps: first, the user fills in the form with all the needed information to describe the item, then validates the information by previewing the final item page.

Items can be posted by a registered user/store man who has an account and token in the Gebeya website.

### Story 6: A user applies to become a store man

A user needs to apply to become a store man and be authorized to use Gebeya API. To apply, he must give the following information:

- Name
- Email
- Company

The store man account must be activated by the admin (Story 9). Once activated, the store man receives a token to use with the API via email.



### Story 7: An admin configures the website

An admin can edit the categories available on the website.

### Story 8: An admin manages the items

An admin can edit and remove any posted item.

### Story 9: An admin manages the user accounts

The admin can create or edit store man. He is responsible for activating a user and can also disable one. When the admin activates a new user, the system creates a unique token to be used by the user.

## Day 3

Now, we will define the Gebeya data model, use an ORM to interact with the database and build the first module of the application. But as Symfony does a lot of work for us, we will have a fully functional web module without writing too much PHP code.

### The Relational Model

The user stories from the previous day describe the main objects of our project: **items**, **customers**, **carts**, **orders**, **shipping** and **categories**. Here is the corresponding entity relationship

In addition to the columns described in the stories, we have also added `created_at` and `updated_at` columns. We will configure Symfony to set their value automatically when an object is saved or updated.

### The Database

To store the Items, affiliates and categories in the database, Symfony 2.3.2 uses [Doctrine ORM](#). To define the database connection parameters, you have to edit the `app/config/parameters.yml` file (for this tutorial we will use MySQL):

Now that Doctrine knows about your database, you can have it create the database for you by typing the following command in your terminal:

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

### The Schema

To tell Doctrine about our objects, we will create “metadata” files that will describe how our objects will be stored in the database. Now go to your code editor and create a directory named `doctrine`, inside `src/Ju/GebeyaBundle/Resources/config` directory. Doctrine will contain three files: `Category.orm.yml`, `Item.orm.yml`, `Cart.orm.yml`, `Customer.orm.yml`, `Order.orm.yml` and `Shipping.orm.yml`.

```
src/Ju/GebeyaBundle/Resources/config/doctrine/Category.orm.yml
```

```
Ju\GebeyaBundle\Entity\Category:
  type: entity
  table: category
  repositoryClass: Ju\GebeyaBundle\Repository\CategoryRepository
  id:
    id:
      type: integer
      generator: { strategy: AUTO }
  fields:
    name:
      type: string
      length: 255
      unique: true
  oneToMany:
    items:
      targetEntity: Item
      mappedBy: category
```

```
src/Ju/GebeyaBundle/Resources/config/doctrine/Item.orm.yml
```

```
Ju\GebeyaBundle\Entity\Item:
  type: entity
  table: item
  repositoryClass: Ju\GebeyaBundle\Repository\ItemRepository
  id:
    id:
      type: integer
      generator: { strategy: AUTO }
  fields:
    company:
```

```
  type: string
  length: 255
logo:
  type: string
  length: 255
  nullable: true
name:
  type: string
  length: 255
price:
  type: float
amount:
  type: float
image:
  type: string
  length: 255
created_at:
  type: datetime
updated_at:
  type: datetime
manyToOne:
  category:
    targetEntity: Category
    inversedBy: items
    joinColumn:
      name: category_id
      referencedColumnName: id
manyToMany:
  carts:
    targetEntity: Cart
```

```
    mappedBy: items
  lifecycleCallbacks:
    prePersist: [ setCreatedAtValue, setUpdatedAtValue ]
    preUpdate: [ setUpdatedAtValue ]
```

```
src/Ju/GebeyaBundle/Resources/config/doctrine/Cart.orm.yml
```

```
Ju\GebeyaBundle\Entity\Cart:
```

```
  type: entity
  table: cart
  id:
    id:
      type: integer
      generator: { strategy: AUTO }
  fields:
    price:
      type: float
    created_at:
      type: datetime
    updated_at:
      type: datetime
  manyToOne:
    customer:
      targetEntity: Customer
      inversedBy: carts
      joinColumn:
        name: customer_id
        referencedColumnName: id
  itemOrders:
    targetEntity: ItemOrder
```

```

    inversedBy: cart
    joinColumn:
      name: order_id
      referencedColumnName: id
  manyToMany:
    items:
      targetEntity: Item
    joinTable:
      name: cart_item
    joinColumns:
      cart_id:
        referencedColumnName: id
    inverseJoinColumns:
      item_id:
        referencedColumnName: id
  lifecycleCallbacks:
    prePersist: [ setCreatedAtValue ]
    preUpdate: [ setUpdatedAtValue ]

```

```
src/Ju/GebeyaBundle/Resources/config/doctrine/Customer.orm.yml
```

```

Ju\GebeyaBundle\Entity\Customer:
  type: entity
  table: customer
  id:
    id:
      type: integer
      generator: { strategy: AUTO }
  fields:
    password:

```

```
    type: string
    length: 255
  name:
    type: string
    length: 255
  oneToMany:
    carts:
      targetEntity: Cart
      mappedBy: customer
    address:
      targetEntity: Address
      mappedBy: customer
```

```
src/Ju/GebeyaBundle/Resources/config/doctrine/ItemOrder.orm.yml
```

```
Ju\GebeyaBundle\Entity\ItemOrder:
```

```
  type: entity
  table: item_order
  id:
    id:
      type: integer
      generator: { strategy: AUTO }
  fields:
    receiver_name:
      type: string
      length: 255
    city:
      type: string
      length: 255
    date_of_purchase:
```

```
    type: date
  oneToMany:
    cart:
      targetEntity: Cart
      mappedBy: itemOrders
    address:
      targetEntity: Address
      mappedBy: itemOrders
  manyToOne:
    shiping:
      targetEntity: Shipping
      inversedBy: itemOrders
      joinColumn:
        name: shipping_id
        referencedColumnName: id
  lifecycleCallbacks:
    prePersist: [ setCreatedAtValue ]
    preUpdate: [ setUpdatedAtValue ]
```

```
src/Ju/GebeyaBundle/Resources/config/doctrine/Address.orm.yml
```

```
Ju\GebeyaBundle\Entity\Address:
  type: entity
  table: address
  id:
    id:
      type: integer
      generator: { strategy: AUTO }
  fields:
    city:
```

```
    type: string
    length: 255
  phone_number:
    type: string
    length: 255
  email:
    type: string
    length: 255
  zip:
    type: string
    length: 255
  manyToOne:
    itemOrders:
      targetEntity: ItemOrder
      inversedBy: address
      joinColumn:
        name: item_order_id
        referencedColumnName: id
    customers:
      targetEntity: Customer
      inversedBy: address
      joinColumn:
        name: customer_id
        referencedColumnName: id
```

```
src/Ju/GebeyaBundle/Resources/config/doctrine/Shipping.orm.yml
```

```
Ju\GebeyaBundle\Entity\Shipping:
```

```
  type: entity
```



```

table: shipping
id:
  id:
    type: integer
    generator: { strategy: AUTO }
fields:
  price:
    type: float
  day_of_delivery:
    type: date
oneToMany:
  itemOrders:
    targetEntity: ItemOrder
    mappedBy: shipping
manyToOne:
  shipingType:
    targetEntity: ShippingType
    inversedBy: shippings
    joinColumn:
      name: shipping_type_id
      referencedColumnName: id
lifecycleCallbacks:
  prePersist: [ setCreatedAtValue ]
  preUpdate: [ setUpdatedAtValue ]

```

```
src/Ju/GebeyaBundle/Resources/config/doctrine/ShippingType.orm.yml
```

```
Ju\GebeyaBundle\Entity\ShippingType:
```

```
type: entity
```

```

table: shipping_type
id:
  id:
    type: integer
    generator: { strategy: AUTO }
fields:
  name:
    type: string
    length: 255
oneToMany:
  shippings:
    targetEntity: Shipping
    mappedBy: shipingType

```

## The ORM

Now Doctrine can generate the classes that define our objects for us with the command:

```
php app/console doctrine:generate:entities JuGebeyaBundle
```

If you take a look into `Entity` directory from `JuGebeyaBundle`, you will find the newly generated classes in there: **Item.php**, **Customer.php**, **Cart.php**, **Order.php**, **Shipping.php** and **Category.php**. Open the `php` files and set the `created_at` and `updated_at` values as below:

```

// ...

/**
 * @ORMPrePersist
 */
public function setCreatedAtValue()
{
    if(!$this->getCreatedAt()) {
        $this->created_at = new DateTime();
    }
}

/**

```

```

    * @ORMPreUpdate
    */
    public function setUpdatedAtValue()
    {
        $this->updated_at = new DateTime();
    }

```

You will do the same for `created_at` value of the other classes:

```

// ...

/**
 * @ORMPrePersist
 */
public function setCreatedAtValue()
{
    $this->created_at = new DateTime();
}

// ...

```

This will make Doctrine to set the `created_at` and `updated_at` values when saving or updating objects. This behaviour was defined in `*.orm.yml` files listed above.

We will also ask Doctrine to create our database tables with the command below:

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

The tables have been created in the database but there is no data in them. For any web application, there are three types of data: **initial data** (this is needed for the application to work, in our case we will have some initial categories and an admin user), **test data** (needed for the application to be tested) and **user data** (created by users during the normal life of the application).

To populate the database with some initial data, we will use [DoctrineFixturesBundle](#). To setup this bundle, we have to follow the next steps:

1. Add the following to your `composer.json` file, in the `require` section:

```

// ...
"require": {
    // ...
    "doctrine/doctrine-fixtures-bundle": "dev-master",
    "doctrine/data-fixtures": "dev-master"
}

```

```
},  
// ...
```

2. Update the `vendor` libraries:

```
php composer.phar update
```

3. Register the bundle `DoctrineFixturesBundle` in `app/AppKernel.php`:

```
// ...  
  
public function registerBundles()  
{  
    $bundles = array(  
        // ...  
        new Doctrine\Bundle\FixturesBundle\DoctrineFixturesBundle()  
    );  
  
    // ...  
}
```

OR

1) just type the following command to install fixture bundle

```
php composer.phar require --dev doctrine/doctrine-fixtures-bundle
```

2) Register the bundle `DoctrineFixturesBundle` in `app/AppKernel.php`:

```
// ...  
  
public function registerBundles()  
{  
    $bundles = array(  
        // ...  
        new Doctrine\Bundle\FixturesBundle\DoctrineFixturesBundle()  
    );  
  
    // ...  
}
```

Now that everything is set up, we will create some new classes to load data in a new folder, named `src/Ju/GebeyaBundle/DataFixtures/ORM`, in our bundle:

src/Ju/GebeyaBundle/DataFixtures/ORM/AddressData.php

```
<?php
/**
 * Created by PhpStorm.
 * User: yabi
 * Date: 2/18/2016
 * Time: 9:11 PM
 */
namespace Ju\GebeyaBundle\DataFixtures\ORM;
use Doctrine\Common\Persistence\ObjectManager;
use Doctrine\Common\DataFixtures\AbstractFixture;
use Doctrine\Common\DataFixtures\OrderedFixtureInterface;
use Ju\GebeyaBundle\Entity\Address;
class AddressData extends AbstractFixture implements OrderedFixtureInterface
{
    public function load(ObjectManager $em)
    {
        $jimma = new Address();
        $jimma->setCity('Jimma');
        $jimma->setPhoneNumber('+251917000000');
        $jimma->setEmail('mr.jimma@ju.edu.et');
        $jimma->setZip('000Zx');
        $jimma->setItemOrders($em->merge($this->getReference('item-order-alemu')));
        $em->persist($jimma);
        $em->flush();
        $this->addReference('address-jimma', $jimma);
    }
    public function getOrder()
    {
        return 5; // the order in which fixtures will be loaded
    }
}
```

```
}  
}
```

src/Ju/GebeyaBundle/DataFixtures/ORM/**CartData.php**

```
<?php  
<?php  
/**  
 * Created by PhpStorm.  
 * User: yabi  
 * Date: 2/19/16  
 * Time: 10:02 AM  
 */  
  
namespace Ju\GebeyaBundle\DataFixtures\ORM;  
use Doctrine\Common\Persistence\ObjectManager;  
use Doctrine\Common\DataFixtures\AbstractFixture;  
use Doctrine\Common\DataFixtures\OrderedFixtureInterface;  
use \Ju\GebeyaBundle\Entity\Cart;  
use Symfony\Component\Validator\Constraints\DateTime;  
class CartData extends AbstractFixture implements OrderedFixtureInterface  
{  
    public function load(ObjectManager $em)  
    {  
        $cart1 = new Cart();  
        $cart1->setPrice(12.5);  
        //$cart1->addItem($em->merge($this->getReference('item-mars')));  
        $cart1->setCustomer($em->merge($this->getReference('customer-abebe')));  
        $cart1->setItemOrders($em->merge($this->getReference('item-order-alemu')));  
        $cart1->setCreatedAt(new \DateTime());  
        $cart1->setUpdatedAt(new \DateTime());  
        $em->persist($cart1);  
        $em->flush();  
    }  
}
```

```

        $this->addReference('cart-cart1', $cart1);
    }

    public function getOrder()
    {
        return 7; // the order in which fixtures will be loaded
    }
}

```

src/Ju/GebeyaBundle/DataFixtures/ORM/CategoryData.php

```
<?php
```

```

/**
 * Created by PhpStorm.
 * User: yabi
 * Date: 2/19/16
 * Time: 10:45 AM
 */

namespace Ju\GebeyaBundle\DataFixtures\ORM;

use Doctrine\Common\Persistence\ObjectManager;
use Doctrine\Common\DataFixtures\AbstractFixture;
use Doctrine\Common\DataFixtures\OrderedFixtureInterface;
use \Ju\GebeyaBundle\Entity\Category;

class CategoryData extends AbstractFixture implements OrderedFixtureInterface
{
    public function load(ObjectManager $em)
    {
        $foods = new Category();
        $foods->setName('Foods');
        $water = new Category();
        $water->setName('Bottled water');
        $juice = new Category();
    }
}

```

```

        $juice->setName('Bottled juice');
        $em->persist($foods);
        $em->persist($water);
        $em->persist($juice);
        $em->flush();

        $this->addReference('category-food', $foods);
        $this->addReference('category-water', $water);
        $this->addReference('category-juice', $juice);
    }

    public function getOrder()
    {
        return 1; // the order in which fixtures will be loaded
    }
}

```

src/Ju/GebeyaBundle/DataFixtures/ORM/CustomerData.php

```
<?php
```

```

/**
 * Created by PhpStorm.
 * User: yabi
 * Date: 2/19/16
 * Time: 10:55 AM
 */

namespace JuGebeyaBundleDataFixturesORM;
namespace JuGebeyaBundle\DataFixtures\ORM;
use Doctrine\Common\Persistence\ObjectManager;
use Doctrine\Common\DataFixtures\AbstractFixture;
use Doctrine\Common\DataFixtures\OrderedFixtureInterface;
use JuGebeyaBundle\Entity\Customer;

```



```

class CustomerData extends AbstractFixture implements OrderedFixtureInterface
{
    public function load(ObjectManager $em)
    {
        $abebe = new Customer();
        $abebe->setPassword('AbebePassword');
        $abebe->setName('Abebe');
        $abebe->addAddress($em->merge($this->getReference('address-jimma')));
        $kebede = new Customer();
        $kebede->setPassword('KebedePassword');
        $kebede->setName('Kebede');
        $kebede->addAddress($em->merge($this->getReference('address-jimma')));
        $em->persist($abebe);
        $em->persist($kebede);
        $em->flush();
        $this->addReference('customer-abebe', $abebe);
        $this->addReference('customer-kebede', $kebede);
    }

    public function getOrder()
    {
        return 6; // the order in which fixtures will be loaded
    }
}

```

src/Ju/GebeyaBundle/DataFixtures/ORM/ItemData.php

```
<?php
```

```

/**
 * Created by PhpStorm.
 * User: yabi
 * Date: 2/19/16

```

```

* Time: 10:10 AM
*/

namespace Ju\GebeyaBundle\DataFixtures\ORM;
use Doctrine\Common\Persistence\ObjectManager;
use Doctrine\Common\DataFixtures\AbstractFixture;
use Doctrine\Common\DataFixtures\OrderedFixtureInterface;
use Ju\GebeyaBundle\Entity\Item;
class ItemData extends AbstractFixture implements OrderedFixtureInterface
{
    public function load(ObjectManager $em)
    {
        $mars = new Item();
        $mars->setCompany('Mars');
        $mars->setLogo('Serve best foods');
        $mars->setName('Mars chocolate');
        $mars->setPrice(12.5);
        $mars->setAmount(200);
        $mars->setImage('product1.jpg');
        $mars->setCreatedAt(new \DateTime());
        $mars->setUpdatedAt(new \DateTime());
        $mars->setCategory($em->merge($this->getReference('category-food')));
        $mars->addCart($em->merge($this->getReference('cart-cart1')));
        $aqua_safe = new Item();
        $aqua_safe->setCompany('Aqua-safe');
        $aqua_safe->setLogo('Perfect Drinking satisfaction');
        $aqua_safe->setName('Aqua-safe 2L water');
        $aqua_safe->setPrice(15.5);
        $aqua_safe->setAmount(200);
        $aqua_safe->setImage('product2.jpg');
        $aqua_safe->setCreatedAt(new \DateTime());
        $aqua_safe->setUpdatedAt(new \DateTime());
    }
}

```

```

    $aqua_safe->setCategory($em->merge($this->getReference('category-water')));
    $aqua_safe->addCart($em->merge($this->getReference('cart-cart1')));
    $rani = new Item();
    $rani->setCompany('Rani Ltd. ');
    $rani->setLogo('yummy juice');
    $rani->setName('Rani 1.5L mango Juice');
    $rani->setPrice(15.5);
    $rani->setAmount(200);
    $rani->setImage('product3.jpg');
    $rani->setCreatedAt(new \DateTime());
    $rani->setUpdatedAt(new \DateTime());
    $rani->setCategory($em->merge($this->getReference('category-juice')));
    $rani->addCart($em->merge($this->getReference('cart-cart1')));
    $em->persist($mars);
    $em->persist($aqua_safe);
    $em->persist($rani);
    $em->flush();
    $this->addReference('item-mars', $mars);
    $this->addReference('item-aqua_safe', $aqua_safe);
    $this->addReference('item-rani', $rani);
}

public function getOrder()
{
    return 8; // the order in which fixtures will be loaded
}
}

```

src/Ju/GebeyaBundle/DataFixtures/ORM/ItemOrderData.php

<?php

```

/**
 * Created by PhpStorm.
 * User: yabi

```

```

* Date: 2/18/2016
* Time: 9:36 PM
*/

namespace Ju\GebeyaBundle\DataFixtures\ORM;

use Doctrine\Common\Persistence\ObjectManager;
use Doctrine\Common\DataFixtures\AbstractFixture;
use Doctrine\Common\DataFixtures\OrderedFixtureInterface;
use Ju\GebeyaBundle\Entity\ItemOrder;
use Symfony\Component\Validator\Constraints\DateTime;

class ItemOrderData extends AbstractFixture implements OrderedFixtureInterface
{
    public function load(ObjectManager $em)
    {
        $alemu = new ItemOrder();
        $alemu->setReceiverName('Alemu');
        $alemu->setCity('Jimma');
        $alemu->setDateOfPurchase(new \DateTime());
        //$alemu->addCart($em->merge($this->getReference('cart-cart1')));
        //$alemu->addAddress($em->merge($this->getReference('address-jimma')));
        $alemu->setShiping($em->merge($this->getReference('shipping-shipping1')));
        $em->persist($alemu);
        $em->flush();
        $this->addReference('item-order-alemu', $alemu);
    }

    public function getOrder()
    {
        return 4; // the order in which fixtures will be loaded
    }
}

```

src/Ju/GebeyaBundle/DataFixtures/ORM/ShippingData.php

<?php

```
/**
 * Created by PhpStorm.
 * User: yabi
 * Date: 2/19/16
 * Time: 11:09 AM
 */

namespace JuGebeyaBundleDataFixturesORM;
namespace Ju\GebeyaBundle\DataFixtures\ORM;
use Doctrine\Common\Persistence\ObjectManager;
use Doctrine\Common\DataFixtures\AbstractFixture;
use Doctrine\Common\DataFixtures\OrderedFixtureInterface;
use Ju\GebeyaBundle\Entity\Shipping;
class ShippingData extends AbstractFixture implements OrderedFixtureInterface
{
    public function load(ObjectManager $em)
    {
        $shipping1 = new Shipping();
        $shipping1->setPrice(17.45);
        $shipping1->setDayOfDelivery(new \DateTime());
        //$shipping1->addItemOrder($em->merge($this->getReference('item-order-alemu')));
        $shipping1->setShippingType($em->merge($this->getReference('shipping-type-car')));
        $em->persist($shipping1);
        $em->flush();
        $this->addReference('shipping-shipping1', $shipping1);
    }
    public function getOrder()
    {
        return 3; // the order in which fixtures will be loaded
    }
}
```

```
}
```

src/Ju/GebeyaBundle/DataFixtures/ORM/ShippingTypeData.php

```
<?php
```

```
/**
 * Created by PhpStorm.
 * User: yabi
 * Date: 2/19/16
 * Time: 11:12 AM
 */

namespace Ju\GebeyaBundle\DataFixtures\ORM;
use Doctrine\Common\Persistence\ObjectManager;
use Doctrine\Common\DataFixtures\AbstractFixture;
use Doctrine\Common\DataFixtures\OrderedFixtureInterface;
use Ju\GebeyaBundle\Entity\ShippingType;
class ShippingTypeData extends AbstractFixture implements OrderedFixtureInterface
{
    public function load(ObjectManager $em)
    {
        $car = new ShippingType();
        $car->setName(17.45);
        // $car->addShiping($em->merge($this->getReference('shipping-shiping1')));
        $em->persist($car);
        $em->flush();
        $this->addReference('shipping-type-car', $car);
    }
    public function getOrder()
    {
        return 2; // the order in which fixtures will be loaded
    }
}
```

```
}
```

Once your fixtures have been written, you can load them via the command line by using the `doctrine:fixtures:load` command:

```
php app/console doctrine:fixtures:load
```

Now, if you check your database, you should see the data loaded into tables.

See it in the browser

If you run the command below, it will create a new controller `src/Ju/GebeyaBundle/Controllers/ItemController.php` with actions for listing, creating, editing and deleting Items (and their corresponding templates, form and routes):

```
php app/console doctrine:generate:crud --entity=JuGebeyaBundle:Item --route-prefix=item --with-write --format=yml
```

After running this command, you will need to do some configurations the prompter requires you to. So just select the default answers for them.

To view this in the browser, we must import the new routes that were created in `src/Ju/GebeyaBundle/Resources/config/routing/item.yml` into our bundle main routing file:

```
ju_gebeya_item:
    resource: "@JuGebeyaBundle/Resources/config/routing/item.yml"
    prefix: /item
```

```
# ...
```

We will also need to add a `_toString()` method to our `Category`, `Cart`, `ItemOrder`, `Customer`, and similar classes to be used by the category drop down from the edit and create forms:

```
// ...

public function _toString()
{
    return $this->getName();
}
```

```
}
```

```
// ...
```

Clear the cache:

```
php app/console cache:clear --env=dev  
php app/console cache:clear --env=prod
```

You can now test the Item controller in a browser: <http://gebeya.local/Item/> or, in development environment, [http://gebeya.local/app\\_dev.php/Item/](http://gebeya.local/app_dev.php/Item/).

You can now create and edit Items. Try to leave a required field blank, or try to enter invalid data. That's right, Symfony has created basic validation rules by introspecting the database schema.

**That's all. Today, we have barely written PHP code but we have a working web module for the Item model, ready to be tweaked and customized. Tomorrow, we will get familiar with the controller and the view. See you next time!**

## Day 4

Today, we are going to customize the basic **controllers** we created yesterday. It already has most of the code we need for Gebeya:

- A page to **list** all Items
- A page to **create** a new Item
- A page to **update** an existing Item
- A page to **delete** a Item

Although the code is ready to be used as is, we will refactor the templates to match closer to the [Gebeya mockups](#).

### The MVC Architecture

For web development, the most common solution for organizing your code nowadays is the [MVC design pattern](#). In short, the MVC design pattern defines a way to organize your code according to its nature. This pattern separates the code into **three layers**:

- The **Model** layer defines the business logic (the database belongs to this layer). You already know that Symfony stores all the classes and files related to the Model in the `Entity/` directory of your bundles.



- The **View** is what the user interacts with (a template engine is part of this layer). In Symfony 2.3.2, the View layer is mainly made of Twig templates. They are stored in various `Resources/views/` directories as we will see later in these lines.
- The **Controller** is a piece of code that calls the Model to get some data that it passes to the View for rendering to the client. When we installed Symfony at the beginning of this tutorial, we saw that all requests are managed by front controllers (`app.php` and `app_dev.php`). These front controllers delegate the real work to **actions**.

## The Layout

If you have a closer look at the [mockups](#), you will notice that much of each page looks the same. You already know that code duplication is bad, whether we are talking about HTML or PHP code, so we need to find a way to prevent these common view elements from resulting in code duplication.

One way to solve the problem is to define a header and a footer and include them in each template. A better way is to use another design pattern to solve this problem: the [decorator design pattern](#). The decorator design pattern resolves the problem the other way around: the template is decorated after the content is rendered by a global template, called a **layout**.

Symfony2 does not come with a default layout, so we will create one and use it to decorate our application pages.

Create a new file `layout.html.twig` in the `src/Ju/GebeyaBundle/Resources/views/` directory and put in the following code:

```
src/Ju/GebeyaBundle/Resources/views/layout.html.twig

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="description" content="">
  <meta name="author" content="">
  <title>Home | Gebeya</title>
  {% block stylesheets %}
    <link rel="stylesheet"
href="{ { asset('bundles/jugebeya/css/bootstrap.min.css') } }" type="text/css"
```

```

media="all" />

    <link rel="stylesheet" href="{{ asset('bundles/jugebeya/css/font-
awesome.min.css') }}" type="text/css" media="all" />

    <link rel="stylesheet" href="{{ asset('bundles/jugebeya/css/prettyPhoto.css') }}"
type="text/css" media="all" />

    <link rel="stylesheet" href="{{ asset('bundles/jugebeya/css/price-range.css') }}"
type="text/css" media="all" />

    <link rel="stylesheet" href="{{ asset('bundles/jugebeya/css/animate.css') }}"
type="text/css" media="all" />

    <link rel="stylesheet" href="{{ asset('bundles/jugebeya/css/main.css') }}"
type="text/css" media="all" />

    <link rel="stylesheet" href="{{ asset('bundles/jugebeya/css/responsive.css') }}"
type="text/css" media="all" />

    {% endblock %}
</head><!--/head-->

<body>
{% block header %}
<header id="header"><!--header-->

    <div class="header_top"><!--header_top-->

        <div class="container">

            <div class="row">

                <div class="col-sm-6">

                    <div class="contactinfo">

                        <ul class="nav nav-pills">

                            <li><a href="#"><i class="fa fa-phone"></i> +2 95 01 88
821</a></li>

                            <li><a href="#"><i class="fa fa-envelope"></i>
info@domain.com</a></li>

                        </ul>

                    </div>

                </div>

                <div class="col-sm-6">

                    <div class="social-icons pull-right">

```

```

        <ul class="nav navbar-nav">
            <li><a href="#"><i class="fa fa-facebook"></i></a></li>
            <li><a href="#"><i class="fa fa-twitter"></i></a></li>
            <li><a href="#"><i class="fa fa-linkedin"></i></a></li>
            <li><a href="#"><i class="fa fa-dribbble"></i></a></li>
            <li><a href="#"><i class="fa fa-google-plus"></i></a></li>
        </ul>
    </div>
</div>
</div>
</div>
</div><!--/header_top-->
<div class="header-middle"><!--header-middle-->
    <div class="container">
        <div class="row">
            <div class="col-sm-4">
                <div class="logo pull-left">
                    <a href="{{path('ju_gebeya_homepage')}}"></a>
                </div>
            </div>
            <div class="col-sm-8">
                <div class="shop-menu pull-right">
                    <ul class="nav navbar-nav">
                        <li><a href="{{path('fos_user_resetting_request')}}"><i class="fa
fa-user"></i> Resetting</a></li>
                        { #<li><a href="#"><i class="fa fa-star"></i> Wishlist</a></li># }
                        <li><a href="{{path('cart')}}"><i class="fa fa-shopping-cart"></i>
My Carts</a></li>
                        { % if is_granted("ROLE_SUPER_ADMIN") or
is_granted("ROLE_ADMIN") or is_granted("ROLE_USER") %}
                        <li>

```

```

        <a href="{path('fos_user_security_logout')}"><i class="fa fa-
sign-out"></i> Logout</a>

        </li>

        {% else %}

        <li>

        <a href="{path('fos_user_security_login')}"><i class="fa fa-
lock"></i> Login</a>

        </li>

        {% endif %}

    </ul>

</div>

</div>

</div>

</div>

</div><!--/header-middle-->

<div class="header-bottom"><!--header-bottom-->

    <div class="container">

        <div class="row">

            <div class="col-sm-6 col-lg-push-3">

                {% for flashMessage in app.session.flashbag.get('error') %}

                    <div>

                        {{ flashMessage }}

                    </div>

                {% endfor %}

                {% for flashMessage in app.session.flashbag.get('success') %}

                    <div>

                        {{ flashMessage }}

                    </div>

                {% endfor %}

            </div>

        </div>

    </div>

```

```

<div class="row">
  <div class="col-sm-9">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse"
data-target=".navbar-collapse">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
    </div>
    <div class="mainmenu pull-left">
      <ul class="nav navbar-nav collapse navbar-collapse">
        <li><a href="{ { path('ju_gebeya_homepage') } }"
class="active">Home</a></li>
        <li class="dropdown"><a href="#">Shop<i class="fa fa-angle-
down"></i></a>
          <ul role="menu" class="sub-menu">
            <li><a href="{ { path('item') } }">Products</a></li>
            <li><a href="{ { path('category') } }">Categories</a></li>
            <li><a href="{ { path('cart') } }">My Carts</a></li>
            {% if is_granted("ROLE_SUPER_ADMIN") or
is_granted("ROLE_ADMIN") or is_granted("ROLE_USER") %}
              <li>
                <a href="{ { path('fos_user_security_logout') } }"><i
class="fa fa-sign-out"></i> Logout</a>
              </li>
            {% else %}
              <li>
                <a href="{ { path('fos_user_security_login') } }"><i
class="fa fa-lock"></i> Login</a>
              </li>
            {% endif %}
          </ul>
        </li>
      </ul>
    </div>
  </div>
</div>

```

```

        </ul>
        </li>
        <li><a href="contact-us.html">Contact</a></li>
    </ul>
</div>
</div>
<div class="col-sm-3">
    <div class="search_box pull-right">
        <input type="text" placeholder="Search"/>
    </div>
</div>
</div>
</div>
</div><!--/header-bottom-->
</header><!--/header-->
{% endblock %}
{% block body %}
{% endblock %}
{% block footer %}
<footer id="footer"><!--Footer-->
    <div class="footer-top">
        <div class="container">
            <div class="row">
                <div class="col-sm-2">
                    <div class="companyinfo">
                        <h2><span>e</span>-shopper</h2>
                        <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit,sed do
eiusmod tempor</p>
                    </div>
                </div>
                <div class="col-sm-7">

```

```

<div class="col-sm-3">
  <div class="video-gallery text-center">
    <a href="#">
      <div class="iframe-img">
        
      </div>
      <div class="overlay-icon">
        <i class="fa fa-play-circle-o"></i>
      </div>
    </a>
    <p>Circle of Hands</p>
    <h2>24 DEC 2014</h2>
  </div>
</div>
<div class="col-sm-3">
  <div class="video-gallery text-center">
    <a href="#">
      <div class="iframe-img">
        
      </div>
      <div class="overlay-icon">
        <i class="fa fa-play-circle-o"></i>
      </div>
    </a>
    <p>Circle of Hands</p>
    <h2>24 DEC 2014</h2>
  </div>
</div>
<div class="col-sm-3">
  <div class="video-gallery text-center">
    <a href="#">

```

```

        <div class="iframe-img">
            
        </div>
        <div class="overlay-icon">
            <i class="fa fa-play-circle-o"></i>
        </div>
    </a>
    <p>Circle of Hands</p>
    <h2>24 DEC 2014</h2>
</div>
</div>
<div class="col-sm-3">
    <div class="video-gallery text-center">
        <a href="#">
            <div class="iframe-img">
                
            </div>
            <div class="overlay-icon">
                <i class="fa fa-play-circle-o"></i>
            </div>
        </a>
        <p>Circle of Hands</p>
        <h2>24 DEC 2014</h2>
    </div>
</div>
</div>
<div class="col-sm-3">
    <div class="address">
        
        <p>505 S Atlantic Ave Virginia Beach, VA(Virginia)</p>
    </div>
</div>

```



```

        </div>
    </div>
</div>
</div>
</div>
<div class="footer-widget">
    <div class="container">
        <div class="row">
            <div class="col-sm-2">
                <div class="single-widget">
                    <h2>Service</h2>
                    <ul class="nav nav-pills nav-stacked">
                        <li><a href="#">Online Help</a></li>
                        <li><a href="#">Contact Us</a></li>
                        <li><a href="#">Order Status</a></li>
                        <li><a href="#">Change Location</a></li>
                        <li><a href="#">FAQ's</a></li>
                    </ul>
                </div>
            </div>
            <div class="col-sm-2">
                <div class="single-widget">
                    <h2>Quock Shop</h2>
                    <ul class="nav nav-pills nav-stacked">
                        <li><a href="#">T-Shirt</a></li>
                        <li><a href="#">Mens</a></li>
                        <li><a href="#">Womens</a></li>
                        <li><a href="#">Gift Cards</a></li>
                        <li><a href="#">Shoes</a></li>
                    </ul>
                </div>
            </div>
        </div>
    </div>
</div>

```

```

    </div>
</div>
<div class="col-sm-2">
    <div class="single-widget">
        <h2>Policies</h2>
        <ul class="nav nav-pills nav-stacked">
            <li><a href="#">Terms of Use</a></li>
            <li><a href="#">Privacy Policy</a></li>
            <li><a href="#">Refund Policy</a></li>
            <li><a href="#">Billing System</a></li>
            <li><a href="#">Ticket System</a></li>
        </ul>
    </div>
</div>
<div class="col-sm-2">
    <div class="single-widget">
        <h2>About Shopper</h2>
        <ul class="nav nav-pills nav-stacked">
            <li><a href="#">Company Information</a></li>
            <li><a href="#">Careers</a></li>
            <li><a href="#">Store Location</a></li>
            <li><a href="#">Affiliate Program</a></li>
            <li><a href="#">Copyright</a></li>
        </ul>
    </div>
</div>
<div class="col-sm-3 col-sm-offset-1">
    <div class="single-widget">
        <h2>About Shopper</h2>
        <form action="#" class="searchform">
            <input type="text" placeholder="Your email address" />

```

```

        <button type="submit" class="btn btn-default"><i class="fa fa-
arrow-circle-o-right"></i></button>

        <p>Get the most recent updates from <br />our site and be updated
your self...</p>

        </form>

    </div>

</div>

</div>

</div>

</div>

<div class="footer-bottom">
    <div class="container">
        <div class="row">
            <p class="pull-left">Copyright © 2013 E-SHOPPER Inc. All rights
reserved.</p>
            <p class="pull-right">Designed by <span><a target="_blank"
href="http://www.themeum.com">Themeum</a></span></p>
        </div>
    </div>
</div>
</footer><!--/Footer-->
{% endblock %}
{% block javascripts %}
    <script type="text/javascript"
        src="{ { asset('bundles/jugebeya/js/jquery.js') } }">
    </script>
    <script type="text/javascript"
        src="{ { asset('bundles/jugebeya/js/bootstrap.min.js') } }">
    </script>
    <script type="text/javascript"
        src="{ { asset('bundles/jugebeya/js/jquery.scrollUp.min.js') } }">
    </script>

```

```

<script type="text/javascript"
    src="{{ asset('bundles/jugebeya/js/price-range.js') }}">
</script>
<script type="text/javascript"
    src="{{ asset('bundles/jugebeya/js/jquery.prettyPhoto.js') }}">
</script>
<script type="text/javascript"
    src="{{ asset('bundles/jugebeya/js/main.js') }}">
</script>
{% endblock %}
</body>
</html>

```

## Twig Blocks

In Twig, the default Symfony template engine, you can define **blocks** as we did above. A twig block can have a default content (look at the `title` block, for example) that can be replaced or extended in the child template as you will see in a moment.

Now, to make use of the layout we created, we will need to edit all the Item templates (`index`, `edit`, `new` and `show` from `src/Ju/GebeyaBundle/Resources/views/Item/`) to extend the parent template (the layout) and to overwrite the `content` block we defined with the `body` block content from the original template

```

{% extends 'JuGebeyaBundle::layout.html.twig' %}

{% block content %}
    <!-- original body block code goes here -->
{% endblock %}

```

## The Stylesheets, Images and JavaScripts

As this is not about web design, we have already prepared all the needed assets we will use for Gebeya: you can get the image files from trainers and put them into the `src/Ju/GebeyaBundle/Resources/public/images/` directory; also get the stylesheet(css) files and put them into the `src/Ju/GebeyaBundle/Resources/public/css/` directory and there are javascript files put them in `src/Ju/GebeyaBundle/Resources/public/js/` directory.

Now run

```
php app/console assets:install web --symlink
```

to tell Symfony to make them available to the public.

If you look in the `css` folder, you will notice that we have seven `css` files: `animate.css`, `bootstrap.min.css`, `font-awesome.min.css`, `prettyPhoto.css`, `price-rance.css`, `responsive.css` and `main.css`. All seven `css` files are needed in all Gebeya pages, so we included it in the layout in the stylesheet twig block.

To add a new `css` file in a template, we will overwrite the stylesheet block, but call the parent before adding the new `css` file (so we would have the `main.css` and the additional `css` files we need).

```
{% extends 'JuGebeyaBundle::layout.html.twig' %}

{% block stylesheets %}
    {{ parent() }}
    <link rel="stylesheet" href="{{ asset('bundles/JuGebeya/css/Items.css') }}" type="text/css"
media="all" />
{% endblock %}
```

```
{% extends 'JuGebeyaBundle::layout.html.twig' %}

{% block stylesheets %}
    {{ parent() }}
    <link rel="stylesheet" href="{{ asset('bundles/JuGebeya/css/Item.css') }}" type="text/css"
media="all" />
{% endblock %}

<!-- rest of the code -->
```

## The Item Homepage Action

Each action is represented by a method of a class. For the Item homepage, the class is `ItemController` and the method is `indexAction()`. It retrieves all the Items from the database.

```
// ...

public function indexAction()
{
    $em = $this->getDoctrine()->getManager();

    $entities = $em->getRepository('JuGebeyaBundle:Item')->findAll();

    return $this->render('JuGebeyaBundle:Item:index.html.twig', array(
        'entities' => $entities
    ));
}

// ...
```

Let's have a closer look at the code: the `indexAction()` method gets the **Doctrine entity manager** object, which is responsible for handling the process of persisting and fetching objects to and from database, and then the **repository**, that will create a query to retrieve all the Items. It returns a Doctrine `ArrayCollection` of `Item` objects that are passed to the template (the View).

### The Item Homepage Template

The `index.html.twig` template generates an HTML table for all the Items. Here is the current template code:

```
{% extends 'JuGebeyaBundle::layout.html.twig' %}

{% block stylesheets %}
    {{ parent() }}
    <link rel="stylesheet" href="{{ asset('bundles/JuGebeya/css/Items.css') }}" type="text/css"
media="all" />
{% endblock %}

{% block content %}
    <h1>Item list</h1>

    <table class="records_list">
        <thead>
            <tr>
                <th>Id</th>
                <th>Type</th>
                <th>Company</th>
                <th>Logo</th>
                <th>Url</th>
                <th>Position</th>
```

```

<th>Location</th>
<th>Description</th>
<th>How_to_apply</th>
<th>Token</th>
<th>Is_public</th>
<th>Is_activated</th>
<th>Email</th>
<th>Expires_at</th>
<th>Created_at</th>
<th>Updated_at</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
{% for entity in entities %}
<tr>
<td><a href="{{ path('Ju_Item_show', { 'id': entity.id }) }}">{{ entity.id }}</a></td>
<td>{{ entity.type }}</td>
<td>{{ entity.company }}</td>
<td>{{ entity.logo }}</td>
<td>{{ entity.url }}</td>
<td>{{ entity.position }}</td>
<td>{{ entity.location }}</td>
<td>{{ entity.description }}</td>
<td>{{ entity.howtoapply }}</td>
<td>{{ entity.token }}</td>
<td>{{ entity.ispublic }}</td>
<td>{{ entity.isactivated }}</td>
<td>{{ entity.email }}</td>
<td>{% if entity.expiresat %} {{ entity.expiresat|date('Y-m-d H:i:s') }} {% endif
%}</td>
<td>{% if entity.createdat %} {{ entity.createdat|date('Y-m-d H:i:s') }} {% endif
%}</td>
<td>{% if entity.updatedat %} {{ entity.updatedat|date('Y-m-d H:i:s') }} {% endif
%}</td>
<td>
<ul>
<li>
<a href="{{ path('Ju_Item_show', { 'id': entity.id }) }}">show</a>
</li>
<li>
<a href="{{ path('Ju_Item_edit', { 'id': entity.id }) }}">edit </a>
</li>
</ul>
</td>
</tr>

```

```

    {% endfor %}
  </tbody>
</table>

<ul>
  <li>
    <a href="{{ path('Ju_Item_new') }}">
      Create a new entry
    </a>
  </li>
</ul>
{% endblock %}

```

Let's clean this up a bit to only display a sub-set of the available columns. Replace the `twig block content` with the one below:

```

{% block content %}
  <div id="Items">
    <table class="Items">
      {% for entity in entities %}
        <tr class="{{ cycle(['even', 'odd'], loop.index) }}">
          <td class="location">{{ entity.location }}</td>
          <td class="position">
            <a href="{{ path('Ju_Item_show', { 'id': entity.id }) }}">
              {{ entity.position }}
            </a>
          </td>
          <td class="company">{{ entity.company }}</td>
        </tr>
      {% endfor %}
    </table>
  </div>
{% endblock %}

```

## The Item Page Template

Now let's customize the template of the Item page. Open the `show.html.twig` file and replace its content with the following code:

```

{% extends 'JuGebeyaBundle::layout.html.twig' %}

{% block title %}
  {{ entity.company }} is looking for a {{ entity.position }}
{% endblock %}

```



```

{% block stylesheets %}
    {{ parent() }}
    <link rel="stylesheet" href="{{ asset('bundles/JuGebeya/css/Item.css') }}" type="text/css"
media="all" />
{% endblock %}

{% block content %}
    <div id="Item">
        <h1>{{ entity.company }}</h1>
        <h2>{{ entity.location }}</h2>
        <h3>
            {{ entity.position }}
            <small> - {{ entity.type }}</small>
        </h3>

        {% if entity.logo %}
            <div class="logo">
                <a href="{{ entity.url }}">
                    
                </a>
            </div>
        {% endif %}

        <div class="description">
            {{ entity.description|nl2br }}
        </div>

        <h4>How to apply?</h4>

        <p class="how_to_apply">{{ entity.howtoapply }}</p>

        <div class="meta">
            <small>posted on {{ entity.createdat|date('m/d/Y') }}</small>
        </div>

        <div style="padding: 20px 0">
            <a href="{{ path('Ju_Item_edit', { 'id': entity.id }) }}">
                Edit
            </a>
        </div>
    </div>
{% endblock %}

```

## The Item Page Action

The Item page is generated by the `show` action, defined in the `showAction()` method of the `ItemController`:

```
public function showAction($id)
{
    $em = $this->getDoctrine()->getManager();

    $entity = $em->getRepository('JuGebeyaBundle:Item')->find($id);

    if (!$entity) {
        throw $this->createNotFoundException('Unable to find Item entity.');
    }

    $deleteForm = $this->createDeleteForm($id);

    return $this->render('JuGebeyaBundle:Item:show.html.twig', array(
        'entity' => $entity,
        'delete_form' => $deleteForm->createView(),
    ));
}
```

As in the `index` action, the `JuGebeyaBundle` repository class is used to retrieve a `Item`, this time using the `find()` method. The parameter of this method is the unique identifier of a `Item`, its primary key. The next section will explain why the `$id` parameter of the `actionShow()` function contains the `Item` primary key.

If the `Item` does not exist in the database, we want to forward the user to a 404 page, which is exactly what the `throw $this->createNotFoundException()` does.

As for exceptions, the page displayed to the user is different in the `prod` environment and in the `dev` environment.

## Day 5

### URLs

If you click on a `Item` on the Gebeya homepage, the URL looks like this: `/Item/1/show`. If you have already developed PHP websites, you are probably more accustomed to URLs like `/Item.php?id=1`. How does Symfony make it work? How does Symfony determine the action to call based on this URL? Why is the `id` of the `Item` retrieved with the `$id` parameter in the action? Here, we will answer all these questions.

You have already seen the following code in the `src/Ju/GebeyaBundle/Resources/views/Item/index.html.twig` template:

```
{{ path('Ju_Item_show', { 'id': entity.id }) }}
```

This uses the `path` template helper function to generate the url for the Item which has the `id` 1. The `Ju_Item_show` is the name of the route used, defined in the configuration as you will see below.

## Routing Configuration

In Symfony2, routing configuration is usually done in the `app/config/routing.yml`. This imports specific bundle routing configuration. In our case, the `src/Ju/GebeyaBundle/Resources/config/routing.yml` file is imported:

```
Ju_Gebeya:
    resource: "@JuGebeyaBundle/Resources/config/routing.yml"
    prefix: /
```

Now, if you look in the `GebeyaBundle routing.yml` you will see that it imports another routing file, the one for the Item controller and defines a route called `Ju_Gebeya_homepage` for the `/hello/{name}` URL pattern:

```
JuGebeyaBundle_Item:
    resource: "@JuGebeyaBundle/Resources/config/routing/Item.yml"
    prefix: /Item

Ju_Gebeya_homepage:
    pattern: /hello/{name}
    defaults: { _controller: JuGebeyaBundle:Default:index }
```

```
Ju_Item:
    pattern: /
    defaults: { _controller: "JuGebeyaBundle:Item:index" }

Ju_Item_show:
    pattern: /{id}/show
    defaults: { _controller: "JuGebeyaBundle:Item:show" }

Ju_Item_new:
    pattern: /new
    defaults: { _controller: "JuGebeyaBundle:Item:new" }

Ju_Item_create:
    pattern: /create
    defaults: { _controller: "JuGebeyaBundle:Item:create" }
```

```

    requirements: { _method: post }

Ju_Item_edit:
    pattern: /{id}/edit
    defaults: { _controller: "JuGebeyaBundle:Item:edit" }

Ju_Item_update:
    pattern: /{id}/update
    defaults: { _controller: "JuGebeyaBundle:Item:update" }
    requirements: { _method: post|put }

Ju_Item_delete:
    pattern: /{id}/delete
    defaults: { _controller: "JuGebeyaBundle:Item:delete" }
    requirements: { _method: post|delete }

```

Let's have a closer look to the `Ju_Item_show` route. The pattern defined by the `Ju_Item_show` route acts like `/*/show` where the wildcard is given the name `id`. For the URL `/1/show`, the `id` variable gets a value of `1`, which is available for you to use in your controller. The `_controller` parameter is a special key that tells Symfony which controller/action should be executed when a URL matches this route, in our case it should execute the `showAction` from the `ItemController` in the `JuGebeyaBundle`.

The route parameters (e.g. `{id}`) are especially important because each is made available as an argument to the controller method.

## Routing Configuration in Dev Environment

The dev environment loads the `app/config/routing_dev.yml` file that contains the routes used by the **Web Debug Toolbar** (you already deleted the routes for the `AcmeDemoBundle` from `app/config/routing_dev.php` – see Day 1, **How to remove the AcmeDemoBundle**). This file loads, at the end, the main `routing.yml` configuration file.

## Route Customizations

For now, when you request the `/` URL in a browser, you will get a 404 Not Found error. That's because this URL does not match any routes defined. We have a `Ju_Gebeya_homepage` route that matches the `/hello/Gebeya` URL and sends us to the `DefaultController`, `index` action. Let's change it to match the `/` URL and to call the `index` action from the `ItemController`. To make the change, modify it to the following:

```

# ...
Ju_Gebeya_homepage:
    pattern: /
    defaults: { _controller: JuGebeyaBundle:Item:index }

```

Now, if you clear the cache and go to <http://Gebeya.local> from your browser, you will see the `Item` homepage. We can now change the link of the Gebeya logo in the layout to use the `Ju_Gebeya_homepage` route:

```
<!-- ... -->
<h1><a href="{{ path('Ju_Gebeya_homepage') }}">
  
</a></h1>
<!-- ... -->
```

For something a bit more involved, let's change the `Item` page URL to something more meaningful:

```
/Item/sensio-labs/paris-france/1/web-developer
```

Without knowing anything about Gebeya, and without looking at the page, you can understand from the URL that Sensio Labs is looking for a Web developer to work in Paris, France.

The following pattern matches such a URL:

```
/Item/{company}/{location}/{id}/{position}
```

Edit the `Ju_Item_show` route from the `Item.yml` file:

```
# ...

Ju_Item_show:
  pattern: /{company}/{location}/{id}/{position}
  defaults: { _controller: "JuGebeyaBundle:Item:show" }
```

Now, we need to pass all the parameters for the changed route for it to work:

```
<!-- ... -->
<a href="{{ path('Ju_Item_show', { 'id': entity.id, 'company': entity.company, 'location':
entity.location, 'position': entity.position }) }}">
  {{ entity.position }}
</a>
<!-- ... -->
```

If you have a look at generated URLs, they are not quite yet as we want them to be:

[http://Gebeya.local/app\\_dev.php/Item/Sensio](http://Gebeya.local/app_dev.php/Item/Sensio) Labs/Paris,France/1/Web Developer

We need to “slugify” the column values by replacing all non ASCII characters by a -. Open the `Item.php` file and add the following methods to the class:

```
// ...
use JuGebeyaBundleUtilsGebeya as Gebeya;

class Item
{
    // ...

    public function getCompanySlug()
    {
        return Gebeya::slugify($this->getCompany());
    }

    public function getPositionSlug()
    {
        return Gebeya::slugify($this->getPosition());
    }

    public function getLocationSlug()
    {
        return Gebeya::slugify($this->getLocation());
    }
}
```

You must also add the `use` statement before the `Item` class definition. After that, create the `src/Ju/GebeyaBundle/Utils/Gebeya.php` file and add the `slugify` method in it:

```
namespace JuGebeyaBundleUtils;

class Gebeya
{
    static public function slugify($text)
    {
        // replace all non letters or digits by -
        $text = preg_replace('/W+/', '-', $text);

        // trim and lowercase
        $text = strtolower(trim($text, '-'));

        return $text;
    }
}
```

We have defined three new “virtual”

**accessors:** `getCompanySlug()`, `getPositionSlug()`, and `getLocationSlug()`. They return their corresponding column value after applying it the `slugify()` method. Now, you can replace the real column names by these virtual ones in the template:

```
<!-- ... -->
<a href="{{ path('Ju_Item_show', { 'id': entity.id, 'company': entity.companyslug, 'location':
entity.locationslug, 'position': entity.positionslug}) }}">
    {{ entity.position }}
</a>
<!-- ... -->
```

## Route Requirements

The routing system has a built-in validation feature. Each pattern variable can be validated by a regular expression defined using the requirements entry of a route definition:

```
# ...
Ju_Item_show:
    pattern: /{company}/{location}/{id}/{position}
    defaults: { _controller: "JuGebeyaBundle:Item:show" }
    requirements:
        id: d+
# ...
```

The above requirements entry forces the id to be a numeric value. If not, the route won't match.

## Route Debugging

While adding and customizing routes, it's helpful to be able to visualize and get detailed information about your routes. A great way to see every route in your application is via the `router:debug` console command. Execute the command by running the following from the root of your project:

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

The command will print a helpful list of all the configured routes in your application. You can also get very specific information on a single route by including the route name after the command:

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

## Day 6: More with the Model

### The Doctrine Query Object

From the second day's requirements: *"On the itempage, the user sees the all available items sorted by Category and Latest First"*. But as of now, all items are displayed oldest first:

```
Src/Ju/GebeyaBundle/Controller/ItemController.php

public function indexAction()
{
    $em = $this->getDoctrine()->getManager();
    $entities = $em->getRepository('JuGebeyaBundle:Item')->findAll();
    return $this->render('JuGebeyaBundle:Item:index.html.twig', array(
        'entities' => $entities,
    ));
}
```

The `$entities = $em->getRepository('JuGebeyaBundle')->findAll()` method will make a request to the database to get all the items. We are not specifying any condition, which means that all the records are retrieved from the database.

Let's change it to display latest first:

```
Src/Ju/GebeyaBundle/Controller/ItemController.php

public function indexAction()
{
    $em = $this->getDoctrine()->getManager();
    $query = $em->createQuery('SELECT j FROM JuGebeyaBundle:Item j ORDER BY j.created_at
DESC');
    $entities = $query->getResult();
    return $this->render('JuGebeyaBundle:Item:index.html.twig', array(
        'entities' => $entities,
    ));
}
```

## Debugging Doctrine generated SQL

Sometimes, it is of great help to see the SQL generated by Doctrine; for instance, to debug a query that does not work as expected. In the dev environment, thanks to the Symfony Web Debug Toolbar, all the information you need is available within the comfort of your browser ([http://gebeya.local/app\\_dev.php](http://gebeya.local/app_dev.php)):



**Queries**

| # | Time    | Info   |
|---|---------|--|
| 1 | 0.26 ms | <pre>SELECT i0_.id AS id0, i0_.company AS company1, i0_.logo AS logo2, i0_.name AS name3, i0_.price AS price4, i0_.image AS image5, i0_.category_id AS category_id6, i0_.cart_id AS cart_id7 FROM item i0_ ORDER BY i0_.id DESC</pre> <p>Parameters: {}</p> <a href="#">View formatted query</a> <a href="#">View runnable query</a> <a href="#">Explain query</a> |

**Database Connections**

| Key     | Value                            |
|---------|----------------------------------|
| default | doctrine.dbal.default_connection |

**Entity Managers**

| Key     | Value                               |
|---------|-------------------------------------|
| default | doctrine.orm.default_entity_manager |

## Object Serialization

The above code relies on the `created_at` value to get latest items because this column stores the creation date. So, we can satisfy our requirement to get the latest item first.

But, if this value is not set in fixture file, it remains always empty. But when a `Item` is created, it can be automatically set to current date.

When you need to do something automatically before a Doctrine object is serialized to the database, you can add a new action to the `lifecycle` callbacks in the file that maps objects to the database. We have seen that on Day 3 for the `created_at` property but to remind you, the following is the callbacks.

```
src/Ju/GebeyaBundle/Resources/config/doctrine/Item.orm.yml
```

```
lifecycleCallbacks:
```

```
  prePersist: [ setCreatedAtValue ]
```

```
  preUpdate: [ setUpdatedAtValue ]
```

```
lifecycleCallbacks:
```

```
  prePersist: [ setCreatedAtValue, setUpdatedAtValue ]
```

```
  preUpdate: [ setUpdatedAtValue ]
```

Then, we have to rebuild the entities classes so Doctrine will add the new function:

```
php app/console doctrine:generate:entities JuGebeyaBundle
```

## Refactoring

Although the code we have written works fine, it's not quite right yet. Can you spot the problem?

The Doctrine query code does not belong to the action (the Controller layer), it belongs to the Model layer. In the MVC model, the Model defines all the business logic, and the Controller only calls the Model to retrieve data from it. As the code returns a collection of items, let's move the code to the model. For that we will need to create a custom repository class for Item entity and to add the query to that class.

Open `/src/JU/GebeyaBundle/Resources/config/doctrine/Item.orm.yml` and add the following to it:

```
/src/JU/GebeyaBundle/Resources/config/doctrine/Item.orm.yml  
  
Ju\GebeyaBundle\Entity\Item:  
    type: entity  
    table: item  
    repositoryClass: Ju\GebeyaBundle\Repository\ItemRepository
```

Doctrine can generate the repository class for you by running the `generate:entities` command used earlier:

```
php app/console doctrine:generate:entities JuGebeyaBundle
```

Next, add a new method - `getLatestItems()` - to the newly generated repository class. This method will query for all of the items sorted by the `created_at` column.

```
Src/Ju/GebeyaBndle/Repository/ItemRepository.php
```

```
<?php
```

```

namespace Ju\GebeyaBundle\Repository;
use Doctrine\ORM\EntityRepository;

/**
 * ItemRepository
 *
 * This class was generated by the Doctrine ORM. Add your own custom
 * repository methods below.
 */
class ItemRepository extends EntityRepository
{
    public function getLatestItems()
    {
        $qb = $this->createQueryBuilder('j')
            ->orderBy('j.created_at', 'DESC');
        $query = $qb->getQuery();
        return $query->getResult();
    }
}

```

Now the action code can use this new method to retrieve the latest items.

```

Src/Ju/GebeyaBundle/Controller/ItemController.php

public function indexAction()
{
    $em = $this->getDoctrine()->getManager();
    $entities = $em->getRepository('JuGebeyaBundle:Item')->getLatestItems();
    return $this->render('JuGebeyaBundle:Item:index.html.twig', array(
        'entities' => $entities,
    ));
}

```

This refactoring has several benefits over the previous code:

- The logic to get the latest item is now in the Model, where it belongs
- The code in the controller is thinner and much more readable
- The `getLatestItems()` method is re-usable (for instance in another action)
- The model code is now unit testable

## Categories on the Homepage

According to the second day's requirements we need to have categories with available items. Until now, we have not taken the item category into account. From the requirements, the homepage must display items by category. First, we need to get all categories with at least one item.

Create a repository class for the Category entity like we did for Item:

```
src/JU/GebeyaBundle/Resources/config/doctrine/Category.orm.yml

Ju\GebeyaBundle\Entity\Category:
    type: entity
    table: category
    repositoryClass: Ju\GebeyaBundle\Repository\CategoryRepository
```

Generate the repository class:

```
php app/console doctrine:generate:entities JuGebeyaBundle
```

Open the `CategoryRepository` class and add a `getWithItems()` method:

```
src/JU/GebeyaBundle/Repository/CategoryRepository.php

<?php

namespace Ju\GebeyaBundle\Repository;

use Doctrine\ORM\EntityRepository;

/**
 * CategoryRepository
 *
 * This class was generated by the Doctrine ORM. Add your own custom
 * repository methods below.
 */
class CategoryRepository extends EntityRepository
{
```

```

public function getWithItems()
{
    $query = $this->getEntityManager()->createQuery(
        'SELECT c FROM JuGebeyaBundle:Category c LEFT JOIN c.items j WHERE j.id > 0');

    return $query->getResult();
}
}

```

Change the index action accordingly:

Src/Ju/GebeyaBundle/Controller/DefaultController.php

```

public function indexAction()
{
    $em = $this->getDoctrine()->getManager();
    $items = $em->getRepository('JuGebeyaBundle:Item')->findAll();
    $categories = $em->getRepository('JuGebeyaBundle:Category')->getWithItems();
    $carts = $em->getRepository('JuGebeyaBundle:Cart')->findAll();
    return $this->render('JuGebeyaBundle:Default:index.html.twig', array(
        'items' => $items, 'categories' => $categories, 'carts' => $carts,
    ));
}

```

On the home page, you can check there is only Categories which have at least one Item are listed.

| CATEGORY       |   |
|----------------|---|
| FOODS          | + |
| MARS CHOCOLATE |   |
| BOTTLED WATER  | + |
| BOTTLED JUICE  | + |
| LAPTOP         | + |



## Best Sell Items

On the home page, There the user can also get Best Sell Items. To get that, We will check items in carts sold and count the most sold item. Add `getBestSell` method to `ItemRepository` class.

Src/Ju/GebeyaBndle/Repository/ItemRepository.php

```
public function getBestSell()
{
    $query = $this->createQueryBuilder('i')
        ->select('i, COUNT(c) AS nbrLikes')
        ->leftJoin('i.carts', 'c')
        ->where('c.id <> 0')
        ->where('c.itemOrders <> 0')
        ->orderBy('nbrLikes', 'DESC')
        ->groupBy('i.id')
        ->setMaxResults(9)
        ->getQuery();
    $ids_carts = $query->getResult();

    $items = array();
```

```

foreach($ids_carts as $index=>$value)
{
    $items[] = $value[0];
}
return $items;
}

```

Change the index action accordingly:

```

Src/Ju/GebeyaBundle/Controller/DefaultController.php

public function indexAction()
{
    $em = $this->getDoctrine()->getManager();
    $items = $em->getRepository('JuGebeyaBundle:Item')->getBestSell();
    $categories = $em->getRepository('JuGebeyaBundle:Category')->getWithItems();
    $carts = $em->getRepository('JuGebeyaBundle:Cart')->findAll();
    return $this->render('JuGebeyaBundle:Default:index.html.twig', array(
        'items' => $items, 'categories' => $categories, 'carts' => $carts,
    ));
}

```

## Day 7 Category Page

Today we will make the Category page like it is described in the second day's requirements:

"The user sees a list of all the Items from the category sorted by date and paginated with 10 Category per page"

### Embedding Other Twig Templates

Notice that we have redundant tag that create a list of Items from the category item in **views/Category/index.html.twig** and **views/Default/index.html.twig** templates. That's bad. When you need to reuse some portion of a template, you need to create a new twig template with that code and include it where you need. Create the `list.html.twig` file:

```
Src/src/Ju/GebeyaBundle/Resources/views/Category/list.html.twig
```

```
<div id="{{ i }}" class="panel-collapse collapse">
    <div class="panel-body">
        <ul>
            {% for item in category.items %}
                <li>
                    <a href="{{ path('item_show',{ 'id': item.id, 'company': item.companyslug , 'category':
item.categoryslug}) }}">{{ item.name }} </a>
                </li>
            {% endfor %}
        </ul>
    </div>
</div>
```

You can include/embed a template by using the embed function. Replace the HTML `<div>` code from both templates with the mentioned function:

```
Src/Ju/GebeyaBundle/Resources/views/Category/index.html.twig
```

```
{% embed "JuGebeyaBundle:Category:list.html.twig" %}
{% endembed %}
```

And

```
Src/Ju/GebeyaBundle/Resources/views/Default/index.html.twig
```

```
{% embed "JuGebeyaBundle:Category:list.html.twig" %}
{% endembed %}
```

## List Pagination

There are different Symfony2 pagination tools we can use to restrict number of categories per page but to solve this problem we will use the old classic method so that you can understand how it works. First, let's add a page parameter to the `ju_gebeya_category` route. The page parameter will have a default value of 1, so it will not be required:

```
src/Ju/GebeyaBundle/Resources/config/routing.yml
```



```
ju_gebeya_category:
  path: /{page}
  defaults: { _controller: "JuGebeyaBundle:Category:index" , page: 1}
  requirements:
    page: \d+
```

Clear the cache after modifying the routing file:

```
php app/console cache:clear --env=dev
php app/console cache:clear --env=prod
```

The number of category on each page will be defined as a custom parameter in the app/config/config.yml file:

```
app/config/config.yml

parameters:
  max_category_pre_page: 10
```

Change the CategoryRepository getAllCategories method to include an \$offset parameter to be used by doctrine when retrieving categories:

```
Src/Ju/GebeyaBundle/Repository/CategoryRepository.php

public function getAllCategories($max = null, $offset = null)
{
    $qb = $this->createQueryBuilder('c');
    if($max)
    {
        $qb->setMaxResults($max);
    }

    if($offset)
    {
        $qb->setFirstResult($offset);
    }

    $query = $qb->getQuery();

    return $query->getResult();
}
```

Change the CategoryController showAction to the following:

src/Ju/GebeyaBundle/Controller/CategoryController.php

```
public function indexAction($page)
{
    $em = $this->getDoctrine()->getManager();

    $total_categories = $em->getRepository('JuGebeyaBundle:Category')-
>countAllCategories();
    $categories_per_page = $this->container->getParameter('max_category_pre_page');
    $last_page = ceil($total_categories / $categories_per_page);
    $previous_page = $page > 1 ? $page - 1 : 1;
    $next_page = $page < $last_page ? $page + 1 : $last_page;
    $entities = $em->getRepository('JuGebeyaBundle:Category')-
>getAllCategories($categories_per_page, ($page - 1) * $categories_per_page);

    return $this->render('JuGebeyaBundle:Category:index.html.twig', array(
        'entities' => $entities,
        'last_page' => $last_page,
        'previous_page' => $previous_page,
        'current_page' => $page,
        'next_page' => $next_page,
        'total_items' => $total_categories
    ));
}
```

Finally, let's update the template

src/Ju/GebeyaBundle/Resources/views/Category/index.html.twig

```
{% extends 'JuGebeyaBundle::layout.html.twig' %}
{% block body -%}

    <div class="container">
        <div class="row">
            <div class="col-sm-10 padding-right">
                <h2>List of Categories</h2>
                <div class="panel-group category-products" id="accordion"><!--category-products-->
                    {% set i=0 %}
                    {% for category in entities %}
                        {% set i = i+1 %}
                        <div class="panel panel-default">
                            <div class="panel-heading">
```

```

        <h4 class="panel-title">
            <a data-toggle="collapse" data-parent="#accordion" href="#{{ i }}">
                {{ category.name }}
            </a>
            <span class="badge pull-right">
                <a class="btn btn-success" href="{{ path('category_show',
{id:category.id}) }}">More...</a>
                <a class="btn btn-success" href="{{ path('category_show',
{id:category.id}) }}">Edit</a>
            </span>
        </h4>
    </div>

    {% embed "JuGebeyaBundle:Category:list.html.twig" %}
    {% endembed %}
</div>
{% endfor %}
<div>
    <a class="btn btn-primary" href="{{ path('category_new') }}">Create new entry</a>
</div>
{% if last_page > 1 %}
    <div>
        <ul class="pagination pagination-lg pull-right">
            <li><a href="{{ path('category',{'page':1}) }}">«</a></li>
            <li><a href="{{ path('category',{'page':previous_page}) }}"><<</a></li>
            <li><a>{{ current_page }}</a></li>
            {% if last_page >= current_page+1 %}
                <li><a href="{{ path('category',
{'page':current_page+1}) }}">{{ current_page+1 }}</a></li>
            {% endif %}
            {% if last_page >= current_page+2 %}
                <li><a href="{{ path('category',
{'page':current_page+2}) }}">{{ current_page+2 }}</a></li>
            {% endif %}
            {% if last_page >= current_page+3 %}
                <li><a href="{{ path('category',

```

```

{{'page':current_page+3}}}}">{{current_page+3}}</a></li>
    {% endif %}
    {% if last_page >= current_page+4 %}
        <li><a href="{{path('category',
{{'page':current_page+4}}}}">{{current_page+4}}</a></li>
    {% endif %}
        <li><a href="{{path('category',{'page':next_page}})}">></a></li>
        <li><a href="{{path('category',{'page':last_page}})}">></a></li>
    </ul>
</div>
{% endif %}
</div><!--/category-products-->
</div>
</div>
</div>
{% endblock %}

```

The result:

|                  |         |      |
|------------------|---------|------|
| „J[PL KDS        | MORE... | EDIT |
| ASJDASD          | MORE... | EDIT |
| BOTTLED JUICE    | MORE... | EDIT |
| BOTTLED WATER    | MORE... | EDIT |
| DESKTOP COMPUTER | MORE... | EDIT |
| FOODS            | MORE... | EDIT |
| IDOGHJDFIOBN     | MORE... | EDIT |
| KP[JHOTREK       | MORE... | EDIT |
| KPOHTLREMW       | MORE... | EDIT |
| KWJEHOL          | MORE... | EDIT |
| Create new entry |         |      |
| « < 1 2 > »      |         |      |

## Day 8: The Unit Tests

### Tests in Symfony

There are two different kinds of automated tests in Symfony: **unit tests** and **functional tests**. Unit tests verify that each method and function is working properly. Each test must be as independent as possible from the others. On the other hand, functional tests verify that the resulting application behaves correctly as a whole.

Unit tests will be covered in this day, whereas the next day will be dedicated to functional tests.

Symfony2 integrates with an independent library, the **PHPUnit**, to give you a rich testing framework. To run tests, you will have to install PHPUnit 3.5.11 or later.

If you don't have PHPUnit installed, use the following to get it:

```
sudo apt-get install phpunit
sudo pear channel-discover pear.phpunit.de
sudo pear channel-discover pear.symfony-project.com
```

```
sudo pear channel-discover components.ez.no
sudo pear channel-discover pear.symfony.com
sudo pear update-channels
sudo pear upgrade-all
sudo pear install pear.symfony.com/Yaml
sudo pear install --alldeps phpunit/PHPUnit
sudo pear install --force --alldeps phpunit/PHPUnit
```

Each test – whether it’s a unit test or a functional test – is a PHP class that should live in the Tests/ subdirectory of your bundles. If you follow this rule, then you can run all of your application’s tests with the following command:

```
phpunit -c app/
```

The -c option tells PHPUnit to look in the app/ directory for a configuration file. If you’re curious about the PHPUnit options, check out the app/phpunit.xml.dist file.

A unit test is usually a test against a specific PHP class. By convention, the Tests/ subdirectory should replicate the directory of your bundle. So, when we are testing a class in our bundle’s Utils/ directory, we put the test in the Tests/Utils/ directory:

```
src/Ju/GebeyaBundle/Tests/Utils/GebeyaTest.php
<?php
/**
 * Created by PhpStorm.
 * User: solomon
 * Date: 3/16/16
 * Time: 3:11 AM
 */
namespace Ju\GebeyaBundle\Tests\Utils;
use Ju\GebeyaBundle\Utils\Gebeya;
class GebeyaTest extends \PHPUnit_Framework_TestCase
{
```

```

public function testSlugify()
{
    $this->assertEquals('rani', Gebeya::slugify('Rani'));
    $this->assertEquals('rani-ltd', Gebeya::slugify('rani ltd'));
    $this->assertEquals('rani-ltd', Gebeya::slugify('Rani ltd'));
    $this->assertEquals('rani-ltd', Gebeya::slugify('rani,ltd'));
    $this->assertEquals('rani', Gebeya::slugify(' rani'));
    $this->assertEquals('rani', Gebeya::slugify('rani '));
}
}

```

To run only this test, you can use the following command:

```
phpunit -c app/ src/Ju/GebeyaBundle/Tests/Utils/GebeyaTest.php
```

As everything should work fine, you should get the following result:

```

PHPUnit 3.7.28 by Sebastian Bergmann.

Configuration read from /var/www/gebeya2/app/phpunit.xml.dist
.

Time: 350 ms, Memory: 4.25Mb

OK (1 test, 6 assertions)

```

For a full list of [assertions](#), you can check the PHPUnit documentation.

### Adding Tests for new Features

The slug for an empty string is an empty string. You can test it, it will work. But an empty string in a URL is not that a great idea. Let's change the `slugify()` method so that it returns the "n-a" string in case of an empty string.

You can write the test first, then update the method, or the other way around. It is really a matter of taste, but writing the test first gives you the confidence that your code actually implements what you planned:

```
src/Ju/GebeyaBundle/Tests/Utils/GebeyaTest.php
<?php
//...
$this->assertEquals('n-a', Gebeya::slugify(''));
//...
```

Now, if we run the test again, we will have a failure:

```
PHPUnit 3.7.28 by Sebastian Bergmann.

Configuration read from /var/www/gebeya2/app/phpunit.xml.dist

F

Time: 79 ms, Memory: 4.25Mb

There was 1 failure:

1) Ju\GebeyaBundle\Tests\Utils\ItemTest::testSlugify
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'n-a'
+'''

/var/www/gebeya2/src/Ju/GebeyaBundle/Tests/Utils/GebeyaTest.php:23
```



**FAILURES!**

**Tests: 1, Assertions: 7, Failures: 1.**

Now, edit the `Item::slugify` method and add the following condition at the beginning:

```
src/Ju/GebeyaBundle/Utils/GebeyaTest.php
//...
static public function slugify($text)
{
    if (empty($text)) {
        return 'n-a';
    }
    //...
}
```

The test must now pass as expected, and you can enjoy the green bar.

### Adding Tests because of a Bug

Let's say that time has passed and one of your users reports a weird bug: some item links point to a 404 error page. After some investigation, you find that for some reason, these items have an empty company or category slug.

How is it possible?

You look through the records in the database and the columns are definitely not empty. You think about it for a while, and bingo, you find the cause. When a string only contains non-ASCII characters, the `slugify()` method converts it to an empty string. So happy to have found the cause, you open the `Gebeya` class and fix the problem right away. That's a bad idea. First, let's add a test:

```
src/Ju/GebeyaBundle/Tests/Utils/GebeyaTest.php
<?php
//...
```

```
$this->assertEquals('n-a', Gebeya::slugify(' - '));  
//...
```

After checking that the test does not pass, edit the Gebeya class and move the empty string check to the end of the method:

```
src/Ju/GebeyaBundle/Utils/GebeyaTest.php  
//...  
  
static public function slugify($text)  
{  
    if (empty($text)) {  
        return 'n-a';  
    }  
    return $text;  
}
```

The new test now passes, as do all the other ones. The slugify() had a bug despite our 100% coverage.

You cannot think about all edge cases when writing tests, and that's fine. But when you discover one, you need to write a test for it before fixing your code. It also means that your code will get better over time, which is always a good thing.

## Code Coverage

When you write tests, it is easy to forget a portion of the code. If you add a new feature or you just want to verify your code coverage statistics, all you need to do is to check the code coverage by using the `--coverage-html` option:

```
phpunit --coverage-html=web/cov/ -c app/
```

Check the code coverage by opening the generated <http://gebeya.local/cov/index.html> page in a browser.

The code coverage only works if you have XDebug enabled and all dependencies installed.

```
sudo apt-get install php5-xdebug
```

## Doctrine Unit Tests

Unit testing a Doctrine model class is a bit more complex as it requires a database connection. You already have the one you use for your development, but it is a good habit to create a dedicated database for tests.

At the beginning of this tutorial, we introduced the environments as a way to vary an application's settings. By default, all symfony tests are run in the test environment, so let's configure a different database for the test environment:

Go to your app/config directory and create a copy of parameters.yml file, called parameters\_test.yml. Open parameters\_test.yml and change the name of your database to gebeya\_test. For this to be imported, we have to add it in the config\_test.yml file :

```
app/config/config_test.yml
imports:
    - { resource: config_dev.yml }
    - { resource: parameters_test.yml }
```

## Testing the Item Entity

First, we need to create the ItemTest.php file in the Tests/Entity folder.

The setUp function will manipulate your database each time you will run the test. At first, it will drop your current database, then it will re-create it and load data from fixtures in it. This will help you have the same initial data in the database you created for the test environment before running the tests.

```
src/Ju/GebeyaBundle/Tests/Entity/ItemTest.php

<?php
namespace Ju\GebeyaBundle\Entity;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;
use Symfony\Bundle\FrameworkBundle\Console\Application;
use Symfony\Component\Console\Output\NullOutput;
use Symfony\Component\Console\Input\ArrayInput;
use Doctrine\Bundle\DoctrineBundle\Command\DropDatabaseDoctrineCommand;
use Doctrine\Bundle\DoctrineBundle\Command\CreateDatabaseDoctrineCommand;
use Doctrine\Bundle\DoctrineBundle\Command\Proxy\CreateSchemaDoctrineCommand;

class ItemTest extends WebTestCase
{
    private $em;
```

```

private $application;

public function setUp()
{
    static::$kernel = static::createKernel();
    static::$kernel->boot();

    $this->application = new Application(static::$kernel);

    // drop the database
    $command = new DropDatabaseDoctrineCommand();
    $this->application->add($command);
    $input = new ArrayInput(array(
        'command' => 'doctrine:database:drop',
        '--force' => true
    ));
    $command->run($input, new NullOutput());

    // we have to close the connection after dropping the database so we don't get "No
    database selected" error
    $connection = $this->application->getKernel()->getContainer()->get('doctrine')-
>getConnection();
    if ($connection->isConnected()) {
        $connection->close();
    }

    // create the database
    $command = new CreateDatabaseDoctrineCommand();
    $this->application->add($command);
    $input = new ArrayInput(array(
        'command' => 'doctrine:database:create',
    ));
    $command->run($input, new NullOutput());

    // create schema
    $command = new CreateSchemaDoctrineCommand();
    $this->application->add($command);
    $input = new ArrayInput(array(
        'command' => 'doctrine:schema:create',
    ));
    $command->run($input, new NullOutput());

    // get the Entity Manager
    $this->em = static::$kernel->getContainer()
->get('doctrine')
->getManager();

    // load fixtures
    $client = static::createClient();
    $loader = new \Symfony\Bridge\Doctrine\DataFixtures\ContainerAwareLoader($client-
>getContainer());
    $loader->loadFromDirectory(static::$kernel-

```

```

>locateResource('@JuGebeyaBundle/DataFixtures/ORM'));
    $purger = new \Doctrine\Common\DataFixtures\Purger\ORMPurger($this->em);
    $executor = new \Doctrine\Common\DataFixtures\Executor\ORMExecutor($this->em,
$purger);
    $executor->execute($loader->getFixtures());
}

protected function tearDown()
{
    parent::tearDown();
    $this->em->close();
}
}

```

## Testing the Repository Classes

Now, let's write some tests for the CategoryRepository class, to see if the functions we created in the previous days are returning the right values:

```

src/Ju/GebeyaBundle/Tests/Repository/CategoryRepositoryTest.php

<?php
namespace Ju\GebeyaBundle\Tests\Repository;
use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;
use Symfony\Bundle\FrameworkBundle\Console\Application;
use Symfony\Component\Console\Output\NullOutput;
use Symfony\Component\Console\Input\ArrayInput;
use Doctrine\Bundle\DoctrineBundle\Command\DropDatabaseDoctrineCommand;
use Doctrine\Bundle\DoctrineBundle\Command\CreateDatabaseDoctrineCommand;
use Doctrine\Bundle\DoctrineBundle\Command\Proxy\CreateSchemaDoctrineCommand;
class CategoryRepositoryTest extends WebTestCase
{
    private $em;
    private $application;
    public function setUp()
    {
        static::$kernel = static::createKernel();
        static::$kernel->boot();
        $this->application = new Application(static::$kernel);
    }
}

```

```

// drop the database
$command = new DropDatabaseDoctrineCommand();
$this->application->add($command);
$input = new ArrayInput(array(
    'command' => 'doctrine:database:drop',
    '--force' => true
));
$command->run($input, new NullOutput());

// we have to close the connection after dropping the database so we don't get "No
database selected" error
$connection = $this->application->getKernel()->getContainer()->get('doctrine')-
>getConnection();
if ($connection->isConnected()) {
    $connection->close();
}

// create the database
$command = new CreateDatabaseDoctrineCommand();
$this->application->add($command);
$input = new ArrayInput(array(
    'command' => 'doctrine:database:create',
));
$command->run($input, new NullOutput());

// create schema
$command = new CreateSchemaDoctrineCommand();
$this->application->add($command);
$input = new ArrayInput(array(
    'command' => 'doctrine:schema:create',
));
$command->run($input, new NullOutput());

// get the Entity Manager
$this->em = static::$kernel->getContainer()
->get('doctrine')

```

```

        ->getManager();

        // load fixtures

        $client = static::createClient();

        $loader = new \Symfony\Bridge\Doctrine\DataFixtures\ContainerAwareLoader($client-
>getContainer());

        $loader->loadFromDirectory(static::$kernel-
>locateResource('@JuGebeyaBundle/DataFixtures/ORM'));

        $purger = new \Doctrine\Common\DataFixtures\Purger\ORMPurger($this->em);

        $executor = new \Doctrine\Common\DataFixtures\Executor\ORMExecutor($this->em,
        $purger);

        $executor->execute($loader->getFixtures());
    }

    public function testCountAllCategories()
    {
        $query = $this->em->createQuery('SELECT COUNT(c.id) FROM
        JuGebeyaBundle:Category c');

        $categories = $query->getSingleScalarResult();

        $categories_db = $this->em->getRepository('JuGebeyaBundle:Category')-
>countAllCategories();

        // This test will verify if the value returned by the countAllCategories() function
        // coincides with the number of categories in the database

        $this->assertEquals($categories, $categories_db);
    }

    public function testGetWithItems()
    {
        $query = $this->em->createQuery('SELECT c FROM JuGebeyaBundle:Category c LEFT
        JOIN c.items j WHERE j.id > 0');

        $categories = $query->getResult();

        $categories_db = $this->em->getRepository('JuGebeyaBundle:Category')-
>getWithItems();

        $this->assertEquals($categories, $categories_db);
    }

    public function testGetAllCategories()
    {

```

```

$query = $this->em->createQuery('SELECT c FROM JuGebeyaBundle:Category c');
$categories = $query->getResult();

$categories_db = $this->em->getRepository('JuGebeyaBundle:Category')->findAll();
// This test will verify if the value returned by the countAllCategories() function
// coincides with the number of categories in the database
$this->assertEquals($categories, $categories_db);
}

protected function tearDown()
{
    parent::tearDown();
    $this->em->close();
}
}

```

After you finish writing the tests, run them with the following command, in order to generate the code coverage percent for the whole functions :

```
phpunit --coverage-html=web/cov/ -c app src/Ju/GebeyaBundle/Tests/Repository/
```

To Test only for Category, type the following

```
phpunit --coverage-html=web/cov/ -c app
src/Ju/GebeyaBundle/Tests/Repository/CategoryRepositoryTest.php
```

You will see a resulting

```

PHPUnit 3.7.28 by Sebastian Bergmann.

Configuration read from /var/www/gebeya2/app/phpunit.xml.dist

..
Time: 8.01 seconds, Memory: 24.50Mb

OK (3 tests, 3 assertions)
















```



Now, if you go

to [http://gebeya.local/cov/Ju\\_GebeyaBundle\\_Repository\\_CategoryRepository.php.html](http://gebeya.local/cov/Ju_GebeyaBundle_Repository_CategoryRepository.php.html) you will see that the code coverage for Repository Tests is not 100% complete.

/var/www/gebeya2/src / Ju / GebeyaBundle / Repository / CategoryRepository.php

|   | Code Coverage   |         |       |   |         |       |              |   |         |        |
|---|---|---------|-------|---|---------|-------|--------------|---|---------|--------|
|   | Classes and Traits  |         |       | Functions and Methods   |         |       |              | Lines   |         |        |
| Total   |  | 0.00%   | 0 / 1 |    | 66.67%  | 2 / 3 | CRAP<br>9.45 |  | 43.75%  | 7 / 16 |
| CategoryRepository                              |  | 0.00%   | 0 / 1 |    | 66.67%  | 2 / 3 | 9.45         |  | 43.75%  | 7 / 16 |
| getWithItems()                                  |  | 100.00% | 1 / 1 |  | 100.00% | 1 / 1 | 1            |  | 100.00% | 3 / 3  |
| getAllCategories(\$max = null, \$offset = null) |  | 0.00%   | 0 / 1 |  | 0.00%   | 0 / 1 | 12           |  | 0.00%   | 0 / 9  |
| countAllCategories()                            |  | 100.00% | 1 / 1 |  | 100.00% | 1 / 1 | 1            |  | 100.00% | 4 / 4  |

That's all for today! See you tomorrow, when we will talk about functional tests.

## Day 9: The Functional Tests

Functional tests are a great tool to test your application from end to end: from the request made by a browser to the response sent by the server. They test all the layers of an application: **the routing, the model, the actions and the templates**. They are very similar to what you probably already do manually: each time you add or modify an action, you need to go to the browser and check that everything works as expected by clicking on links and checking elements on the rendered page. In other words, you run a scenario corresponding to the use case you have just implemented.

As the process is manual, it is tedious and error prone. Each time you change something in your code, you must step through all the scenarios to ensure that you did not break something. That's insane. Functional tests in symfony provide a way to easily describe scenarios. Each scenario can then be played automatically over and over again by simulating the experience a user has in a browser. Like unit tests, they give you the confidence to code in peace.

Functional tests have a very specific workflow:

- Make a request;
- Test the response;
- Click on a link or submit a form;
- Test the response;
- Rinse and repeat;

## Our First Functional Test

Functional tests are simple PHP files that typically live in the Tests/Controller directory of your bundle. If you want to test the pages handled by your CategoryController class, start by creating a new CategoryControllerTest class that extends a special WebTestCase class:

```
src/Ju/GebeyaBundle/Tests/Controller/CategoryControllerTest.php

<?php
namespace Ju\GebeyaBundle\Tests\Controller;
use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;
class CategoryControllerTest extends WebTestCase
{
    private $em;
    private $application;
    public function setUp()
    {
        static::$kernel = static::createKernel();
        static::$kernel->boot();
        $this->application = new Application(static::$kernel);
        // drop the database
        $command = new DropDatabaseDoctrineCommand();
        $this->application->add($command);
        $input = new ArrayInput(array(
            'command' => 'doctrine:database:drop',
            '--force' => true
        ));
        $command->run($input, new NullOutput());
        // we have to close the connection after dropping the database so we don't get "No
        database selected" error
        $connection = $this->application->getKernel()->getContainer()->get('doctrine')-
        >getConnection();
        if ($connection->isConnected()) {
            $connection->close();
        }
    }
}
```

```

    }

    // create the database
    $command = new CreateDatabaseDoctrineCommand();
    $this->application->add($command);
    $input = new ArrayInput(array(
        'command' => 'doctrine:database:create',
    ));
    $command->run($input, new NullOutput());

    // create schema
    $command = new CreateSchemaDoctrineCommand();
    $this->application->add($command);
    $input = new ArrayInput(array(
        'command' => 'doctrine:schema:create',
    ));
    $command->run($input, new NullOutput());

    // get the Entity Manager
    $this->em = static::$kernel->getContainer()
        ->get('doctrine')
        ->getManager();

    // load fixtures
    $client = static::createClient();

    $loader = new \Symfony\Bridge\Doctrine\DataFixtures\ContainerAwareLoader($client->getContainer());
    $loader->loadFromDirectory(static::$kernel->locateResource('@JuGebeyaBundle/DataFixtures/ORM'));

    $purger = new \Doctrine\Common\DataFixtures\Purger\ORMPurger($this->em);
    $executor = new \Doctrine\Common\DataFixtures\Executor\ORMExecutor($this->em, $purger);
    $executor->execute($loader->getFixtures());
}

public function testShow()
{

```

```

$client = static::createClient();
$crawler = $client->request('GET', '/category/index');

$this->assertEquals('Ju\GebeyaBundle\Controller\CategoryController::showAction',
$client->getRequest()->attributes->get('_controller'));

$this->assertTrue(200 === $client->getResponse()->getStatusCode());
}
}

```

To learn more about crawler, read the Symfony documentation [here](#).

## Running Functional Tests

As for unit tests, launching functional tests can be done by executing the `phpunit` command:

```
phpunit -c app/ src/Ju/GebeyaBundle/Tests/Controller/CategoryControllerTest
```

This test will fail because the tested url, `/category/index`, is not a valid url in Gebeya Project:

```
PHPUnit 3.7.28 by Sebastian Bergmann.
```

```
Configuration read from /var/www/gebeya2/app/phpunit.xml.dist
```

```
F
```

```
Time: 3.45 seconds, Memory: 27.50Mb
```

```
There was 1 failure:
```

```
1) Ju\GebeyaBundle\Tests\Controller\CategoryControllerTest::testShow
```

```
Failed asserting that null matches expected
```

```
'Ju\GebeyaBundle\Controller\CategoryController::showAction'.
```

```
/var/www/gebeya2/src/Ju/GebeyaBundle/Tests/Controller/CategoryControllerTest.php:75
```

**FAILURES!**

**Tests: 1, Assertions: 1, Failures: 1.**

## The Right Way

```
public function testShow()
{
    $client = static::createClient();
    $crawler = $client->request('GET', '/category/ONE_CATEGORY_ID/show');
    $this->assertEquals('Ju\GebeyaBundle\Controller\CategoryController::showAction',
        $client->getRequest()->attributes->get('_controller'));
    $this->assertTrue(200 === $client->getResponse()->getStatusCode());
}
```

## Writing Functional Tests

Writing functional tests is like playing a scenario in a browser. We already have written all the scenarios we need to test as part of the day 2 stories.

First, let's test the Item page by editing the ItemControllerTest class. Replace the code with the following one:

### Only N Items are Listed on the Item Page

Add the following code at the end of your `testIndex()` function. To get the custom parameter defined in `app/config/config.yml` in our functional test, we will use the kernel:

```
Src/Ju/GebeyaBundle/Tests/Controller/ItemControllerTest.php
```

```
<?php
namespace Ju\GebeyaBundle\Tests\Controller;
use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;
use Symfony\Bundle\FrameworkBundle\Console\Application;
```

```

use Symfony\Component\Console\Output\NullOutput;
use Symfony\Component\Console\Input\ArrayInput;
use Doctrine\Bundle\DoctrineBundle\Command\DropDatabaseDoctrineCommand;
use Doctrine\Bundle\DoctrineBundle\Command\CreateDatabaseDoctrineCommand;
use Doctrine\Bundle\DoctrineBundle\Command\Proxy\CreateSchemaDoctrineCommand;
class ItemControllerTest extends WebTestCase
{
    private $em;
    private $application;
    public function setUp()
    {
        static::$kernel = static::createKernel();
        static::$kernel->boot();
        $this->application = new Application(static::$kernel);
        // drop the database
        $command = new DropDatabaseDoctrineCommand();
        $this->application->add($command);
        $input = new ArrayInput(array(
            'command' => 'doctrine:database:drop',
            '--force' => true
        ));
        $command->run($input, new NullOutput());
        // we have to close the connection after dropping the database so we don't get "No
        database selected" error
        $connection = $this->application->getKernel()->getContainer()->get('doctrine')-
        >getConnection();
        if ($connection->isConnected()) {
            $connection->close();
        }
        // create the database
        $command = new CreateDatabaseDoctrineCommand();
        $this->application->add($command);
    }
}

```

```

$input = new ArrayInput(array(
    'command' => 'doctrine:database:create',
));
$command->run($input, new NullOutput());
// create schema
$command = new CreateSchemaDoctrineCommand();
$this->application->add($command);
$input = new ArrayInput(array(
    'command' => 'doctrine:schema:create',
));
$command->run($input, new NullOutput());
// get the Entity Manager
$this->em = static::$kernel->getContainer()
    ->get('doctrine')
    ->getManager();
// load fixtures
$client = static::createClient();
$loader = new \Symfony\Bridge\Doctrine\DataFixtures\ContainerAwareLoader($client->getContainer());
$loader->loadFromDirectory(static::$kernel->locateResource('@JuGebeyaBundle/DataFixtures/ORM'));
$purger = new \Doctrine\Common\DataFixtures\Purger\ORMPurger($this->em);
$executor = new \Doctrine\Common\DataFixtures\Executor\ORMExecutor($this->em, $purger);
$executor->execute($loader->getFixtures());
}

public function testIndex()
{
    $client = static::createClient();
    $crawler = $client->request('GET', '/item/');

    $this->assertEquals('JuGebeyaBundle\Controller\ItemController::indexAction', $client->getRequest()->attributes->get('_controller'));

    $this->assertTrue($crawler->filter('.col-sm-3:contains("Edit Item")')->count() <= 12);
}

```

```
}  
}
```

## ITEMS ARE SORTED BY DATE

To test if items are actually sorted by date, we need to check that the first item listed on the item page is the one we expect. This can be done by checking that the URL contains the expected primary key. As the primary key can change between runs, we need to get the Doctrine object from the database first.

Src/Ju/GebeyaBundle/Tests/Controller/ItemControllerTest.php

```
public function testIndex()  
{  
    $client = static::createClient();  
    $crawler = $client->request('GET', '/item/');  
    $kernel = static::createKernel();  
    $kernel->boot();  
    $max_item_per_page = $kernel->getContainer()->getParameter('max_item_per_page');  
    $em = $kernel->getContainer()->get('doctrine.orm.entity_manager');  
    $query = $em->createQuery('SELECT i from JuGebeyaBundle:Item i ORDER BY  
i.created_at DESC');  
    $query->setMaxResults(1);  
    $item = $query->getSingleResult();  
    $this->assertTrue($crawler->filter('col-sm-3 div')->first()->filter(sprintf('a[href*="/%d/"]', $item->getId()))->count()==1);  
    $this->assertEquals('Ju\GebeyaBundle\Controller\ItemController::indexAction', $client->getRequest()->attributes->get('_controller'));  
    $this->assertTrue($crawler->filter('col-sm-3:contains("Edit Item")')->count() <=  
$max_item_per_page);  
}
```

Even if the test works in this very moment, we need to refactor the code a bit, as getting the first item of the can be reused elsewhere in our tests. We won't move the code to the Model layer as the code is test specific. Instead, we will move the code to the `getMostRecentItem` function in our test class:



Src/Ju/GebeyaBundle/Tests/Controller/ItemControllerTest.php

```
public function getMostRecentItem(){
    $kernel = static::createKernel();
    $kernel->boot();
    $em = $kernel->getContainer()->get('doctrine.orm.entity_manager');
    $query = $em->createQuery('SELECT i from JuGebeyaBundle:Item i ORDER BY
i.created_at DESC');
    $query->setMaxResults(1);
    return $query->getSingleResult();
}
```

You can now replace the previous test code by the following one:

Src/Ju/GebeyaBundle/Tests/Controller/ItemControllerTest.php

```
public function testIndex(){
    #...
    $this->assertTrue($crawler->filter('.col-sm-3 div')->first()->filter(sprintf('a[href*="/%d/"]',
    $this->getMostRecentItem()->getId()))->count()==1);
}
```

## Day 10: The Forms

Any website has forms, from the simple contact form to the complex ones with lots of fields. Writing forms is also one of the most complex and tedious task for a web developer: you need to write the HTML form, implement validation rules for each field, process the values to store them in a database, display error messages, repopulate fields in case of errors and much more ...

In Day 3 of this tutorial we used the `doctrine:generate:crud` command to generate a simple CRUD controller for our `Item` entity. This also generated a `Item` form that you can find in `/src/Ju/GebeyaBundle/Form/ItemType.php` file.

### Customizing the Item Form

By default, the Doctrine generated form displays fields for all the table columns. But for the `Item` form, some of them must not be editable by the end user. Edit the `Item` form as you see below:

src/Ju/GebeyaBundle/Form/ItemType.php

```
<?php
namespace Ju\GebeyaBundle\Form;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolverInterface;
class ItemType extends AbstractType
{
    /**
     * @param FormBuilderInterface $builder
     * @param array $options
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('category')
            ->add('company')
            ->add('logo')
            ->add('name')
            ->add('amount')
            ->add('price')
        ;
    }

    /**
     * @param OptionsResolverInterface $resolver
     */
    public function setDefaultOptions(OptionsResolverInterface $resolver)
    {
        $resolver->setDefaults(array(
            'data_class' => 'Ju\GebeyaBundle\Entity\Item'
        ));
    }
}
```

```

    ));
}
/**
 * @return string
 */
public function getName()
{
    return 'ju_gebeyabundle_item';
}
}

```

The form configuration must sometimes be more precise than what can be introspected from the database schema. For example, the name column can't be blank. In Symfony2, validation is applied to the underlying object (e.g. Item). In other words, the question isn't whether the form is valid, but whether or not the Item object is valid after the form has applied the submitted data to it. To do this, create a new validation.yml file in the Resources/config directory of our bundle:

Src/Ju/GebeyaBundle/Resources/config/validation.yml

```

Ju\GebeyaBundle\Entity\Item:
    properties:
        name:
            - NotBlank: ~

```

When you want to add any validation to the object you can edit the validation.yml file. After modifying validation.yml, you need to clear the cache.

For each field, symfony automatically generates a label (which will be used in the rendered tag). This can be changed with the label option:

src/Ju/GebeyaBundle/Form/ItemType.php

```

public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder

```

```

->add('category')
->add('company')
->add('logo', 'text', array('label' => 'Company Logo'))
->add('name')
->add('amount', 'number', array('label' => 'Unit Price'))
->add('price')

;
}

```

Handling File Uploads in Symfony2

## The Form Template

Now that the form class has been customized, we need to display it. Open the `new.html.twig` template and edit it:

Src/JuGebeyaBundle/Resources/views/Item/new.html.twig

```

{% extends 'JuGebeyaBundle::layout.html.twig' %}
{% block body -%}
<div class="container">
  <div class="row">
    <div class="col-sm-12 padding-right">
      <div class="features_items">
        <h2 class="title text-center">Item creation</h2>
        <div class="col-sm-6 col-lg-push-3">
          {{ form_start(form) }}
          <table class="table bootstrap-datatable countries">
            <tbody>
              <tr>
                <div class="form-group">
                  <th>{{ form_label(form.category) }}</th>
                  <td>
                    {{ form_errors(form.category) }}

```

```

        {{ form_widget(form.category) }}
    </td>
</div>
</tr>
<tr>
    <div class="form-group">
        <th>{{ form_label(form.company) }}</th>
        <td>
            {{ form_errors(form.company) }}
            {{ form_widget(form.company) }}
        </td>
    </div>
</tr>
<tr>
    <div class="form-group">
        <th>{{ form_label(form.logo) }}</th>
        <td>
            {{ form_errors(form.logo) }}
            {{ form_widget(form.logo) }}
        </td>
    </div>
</tr>
<tr>
    <div class="form-group">
        <th>{{ form_label(form.name) }}</th>
        <td>
            {{ form_errors(form.name) }}
            {{ form_widget(form.name) }}
        </td>
    </div>
</tr>

```

```

        <tr>
            <div class="form-group">
                <th>{{ form_label(form.price) }}</th>
                <td>
                    {{ form_errors(form.price) }}
                    {{ form_widget(form.price) }}
                </td>
            </div>
        </tr>
        <tr>
            <div class="form-group">
                <th>{{ form_label(form.amount) }}</th>
                <td>
                    {{ form_errors(form.amount) }}
                    {{ form_widget(form.amount) }}
                </td>
            </div>
        </tr>
    </tbody>
</table>
    {{ form_row(form.submit) }}
    <a href="{{ path('item') }}">
        Back to the list
    </a>
    {{ form_end(form) }}
</div>
</div>
</div>
</div>
</div>
{% endblock %}

```

We could render the form by just using the following line of code, but as we need more customization, we choose to render each form field by hand.

```
{{ form(form) }}
```

By printing `form(form)`, each field in the form is rendered, along with a label and error message (if there is one). As easy as this is, it's not very flexible (yet). Usually, you'll want to render each form field individually so you can control how the form looks.

We also used a technique named [form theming](#) to customize how the [form errors](#) will be rendered. You can read more about this in the official [Symfony2 documentation](#).

Do the same thing with the `edit.html.twig` template:

Src/JuGebeyaBundle/Resources/views/Item/new.html.twig

```
{% extends 'JuGebeyaBundle::layout.html.twig' %}
{% block body -%}
    <div class="container">
        <div class="row">
            <div class="col-sm-12 padding-right">
                <div class="features_items">
                    <h2 class="title text-center">Edit Item</h2>
                    <div class="col-sm-6 col-lg-push-3">
                        {{ form_start(edit_form) }}
                        <table class="table bootstrap-datable countries">
                            <tbody>
                                <tr>
                                    <div class="form-group">
                                        <th>{{ form_label(edit_form.category) }}</th>
                                        <td>
                                            {{ form_errors(edit_form.category) }}
                                            {{ form_widget(edit_form.category) }}
                                        </td>
                                    </div>
                                </tr>
                            </tbody>
                        </table>
                    </div>
                </div>
            </div>
        </div>
    </div>
```

```

</tr>
<tr>
  <div class="form-group">
    <th>{{ form_label(edit_form.company) }}</th>
    <td>
      {{ form_errors(edit_form.company) }}
      {{ form_widget(edit_form.company) }}
    </td>
  </div>
</tr>
<tr>
  <div class="form-group">
    <th>{{ form_label(edit_form.logo) }}</th>
    <td>
      {{ form_errors(edit_form.logo) }}
      {{ form_widget(edit_form.logo) }}
    </td>
  </div>
</tr>
<tr>
  <div class="form-group">
    <th>{{ form_label(edit_form.name) }}</th>
    <td>
      {{ form_errors(edit_form.name) }}
      {{ form_widget(edit_form.name) }}
    </td>
  </div>
</tr>
<tr>
  <div class="form-group">
    <th>{{ form_label(edit_form.price) }}</th>

```



```

        <td>
            {{ form_errors(edit_form.price) }}
            {{ form_widget(edit_form.price) }}
        </td>
    </div>
</tr>
<tr>
    <div class="form-group">
        <th>{{ form_label(edit_form.amount) }}</th>
        <td>
            {{ form_errors(edit_form.amount) }}
            {{ form_widget(edit_form.amount) }}
        </td>
    </div>
</tr>
</tbody>
</table>
    {{ form_row(edit_form.submit) }}
    <a href="{{ path('item') }}">
        Back to the list
    </a>
    {{ form_end(edit_form) }}
</div>
</div>
</div>
</div>
{% endblock %}

```

## The Form Action

We now have a form class and a template that renders it. Now, it's time to actually make it work with some actions. The item form is managed by four methods in the `ItemController`:

- `newAction`: Displays a blank form to create a new item
- `createAction`: Processes the form (validation, form repopulation) and creates a new item with the user submitted values
- `editAction`: Displays a form to edit an existing item
- `updateAction`: Processes the form (validation, form repopulation) and updates an existing item with the user submitted values

When you browse to the `/item/new` page, a form instance for a new item object is created by calling the `createForm()` method and passed to the template (`newAction`).

When the user submits the form (`createAction`), the form is bound (`bind($request)` method) with the user submitted values and the validation is triggered.

Once the form is bound, it is possible to check its validity using the `isValid()` method: if the form is valid (returns true), the item is saved to the database (`$em->persist($entity)`), and the user is redirected to the item show page; if not, the `new.html.twig` template is displayed again with the user submitted values and the associated error messages.

The modification of an existing item is quite similar. The only difference between the new and the edit action is that the item object to be modified is passed as the second argument of the `createForm` method. This object will be used for default widget values in the template.

You can also define default values for the creation form. For this we will pass a pre-modified `Item` object to the `createForm()` method to set the price default value to 0:

```
Src/Ju/GebeyaBundle/Controller/ItemController.php
```

```
public function newAction()
{
    $entity = new Item();
    $entity->setPrice(0);
    $form = $this->createForm($entity);
    return $this->render('JuGebeyaBundle:Item:new.html.twig', array(
```

```
'entity' => $entity,
'form'  => $form->createView(),
));
}
```

## Day 11 : The Fos User Bundle

The Symfony Security component provides a flexible security framework that allows you to load users from configuration, a database, or anywhere else you can imagine. The FOSUserBundle builds on top of this to make it quick and easy to store users in a database. So, if you need to persist and fetch the users in your system to and from a database, then you can use this Bundle.

### Prerequisites

This version of the bundle requires Symfony 2.1+. If you are using Symfony 2.0.x, please use the 1.2.x releases of the bundle.

### Translations

If you wish to use default texts provided in this bundle, you have to make sure you have translator enabled in your config.

```
app/config/config.yml
```

```
framework:
    translator: ~
```

For more information about translations, check [Symfony documentation](#).

#### Step 1: Download FOSUserBundle using composer

Require the bundle with composer:

```
php composer.phar require friendsofsymfony/user-bundle "~2.0@dev"
```

Composer will install the bundle to your project's vendor/friendsofsymfony/user-bundle directory.

#### Step 2: Enable the bundle

Enable the bundle in the kernel:

```
app/AppKernel.php
```

```

public function registerBundles()
{
    $bundles = array(
        //...
        new FOS\UserBundle\FOSUserBundle(),
    );
    //...
}

```

### Step 3: Create your User class

The goal of this bundle is to persist some **User** class to a database (MySQL, MongoDB, CouchDB, etc). Your first item, then, is to create the **User** class for your application. This class can look and act however you want: add any properties or methods you find useful. This is *your* **User** class.

The bundle provides base classes which are already mapped for most fields to make it easier to create your entity. Here is how you use it:

1. Extend the base **User** class (from the **Model** folder if you are using any of the **doctrine** variants, or **Propel** for propel 1.x)
2. Map the **id** field. It must be **protected** as it is inherited from the parent class.

In the following sections, you'll see how your User class should look, depending on how you're storing your users (Doctrine ORM, MongoDB ODM, or CouchDB ODM). Since we are using Doctrine ORM, we will see the Doctrine part.

### Doctrine ORM User class

If you're persisting your users via the Doctrine ORM, then your User class should live in the Entity namespace of your bundle and look like this to start:

```

src/Ju/GebeyaBundle/Resources/config/doctrine/User.orm.yml

Ju\GebeyaBundle\Entity\User:
    type: entity
    table: fos_user
    gedmo:
        loggable: true
    id:

```

```
id:
  type: integer
  generator:
    strategy: AUTO
```

Run the following Command and it will generate entities from the doctrine orm configurations.

```
php app/console doctrine:generate:entities JuGebeyaBundle
```

After the execution, the Doctrine will generate Entities in src/Ju/GebeyaBundle/Entity Directory with all setter and Getter methods.

extends BaseUser

Src/Ju/GebeyaBundle/Entity/User.php

```
<?php
namespace Ju\GebeyaBundle\Entity;
use FOS\UserBundle\Model\User as BaseUser;
use Doctrine\ORM\Mapping as ORM;
/**
 * User
 */
class User extends BaseUser
{
    /**
     * @var integer
     */
    protected $id;

    // Setter and Getter methods
}
}
```

#### Step 4: Configure your application's security.yml

In order for Symfony's security component to use the FOSUserBundle, you must tell it to do so in the security.yml file. The security.yml file is where the basic security configuration for your application is contained.

Below is a minimal example of the configuration necessary to use the FOSUserBundle in your application:

```
app/config/security.yml

security:
    encoders:
        FOS\UserBundle\Model\UserInterface: bcrypt
    role_hierarchy:
        ROLE_USER:       ROLE_USER
        ROLE_SUPER_ADMIN: ROLE_USER, ROLE_ADMIN
    providers:
        fos_userbundle:
            id: fos_user.user_provider.username_email
        in_memory:
            memory: ~
    firewalls:
        main:
            pattern: ^/
            form_login:
                provider: fos_userbundle
                csrf_provider: form.csrf_provider

            logout: true
            anonymous: true

    access_control:
        - { path: ^/login$, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/register, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/resetting, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/resetting/request, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/home, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/item/index, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/category/index, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/category/.*/show, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/, role: ROLE_USER }
```

Under the `providers` section, you are making the bundle's packaged user provider service available via the alias `fos_userbundle`. The id of the bundle's user provider service is `fos_user.user_provider.username`.

Next, take a look at and examine the `firewalls` section. Here we have declared a firewall named `main`. By specifying `form_login`, you have told the Symfony Framework that any time a request is made to this firewall that leads to the user needing to authenticate himself, the user will be redirected to a form where he will be able to enter his credentials. It should come as no surprise then that you have specified the user provider service we declared earlier as the provider for the firewall to use as part of the authentication process.

The `access_control` section is where you specify the credentials necessary for users trying to access specific parts of your application. The bundle requires that the login form and all the routes used to create a user and reset the password be available to unauthenticated users but use the same firewall as the pages you want to secure with the bundle. This is why you have specified that any request matching the `/login` pattern or starting with `/register` or `/resetting`, `/home` the home page, `item/index` list of available items, `/category/index` a page which displays all item categories, and `/category/*/show` which displays detail about the specific category have been made available to anonymous users. You have also specified that any other request will require a user to have the `ROLE_USER` role.

#### Step 5: Configure the FOSUserBundle

Now that you have properly configured your application's `security.yml` to work with the FOSUserBundle, the next step is to configure the bundle to work with the specific needs of your application.

Add the following configuration to your `config.yml` file according to which type of datastore you are using.

```
app/config/config.yml

fos_user:
  db_driver: orm # other valid values are 'mongodb', 'couchdb' and 'propel'
  firewall_name: main
  user_class: Ju\GebeyaBundle\Entity\User
  group:
    group_class: Ju\GebeyaBundle\Entity\Group
  use_listener: true
  use_username_form_type: true
  profile:
```

```
form:
    type:          fos_user_profile
    name:          fos_user_profile_form
    validation_groups: [Profile, Default]
```

Only three configuration values are required to use the bundle:

- The type of datastore you are using (orm, mongodb, couchdb or propel).
- The firewall name which you configured in Step 4.
- The fully qualified class name (FQCN) of the **User** class which you created in Step 3.

#### Step 6: Import FOSUserBundle routing files

Now that you have activated and configured the bundle, all that is left to do is import the FOSUserBundle routing files.

By importing the routing files you will have ready made pages for things such as logging in, creating users, etc.

```
app/config/routing.yml
```

```
fos_user:
    resource: "@FOSUserBundle/Resources/config/routing/all.xml"
```

In order to use the built-in email functionality (confirmation of the account, resetting of the password), you must activate and configure the SwiftmailerBundle.

#### Step 7: Update your database schema

Now that the bundle is configured, the last thing you need to do is to update your database schema because you have added a new entity, the User class which you created in Step 4.

For ORM run the following command.

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

You now can log in at [http://gebeya.local/app\\_dev.php/login](http://gebeya.local/app_dev.php/login)

## Day 12 : Security

### Securing the Application



Security is a two-step process whose goal is to prevent a user from accessing a resource that he/she should not have access to. In the first step of the process, the **authentication**, the security system identifies who the user is by requiring the user to submit some sort of identification. Once the system knows who you are, the next step, called the **authorization**, is to determine if you should have access to a given resource (it checks to see if you have privileges to perform a certain action).

The security component can be configured via your application configuration using the `security.yml` file from the `app/config` folder. As of Day 12, some content of the application (home page, list of item, list of categories, and page which display item detail and a page which shows category detail ) is accessible anonymously, and the rest will allow only users with `ROLE_USER`. To secure our application change your `security.yml` file:

```
app/config/security.yml

security:
  encoders:
    FOS\UserBundle\Model\UserInterface: bcrypt
  role_hierarchy:
    ROLE_USER:       ROLE_USER
    ROLE_SUPER_ADMIN: ROLE_USER, ROLE_ADMIN
  providers:
    fos_userbundle:
      id: fos_user.user_provider.username_email
    in_memory:
      memory: ~
  firewalls:
    main:
      pattern: ^/
      form_login:
        provider: fos_userbundle
        csrf_provider: form.csrf_provider
        login_path: /login
        default_target_path: /home
      logout:
```

```

    path: /logout
    target: /home
    anonymous: true
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    default:
        anonymous: ~
access_control:
- { path: ^/login$, role: IS_AUTHENTICATED_ANONYMOUSLY }
- { path: ^/register, role: IS_AUTHENTICATED_ANONYMOUSLY }
- { path: ^/resetting, role: IS_AUTHENTICATED_ANONYMOUSLY }
- { path: ^/resetting/request, role: IS_AUTHENTICATED_ANONYMOUSLY }
- { path: ^/home, role: IS_AUTHENTICATED_ANONYMOUSLY }
- { path: ^/item/index, role: IS_AUTHENTICATED_ANONYMOUSLY }
- { path: ^/item/.*/show, role: IS_AUTHENTICATED_ANONYMOUSLY }
- { path: ^/category/index, role: IS_AUTHENTICATED_ANONYMOUSLY }
- { path: ^/category/.*/show, role: IS_AUTHENTICATED_ANONYMOUSLY }
- { path: ^/, role: ROLE_USER }

```

Any request matching the `/login` pattern or starting with `/register`, `/resetting`, `/home` the home page, `item/index` list of available items, `item/.*/show` page to show detail information about specific item, `category/index` a page which displays all item categories, and `category/.*/show` which displays detail about the specific category have been made available to anonymous users. The rest of the pages will only be available if the user has `ROLE_USER`. But we can specify other permissions from controller request end with `/new`, `/edit`, and `/delete` will require a user to have the `ROLE_ADMIN` or `ROLE_SUPER_ADMIN` role.

For authenticating users, a traditional login form will be used and it works well but may not look great for our application.

[Log in](#)

Username  Password  ☐ Remember me

So, we can edit the form to have a better and beautiful look. First, create a layout for the user because it can be reusable for any authentication pages.

Src/Ju/GebeyaBundle/Resources/views/user.html.twig

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="description" content="Creative - Bootstrap 3 Responsive Admin
Template">
  <meta name="author" content="GeeksLabs">
  <meta name="keyword" content="Creative, Dashboard, Admin, Template, Theme,
Bootstrap, Responsive, Retina, Minimal">
  <link rel="shortcut icon" href="img/favicon.png">
  {% block stylesheets %}
    <link rel="stylesheet" href="{{ asset('bundles/jugebeya/css/bootstrap.min.css') }}"
type="text/css" media="all" />
    <link rel="stylesheet" href="{{ asset('bundles/jugebeya/css/font-
awesome.min.css') }}" type="text/css" media="all" />
    <link rel="stylesheet" href="{{ asset('bundles/jugebeya/css/prettyPhoto.css') }}"
type="text/css" media="all" />
    <link rel="stylesheet" href="{{ asset('bundles/jugebeya/css/price-range.css') }}"
type="text/css" media="all" />
    <link rel="stylesheet" href="{{ asset('bundles/jugebeya/css/animate.css') }}"
type="text/css" media="all" />
    <link rel="stylesheet" href="{{ asset('bundles/jugebeya/css/main.css') }}"
type="text/css" media="all" />
    <link rel="stylesheet" href="{{ asset('bundles/jugebeya/css/responsive.css') }}"
type="text/css" media="all" />
  {% endblock %}
  <!-- HTML5 shim and Respond.js IE8 support of HTML5 -->
  <!--[if lt IE 9]>
  <script src="js/html5shiv.js"></script>
  <script src="js/respond.min.js"></script>
  <![endif]-->
```

```

</head>
<body class="login-img3-body">
<div class="container">
    {% block body %}
    {% endblock body %}
</div>
</body>
</html>

```

And we will make the login form to extend the user layout from GebeyaBundle.

```

Vendor/friendsofsymfony/user-bundle/Resources/views/Security/login.html.twig

{% extends "JuGebeyaBundle::user.html.twig" %}
{% block body -%}
    <div class="container">
        <div class="row">
            <div class="col-sm-6 col-lg-push-3">
                <form action="{{ path('fos_user_security_check') }}" method="post" class="login-form" >
                    <input type="hidden" name="_csrf_token" value="{{ csrf_token }}" />
                    <div class="login-wrap">
                        <p class="login-img"><i class="icon_lock_alt"></i></p>
                        <div class="input-group">
                            <span class="input-group-addon"><i class="icon_profile"></i></span>
                            <input type="text" id="username" name="_username" required="required"
                                class="form-control" placeholder="Username" autofocus>
                        </div>
                        <div class="input-group">
                            <span class="input-group-addon"><i class="icon_key_alt"></i></span>
                            <input type="password" id="password" name="_password" class="form-control" placeholder="Password">
                        </div>
                        {% if error %}

```

```

        <div>
            {{ error.messageKey|trans(error.messageData, 'security') }}
        </div>
        {% endif %}
        <label class="checkbox">
            <input type="checkbox" id="remember_me" name="_remember_me"
value="on" > Remember me
            <span class="pull-right"> <a href="{{path('fos_user_resetting_request')}}">
Forgot Password?</a></span>
        </label>
        <button id="_submit" name="_submit" class="btn btn-primary btn-lg btn-block"
type="submit">Login</button>
        <a class="btn btn-info btn-lg btn-block"
href="{{path('fos_user_registration_register')}}">Signup</a>
    </div>
</form>
</div>
</div>
</div>
{% endblock body %}

```

The better look

Username

Password

☐ Remember me [Forgot Password?](#)

Login

Signup

User Providers

During authentication, the user submits a set of credentials (usually a username and password). The Item of the authentication system is to match those credentials against some pool of users. So where does this list of users come from?

In Symfony2, users can come from anywhere – a configuration file, a database table, a web service, or anything else you can dream up. Anything that provides one or more users to the authentication system is known as a “user provider”.

Symfony2 comes standard with the two most common user providers: one that loads users from a configuration file and one that loads users from a database table.

In our case we used the second case: the user provider uses fos\_user\_bundle that access the User entity class to retrieve from the database. It retrieve the entity either in username or email

```
app/config/security.yml

providers:
    fos_userbundle:
        id: fos_user.user_provider.username_email
    in_memory:
        memory: ~
```

## Logout

Logging out is handled automatically by the firewall. All you have to do is to activate the logout config parameter:

```
app/config/security.yml

security:
    firewalls:
        main:
            pattern: ^/
            form_login:
                provider: fos_userbundle
                csrf_provider: form.csrf_provider
```

```
login_path: /login
default_target_path: /home
logout:
  path: /logout
  target: /home
anonymous: true
```

You will not need to implement a controller for the /logout URL as the firewall takes care of everything. You can also change the target where the logout will redirect you. In our Project, since anonymous user can access the homepage; we will redirect the user to home page.

Once this is configured, sending a user to /logout (or whatever you configure the path to be), will un-authenticate the current user. The user will then be sent to the homepage (the value defined by the target parameter).

And we already have the logout link for loggedin users in our project layout.

Src/Ju/GebeyaBundle/Resources/views/layout.html.twig

```
{% if is_granted("ROLE_SUPER_ADMIN") or is_granted("ROLE_ADMIN") or
is_granted("ROLE_USER") %}
  <li>
    <a href="{{path('fos_user_security_logout')}}">
      <i class="fa fa-sign-out"></i> Logout
    </a>
  </li>
{% else %}
```

Now, if you try to access any secure section (clear the cache first), you will be asked for an username and password and then, the logout link will be shown in the top-right corner and as menu item on the navigation section.

## The User Session

Symfony2 provides a nice session object that you can use to store information about the user between requests. By default, Symfony2 stores the attributes in a cookie by using the native PHP sessions.

You can store and retrieve information from the session easily from the controller: As the user Logged in and Add the first item to Cart, new Cart will be created and

stored in the session. Then, the user will add any other items only to that cart till he logout. In the next login, it will be another new cart.

```
Src/JuGebeyaBundle/Controller/DefaultController.php

public function addToCartAction(Request $request, $item_id)
{
    $em = $this->getDoctrine()->getManager();
    $item = $em->getRepository('JuGebeyaBundle:Item')->find($item_id);
    if(!$this->get('session')->get('cart_id')) {
        $cart = new Cart();
        $cart->setCreatedAt(new \DateTime());
    }
    else
        $cart = $em->getRepository('JuGebeyaBundle:Cart')->find($this->get('session')->get('cart_id'));
    if(!$cart->getItems()->contains($item))
    {
        $cart->setUpdatedAt(new \DateTime());
        $cart->addItem($item);
        $cart->setPrice($cart->getPrice()+$item->getPrice());
        $em->persist($cart);
        $em->flush();
        $this->get('session')->set('cart_id',$cart->getId());
    }
    $referer = $request->headers->get('referer');
    return new RedirectResponse($referer);
}
```

## Flash Messages

Flash messages are small messages you can store on the user's session for exactly one additional request. This is useful when processing a form: you want to redirect and have a special message shown on the next request.



When the user add item to cart s/he want to know if the operation was successful or not. So, we use flash messages to notify the user about the result.

```
Src/Ju/GebeyaBundle/Controller/DefaultController.php

public function addToCartAction(Request $request, $item_id)
{
    if(!$cart->getItems()->contains($item))
    {
        //...
        $this->get('session')->getFlashBag()->add('success', 'Success: Item Added to Cart');
    }
    else
        $this->get('session')->getFlashBag()->add('error', 'Error: Item was Already in the Cart');
    //...
}
```

At last, you will add a div in which the flash message will be displayed:

```
src/lbw/ItemmeetBundle/Resources/views/layout.html.twig

<div class="row">
    <div class="col-sm-6 col-lg-push-3">
        {% for flashMessage in app.session.flashbag.get('error') %}
            <div>
                {{ flashMessage }}
            </div>
        {% endfor %}
        {% for flashMessage in app.session.flashbag.get('success') %}
            <div>
                {{ flashMessage }}
            </div>
        {% endfor %}
    </div>
```

</div>

After the user Logged in and added items to cart, he can see all his carts latest first. He can then choose any cart and click “order the cart” link and then the delivery will be takes place also the item can be considered as sold. So, you can get it in Best Sell item.

Src/JuGebeyaBundle/Controller/DefaultController.php

```
public function orderCartAction(Request $request, $cart_id)
{
    $em = $this->getDoctrine()->getManager();
    $cart = $em->getRepository('JuGebeyaBundle:Cart')->find($cart_id);
    if(!$this->get('session')->get('order_id')) {
        $itemOrder = new ItemOrder();
        $itemOrder->setDateOfPurchase(new \DateTime());
    }
    else
        $itemOrder = $em->getRepository('JuGebeyaBundle:ItemOrder')->find($this->get('session')->get('order_id'));
    if(!$itemOrder->getCart()->contains($cart))
    {
        $cart->setItemOrders($itemOrder);
        $itemOrder->addCart($cart);
        $itemOrder->setCreatedBy($this->get('security.context')->getToken()->getUser());
        $em->persist($itemOrder);
        $em->persist($cart);
        $em->flush();
        $this->get('session')->set('order_id',$itemOrder->getId());
        $this->get('session')->getFlashBag()->add('success', 'Success: Cart Ordered');
    }
    else
        $this->get('session')->getFlashBag()->add('error', 'Error: Cart was Already Ordered');
```

```
$referer = $request->headers->get('referer');  
return new RedirectResponse($referer);  
}
```

Goto [http://gebeya.local/app\\_dev.php/cart/{any\\_cart\\_id}/show](http://gebeya.local/app_dev.php/cart/{any_cart_id}/show)

Order Now

Back to the list

### SELECTED CART DETAILS

|                  |      |
|------------------|------|
| Total Cart Price | 12.5 |
|------------------|------|

|                            |                     |
|----------------------------|---------------------|
| The is Category Created At | 2016-03-16 05:10:07 |
|----------------------------|---------------------|

| Contained Items | <table><tr><th>Company</th><th>Item Name</th><th>Unit Price</th><th>Available Amount</th></tr><tr><td>Mars</td><td>Mars chocolate</td><td>12.5 Birr</td><td>200</td></tr></table> | Company    | Item Name        | Unit Price | Available Amount | Mars | Mars chocolate | 12.5 Birr | 200 |
|-----------------|---|------------|------------------|------------|------------------|------|----------------|-----------|-----|
| Company         | Item Name   | Unit Price | Available Amount |            |                  |      |                |           |     |
| Mars            | Mars chocolate  | 12.5 Birr  | 200              |            |                  |      |                |           |     |