# Symfony 3 Learning Project

## Overview

This project is to implement a fully-functional ***Task Management*** application using Symfony 3 Framework.

To keep simple, the project will be structured in different releases or versions. Each one has new a more complex requirements than prior release.

This project is intended not only to know the framework capabilities, but also to apply some of the most common Software Development best practices as Automatic Testing and Version Control.

## Requirements

### Technical Requirements

* PHP 5.6.x or later
* MySQL 5.7.x Server and Client Tools
* SQLite 3
* Composer
* Any IDE that supports Symfony 3 (PHPStorm prefered)
  + Install Symfony Plugin and PHP annotations Plugin
* Git client
* A compatible Web Browser with HTML5 and CSS3

### Previous Knowledge Requirements

* Basic usage of Git
* Good understanding of OOP
* Execute commands in a terminal
* Basic understanding of HTML and CSS
* PHP knowledge is desirable but not mandatory

## Project Setup

1. Ensure you can execute php from a console with the following command

$ php –v

PHP 5.6.19 (cli) (built: Mar 2 2016 20:09:42)

Copyright (c) 1997-2016 The PHP Group

Zend Engine v2.6.0, Copyright (c) 1998-2016 Zend Technologies

**Note**: If you have an error configure your PATH variable accordingly to your Operative System.

1. Edit your php.ini file located in the the root of your PHP installation and define the following properties as shown here:

...

; Directory in which the loadable extensions (modules) reside.

; http://php.net/extension-dir

; extension\_dir = "./"

; On windows:

**extension\_dir = "ext"**

...

**extension=php\_openssl.dll**

**extension=php\_pdo\_mysql.dll**

**extension=php\_pdo\_sqlite.dll**

...

**extension=php\_sqlite3.dll**

...

[Date]

; Defines the default timezone used by the date functions

; http://php.net/date.timezone

**date.timezone = America/Mexico\_City**

1. Check if Composer is installed

$ composer -V

Composer version 1.0-dev (4cc719cab3906ce444e4eb38d59a789bd90b31e2) 2016-03-07

3:07:02

1. Check SQLite installation

$ sqlite3 -version

3.11.1 2016-03-03 16:17:53 f047920ce16971e573bc6ec9a48b118c9de2b3a7

1. Create the following database schemas and users

|  |  |
| --- | --- |
| Object name | Description |
| taskmanager\_prod | Database schema for production releases |
| taskmanager\_dev | Database schema for development |
| taskmanager\_prod\_user | Production database user (password : prod) |
| taskmanager\_dev\_user | Development database user (password : dev) |

1. Open your terminal and go to your projects folder
2. Execute the following command to download the Symfony installer

$ php -r "file\_put\_contents('symfony', file\_get\_contents('https://symfony.com/installer'));"

1. Create the project structure

$ php symfony new task-manager

Downloading Symfony...

0.00 B/4.97 MB >----------------------------------------------------------- 0%

...

4.97 MB/4.97 MB ============================================================ 100%

Preparing project...

OK Symfony 3.0.3 was successfully installed. Now you can:

\* Change your current directory to C:\Users\cisneben\projects\task-manager

\* Configure your application in app/config/parameters.yml file.

\* Run your application:

1. Execute the php bin/console server:run command.

2. Browse to the http://localhost:8000 URL.

\* Read the documentation at http://symfony.com/doc

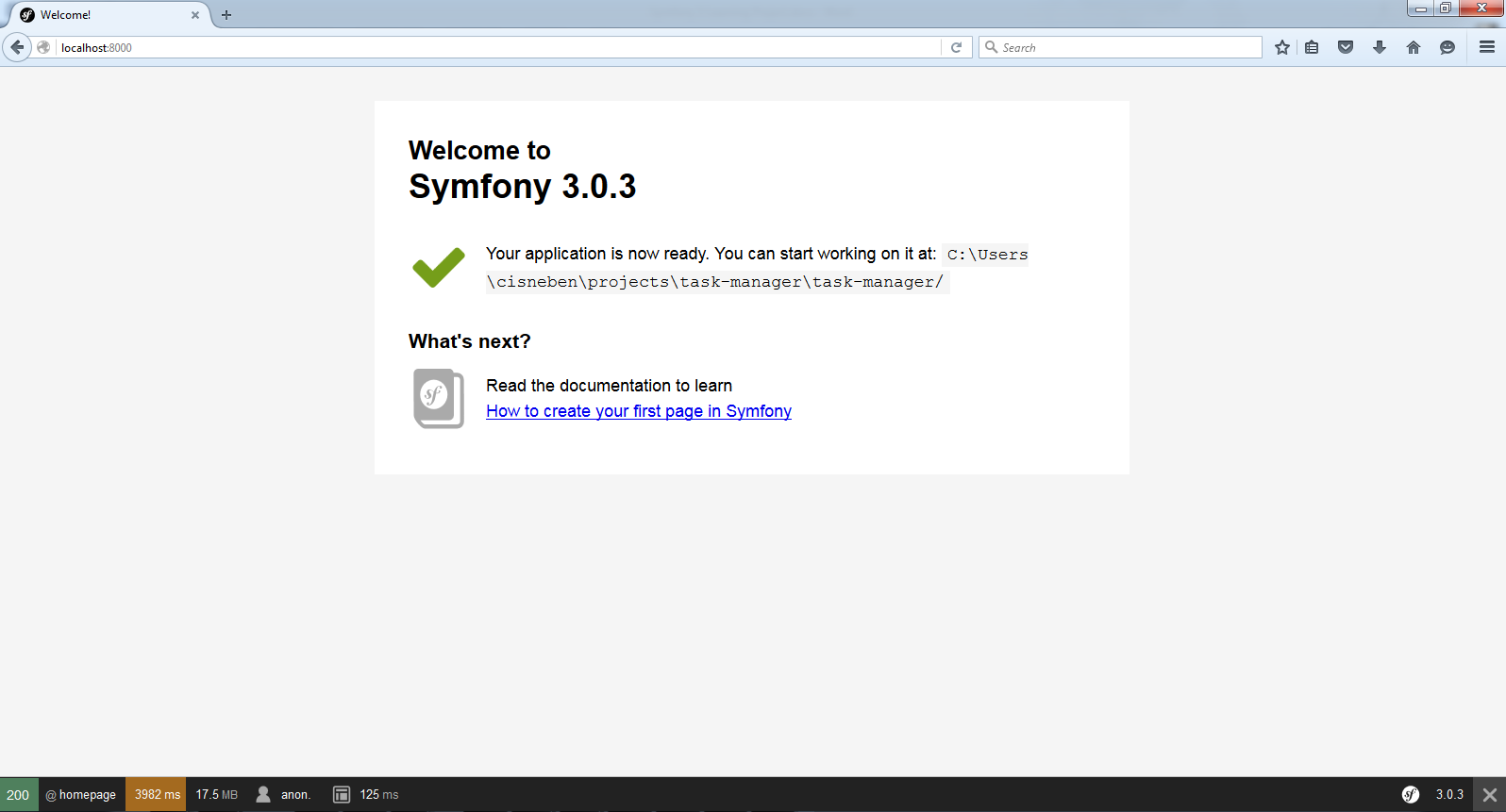
1. Move to task-manager folder

$ cd task-manager

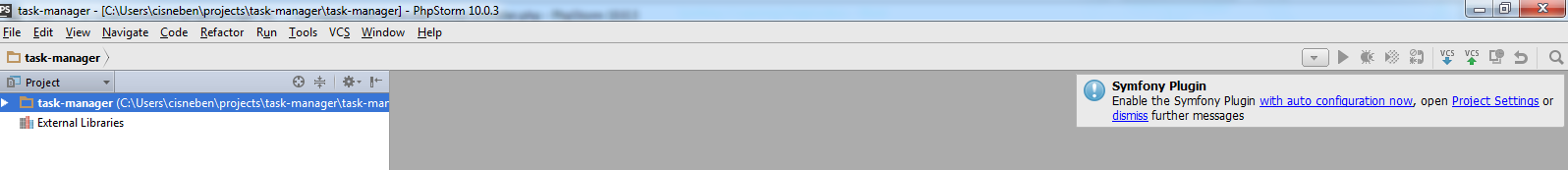
1. Run the server

$ php bin/console server:run

1. Open your browser at <http://localhost:8000>



1. Open your project in PHPStorm and enable Symfony Plugin with auto configuration now



1. Edit composer.json file and add phpunit dependency

"require-dev": {

"sensio/generator-bundle": "^3.0",

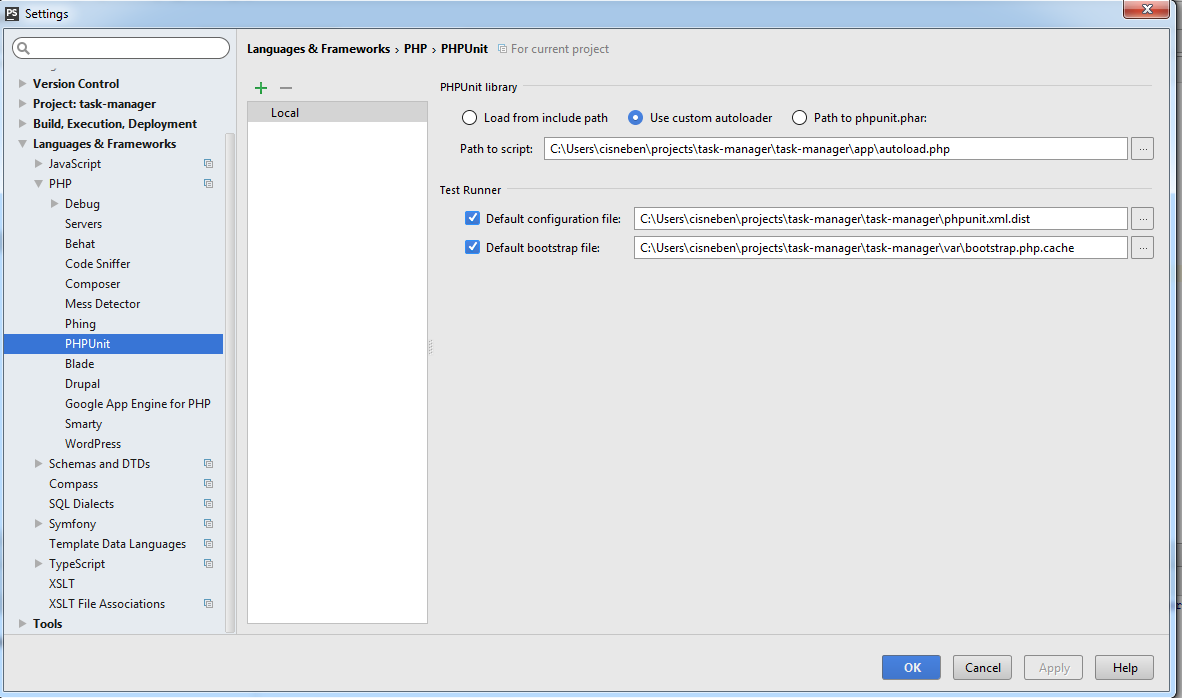
"symfony/phpunit-bridge": "^3.0",

**"phpunit/phpunit": "4.8.\*"**

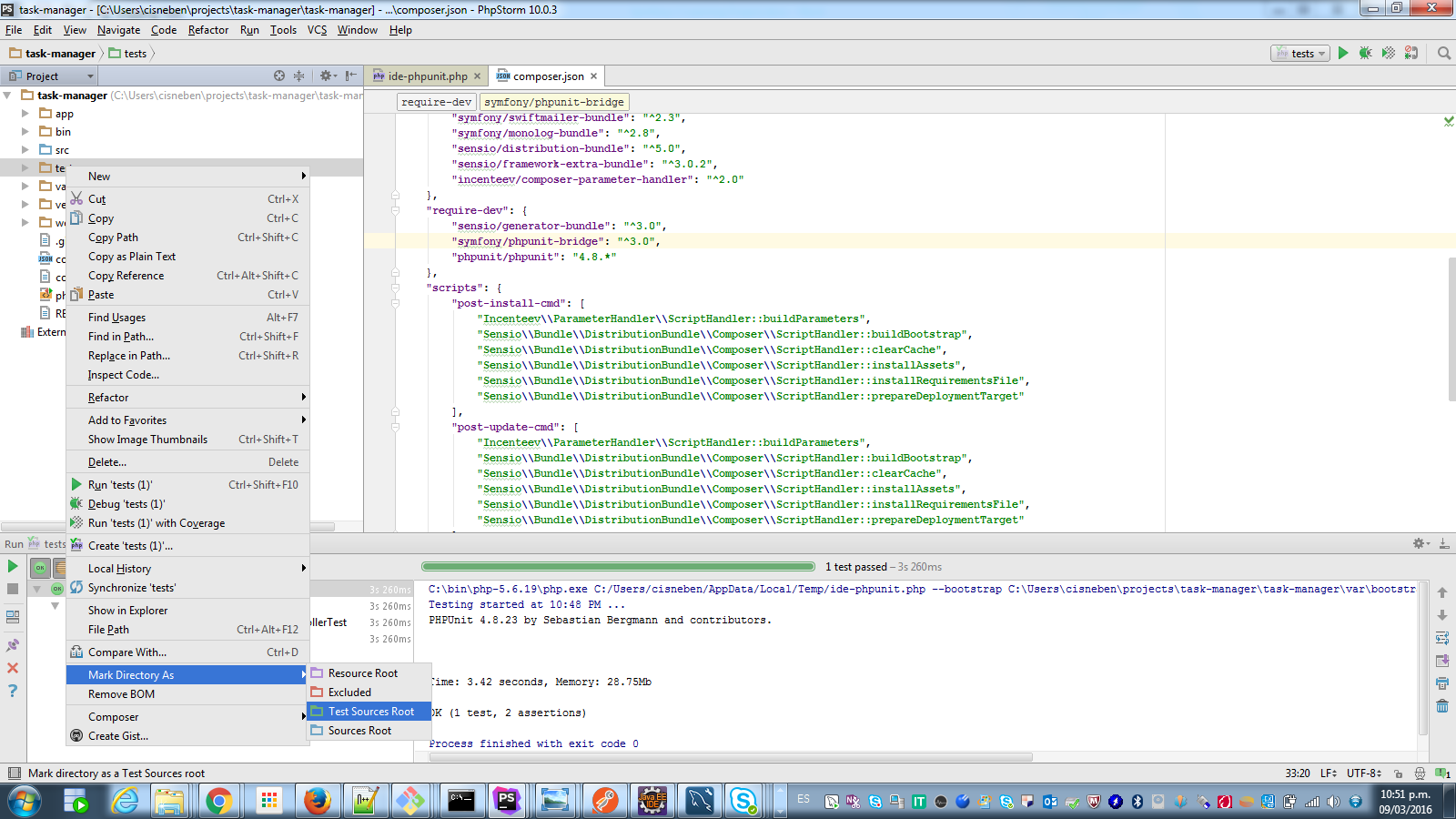
1. Execute this command

$ composer update -vvv

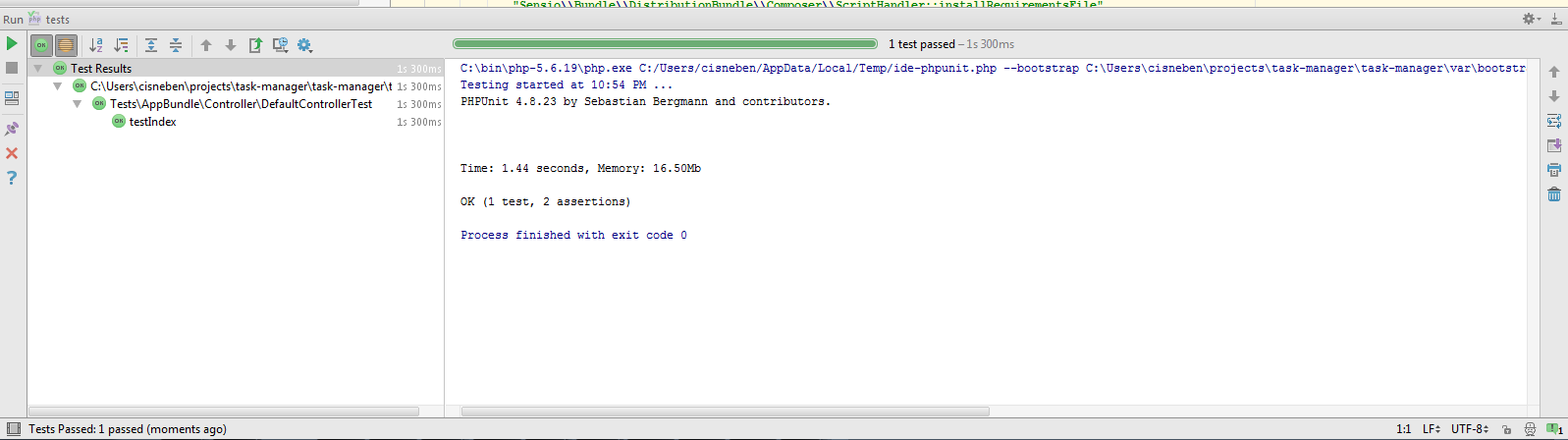
1. Configure PHP Unit



1. Mark tests directory as Test Source Folder



1. Run your tests



1. Initialize your repository

$ git init

Initialized empty Git repository in C:/Users/cisneben/projects/task-manager/task-manager/.git/

$ git checkout -b development

1. Edit .gitignore and add a new line with this entry

.idea/workspace.xml

1. Commit your files

$ git add.

$ git commit -m “Project structure”

## Functional Requirements

### Version 1.0

#### Release overview

Create an aplication that manage a list of tasks for personal reasons.

A task should have the following data:

|  |  |  |
| --- | --- | --- |
| Field | Data Type | Comments and restrictions |
| Id | Integer number | Task identifier. Primary key auto-generated |
| Name | String | Short name of the task. Required for all tasks and must have a length between 5 and 60 characters. |
| Description | String | Long description of the task. Optional and could store up 255 characters. |
| Due date | Datetime | Stores the end date of the task. Required. Can store date and time components including seconds. |
| Creation date | Datetime | Stores the creation date of the task. Required and must be generated for the application automatically. |
| Category | String | Stores the category of the task. Examples could be: Work, Personal, Social, etc. No required. |
| Priority | String | Stores the priority of the task. Examples could be: Normal, Low, High and Urgent. |
| Status | String | Stores the current status of the task. Examples could be: New, In progress or Closed. |

The application should have or perform the following actions or requirements:

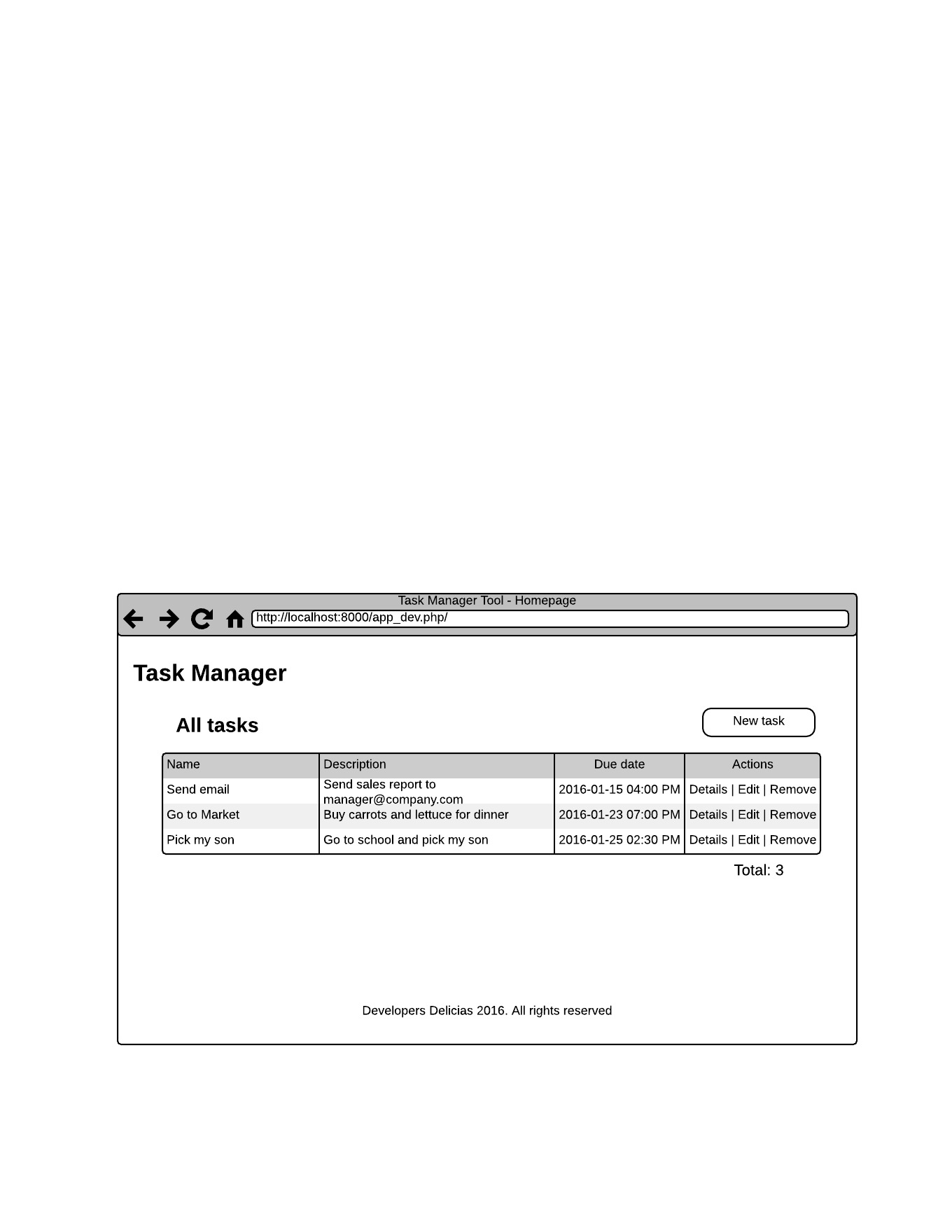
|  |  |  |
| --- | --- | --- |
| Requirement Id | Name | Description |
| FR-01-001 | Show the list of tasks | Homepage should list all tasks created ordered by due date and priority. |
| FR-01-002 | Create a new task | Should can create new tasks |
| FR-01-003 | Edit a task | Should can edit task information |
| FR-01-004 | Show task details | Should can review all task information |
| FR-01-005 | Delete a task | Should can delete a task |

#### Detailed requirements specifications

Note: Following templates are just to have an idea about the implementation and does not represents the real interface. Is up to you apply different styles and colors, but keeping the original idea.

##### FR-01-001 Show the list of tasks

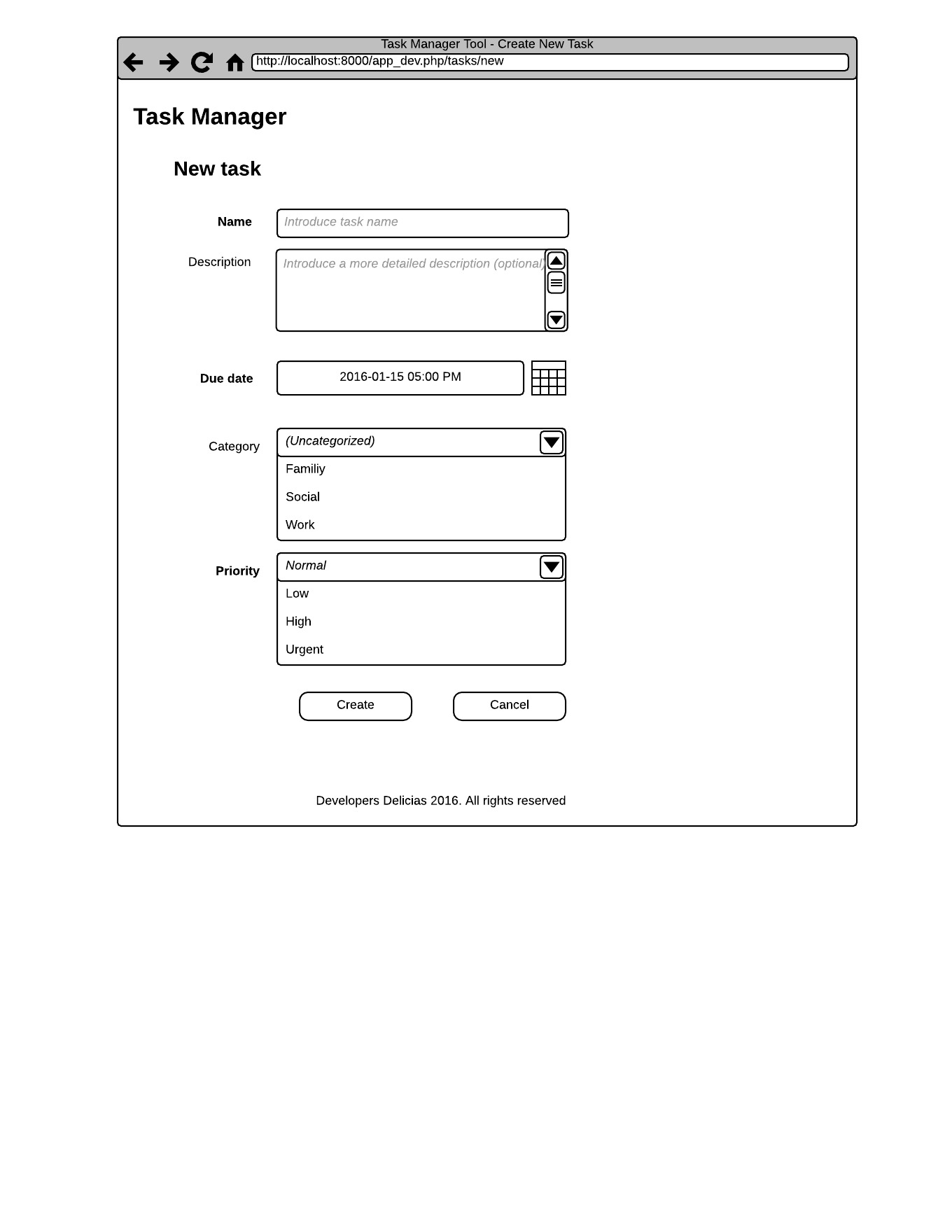
This is a template of how to show the list of tasks in homepage



###### Validations and requirements

* Should be ordered by due date ascendent
* Actions could be links or buttons
* Should show the total number of tasks
* Due date should apply date format as shown

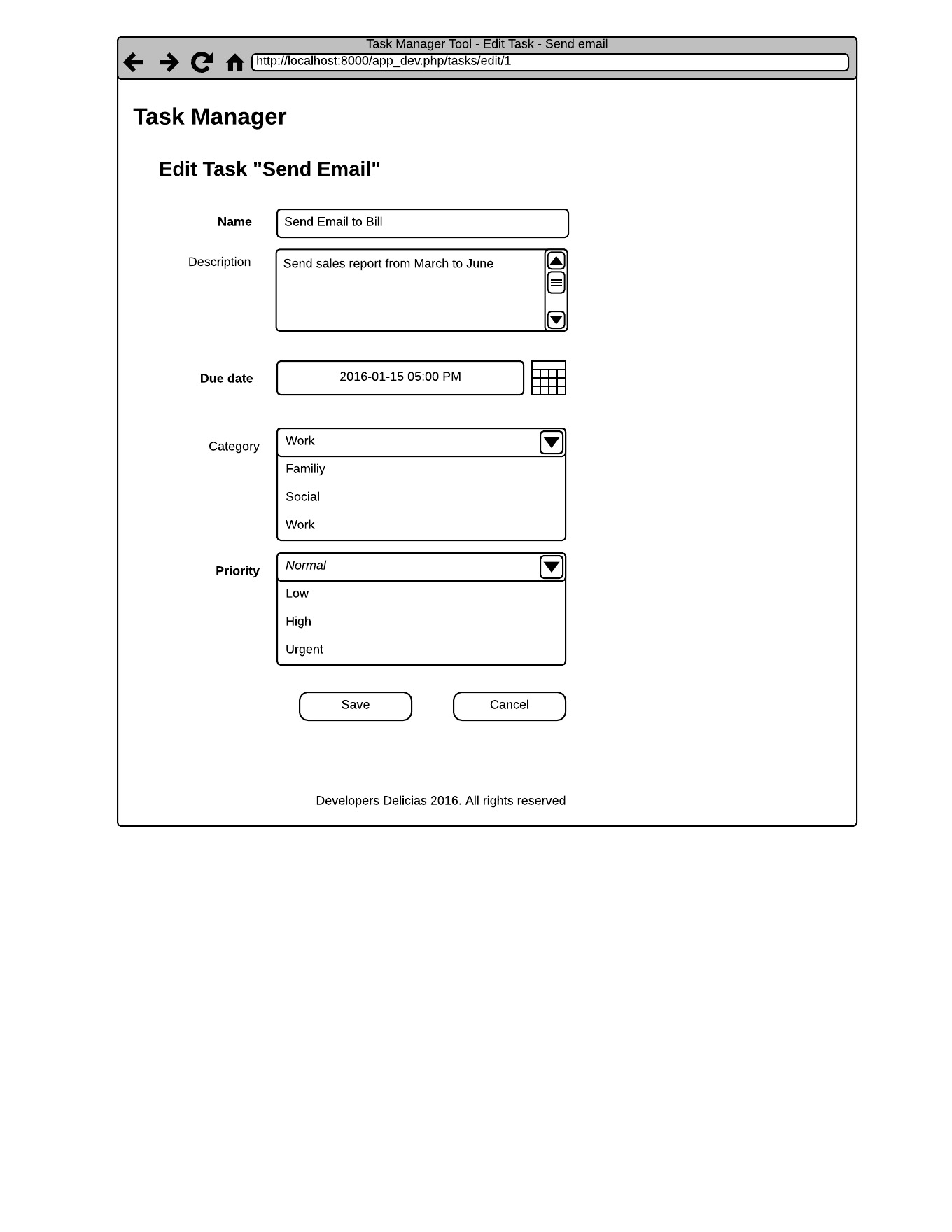
##### FR-01-002 Create new task

When creating a new task just show the following fields:Validations and Requirements

* Name, due date and priority are mandatory fields so, show a different style that express this condition.
* By default Category show Uncategorized option that means no category or null.
* Due date text field cannot be edited, just picking from date time picker
* Show Normal priority as default.
* When a validation fails, show a short message below each field indicationg what is wrong.
* When click Create button execute creation action and redirect to the task list page and show the new created task.
* When click Cancel button do not create nothing and return to the task list page.

##### FR-01-003 Edit a task

When editing a task just show the following fields:

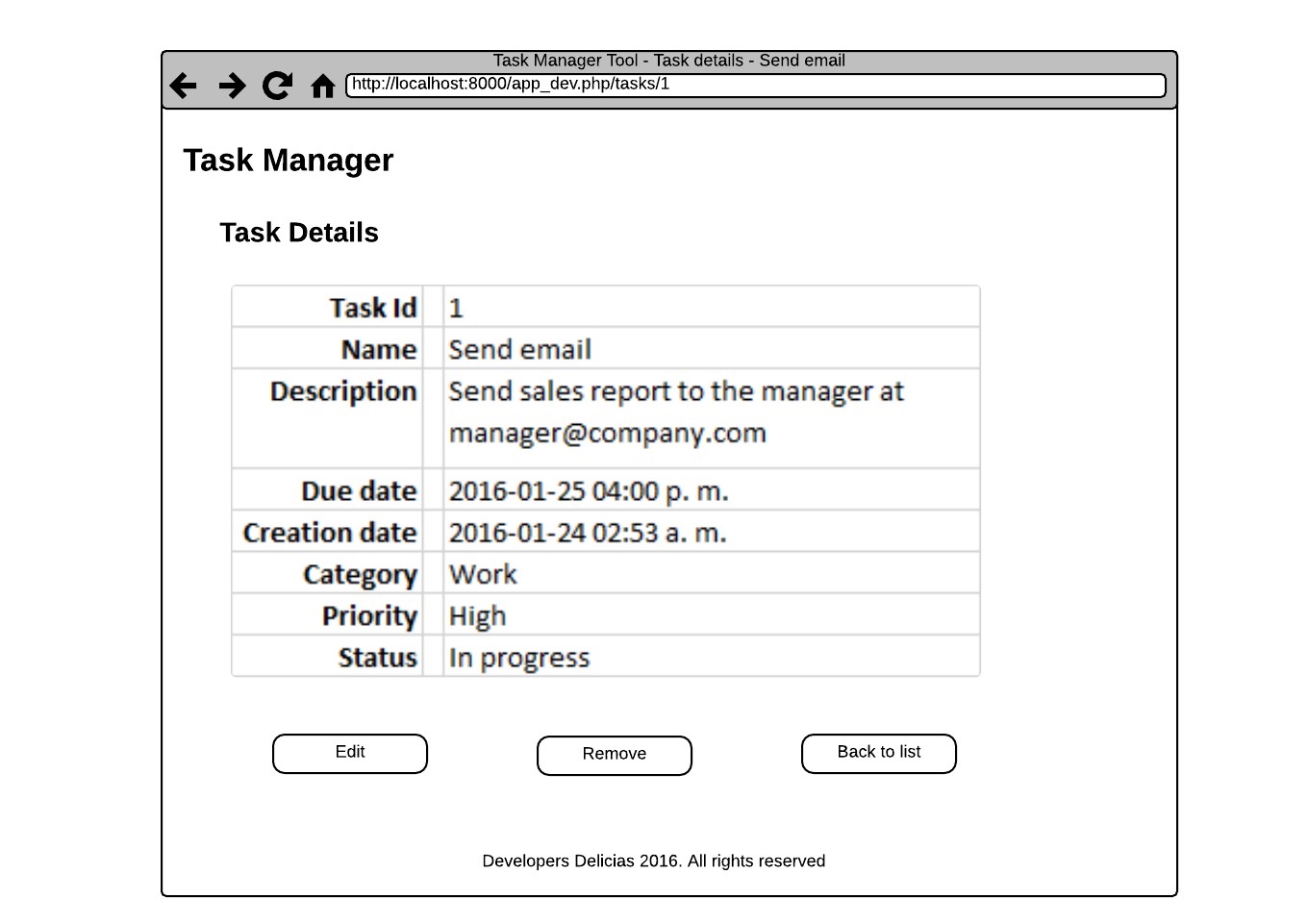


Validations and Requirements

* Apply the same validations as Create New Task

##### FR-01-004 Display task details

Display a table with tasks details. Also add some buttons to apply the other actions.

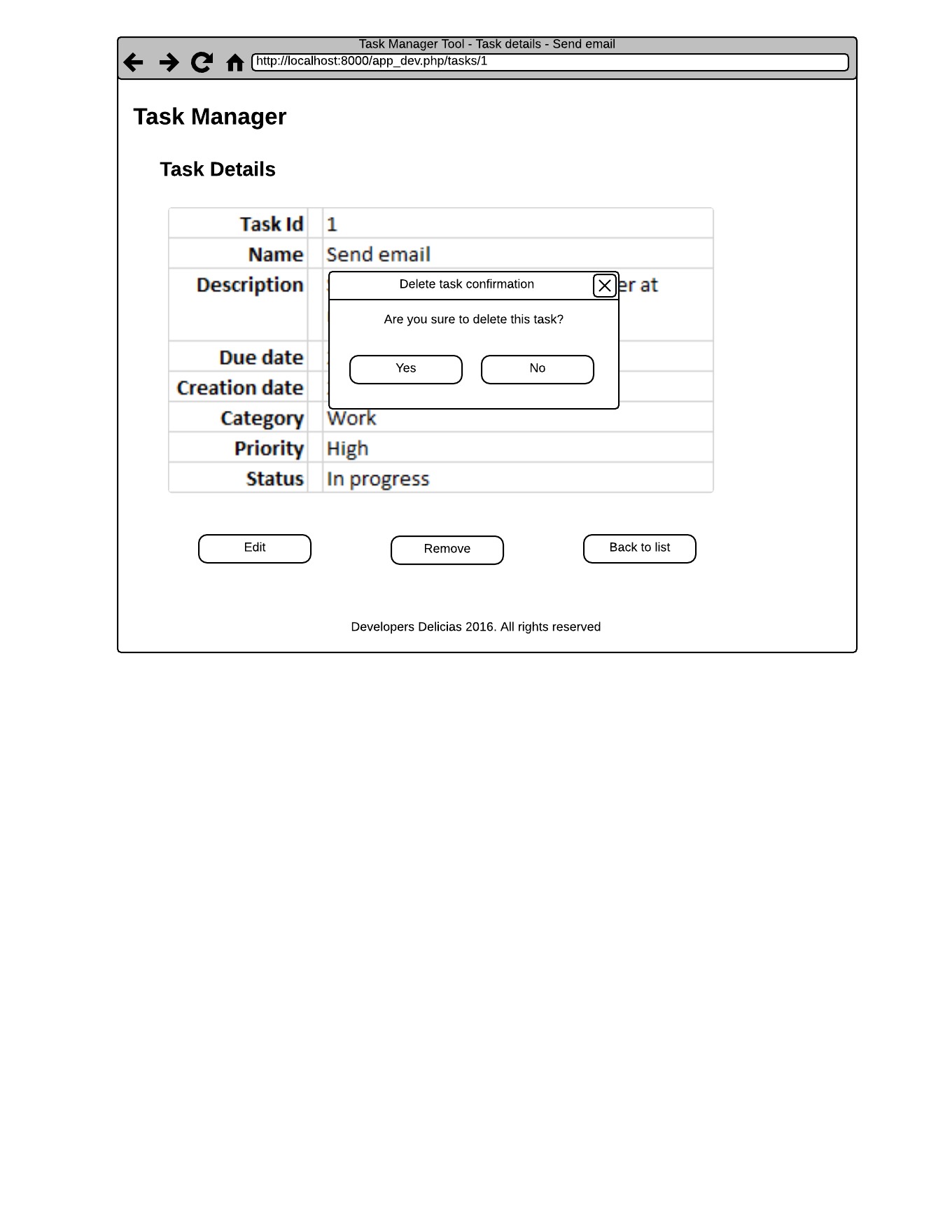


Validations and Requirements

* Every button should redirect to the especified action

##### FR-01-005 Delete a task

When delete a task show a confirm dialog to ensure task deletion.



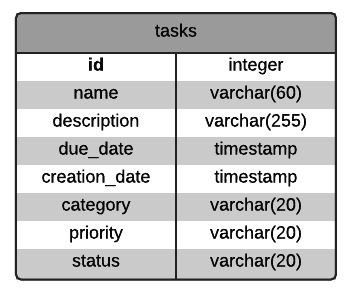
Validations and Requirements

* If user confirm deletion the task is deleted from the database
* After deletion you should redirect to the task list view
* If a user clicks on “No” button, no action is performed.

#### Design

##### Relational Model

For now our model only consists in one table as shown below:



Check release overview for more details about constraints of each field.

##### Components Design

We are going to analize every component to determine how many components and their type. We use MVC design pattern as base.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Req. Id | Name | Model | View | Controller/Action | Route | Method |
| FR-01-001 | Show the list of tasks | Task | index.html.twig | TaskController/list | /tasks | GET |
| FR-01-002 | Create a new task | Task | new.html.twig | TaskController/new  TaskController/create | /tasks/new  /tasks/ | GET  POST |
| FR-01-003 | Edit a task | Task | edit.html.twig | TaskController/edit  TaskController/save | /tasks/edit/{id}  /tasks/ | GET  PUT |
| FR-01-004 | Show task details | Task | show.html.twig | TaskController/show | /tasks/{id} | GET |
| FR-01-005 | Delete a task | Task | n/a | TaskController/delete | /tasks/{id} | DELETE |

#### Construction phase

##### Create a new bundle

Execute the following command

$ php bin/console generate:bundle

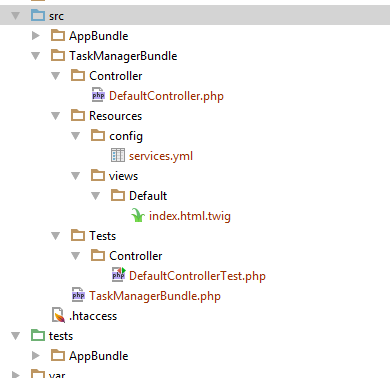
Are you planning on sharing this bundle across multiple applications? (no)

Bundle name: TaskManagerBundle

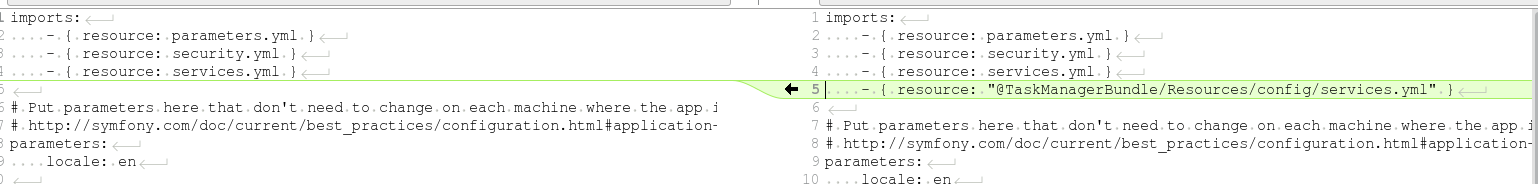
Target Directory: /src

Configuration format (annotation, yml, xml, php): annotation

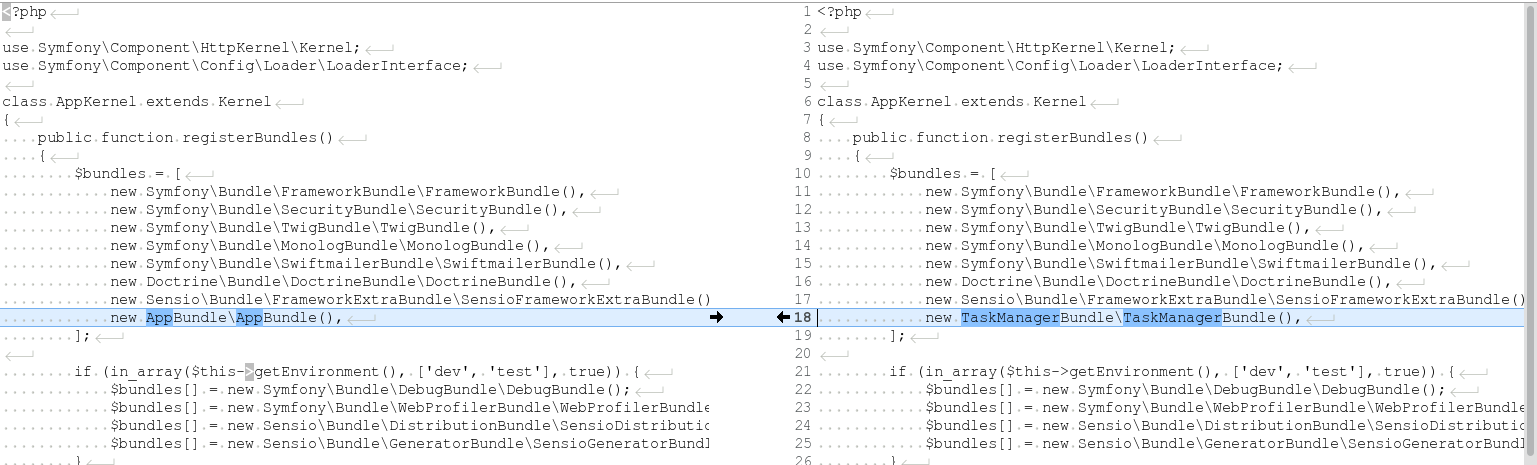
Check your files:



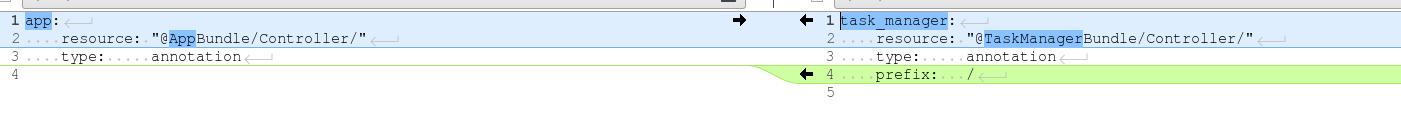
app/config/config.yml



1. Create a new folder TaskManagerBundle inside tests folder
2. Move Controller folder inside src/TaskManagerBundle/Tests folder
3. Delete src/TaskManagerBundle/Tests folder
4. Delete src/AppBundle folder
5. Delete tests/AppBundle folder
6. Remove AppBundle from AppKernel.php class



1. Remove from routing.yml app entry



1. Test your application

##### Create model layer

1. Execute in the console:

$ php bin/console generate:doctrine:entity

1. Follow the wizard to create the model entity accordingly to the definition.
2. Edit the generated class inside src/TaskManagerBundle/Entity/Task.php to map “tasks” table instead of “task”

<?php

namespace TaskManagerBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/\*\*

\* Task

\*

\* **@ORM\Table(name="tasks")**

\* @ORM\Entity(repositoryClass="TaskManagerBundle\Repository\TaskRepository")

\*/

class Task

{

1. Add this configuration at the end in app/config/config\_dev.yml file

***# Doctrine Configuration*doctrine:  
 dbal:  
 driver: pdo\_mysql  
 host: "%database\_host%"  
 port: "%database\_port%"  
 dbname: taskmanager\_dev  
 user: taskmanager\_dev\_user  
 password: dev  
 charset: UTF8**

1. Execute this command

$ php bin/console doctrine:schema:create --env=dev

ATTENTION: This operation should not be executed in a production environment.

Creating database schema...

Database schema created successfully!

1. Create data fixtures for tasks

$ composer require "doctrine/doctrine-fixtures-bundle" -vvv

1. Add this entry for dev and test environments

use Symfony\Component\HttpKernel\Kernel;

class AppKernel extends Kernel

{

public function registerBundles()

{

$bundles = array(

// ...

);

if (in\_array($this->getEnvironment(), array('dev', 'test'))) {

// ...

**$bundles[] = new Doctrine\Bundle\FixturesBundle\DoctrineFixturesBundle();**

}

return $bundles;

}

// ...

}

1. Create the following class in src/TaskManagerBundle/DataFixtures/ORM/LoadInitialTaskData.php

<?php

namespace TaskManagerBundle\DataFixtures\ORM;

use Doctrine\Common\DataFixtures\FixtureInterface;

use Doctrine\Common\Persistence\ObjectManager;

use Symfony\Component\Validator\Constraints\DateTime;

use TaskManagerBundle\Entity\Task;

class LoadInitialTaskData implements FixtureInterface

{

/\*\*

\* Load data fixtures with the passed EntityManager

\*

\* @param ObjectManager $manager

\*/

public function load(ObjectManager $manager)

{

$sendEmailTask = new Task();

$sendEmailTask->setName("Send email");

$sendEmailTask->setDescription("Send sales report to manager@company.com");

$sendEmailTask->setDueDate(new \DateTime('15-01-2016 16:00'));

$sendEmailTask->setCreationDate(new \DateTime('14-01-2016 02:45'));

$sendEmailTask->setCategory("Work");

$sendEmailTask->setPriority("Normal");

$sendEmailTask->setStatus("New");

$goToMarketTask = new Task();

$goToMarketTask->setName("Go to Market");

$goToMarketTask->setDescription("Buy carrots and lettuce for dinner");

$goToMarketTask->setDueDate(new \DateTime('23-01-2016 19:00'));

$goToMarketTask->setCreationDate(new \DateTime('20-01-2016 04:16'));

$goToMarketTask->setCategory("Home");

$goToMarketTask->setPriority("Low");

$goToMarketTask->setStatus("New");

$pickMySonTask = new Task();

$pickMySonTask->setName("Pick my son");

$pickMySonTask->setDescription("Go to school and pick my son");

$pickMySonTask->setDueDate(new \DateTime('25-01-2016 14:30'));

$pickMySonTask->setCreationDate(new \DateTime('25-01-2016 02:16'));

$pickMySonTask->setCategory("Personal");

$pickMySonTask->setPriority("Urgent");

$pickMySonTask->setStatus("Open");

$manager->persist($sendEmailTask);

$manager->persist($goToMarketTask);

$manager->persist($pickMySonTask);

$manager->flush();

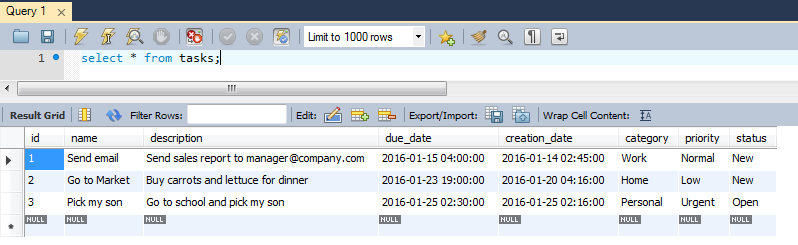
}

}

1. Execute the following command to load the fixtures

$ php bin/console doctrine:fixtures:load

1. Check your database if you have created this values



1. Generate the CRUD

$ php bin/console generate:doctrine:crud --entity=TaskManagerBundle:Task --format=annotation --with-write --no-interaction

CRUD generation

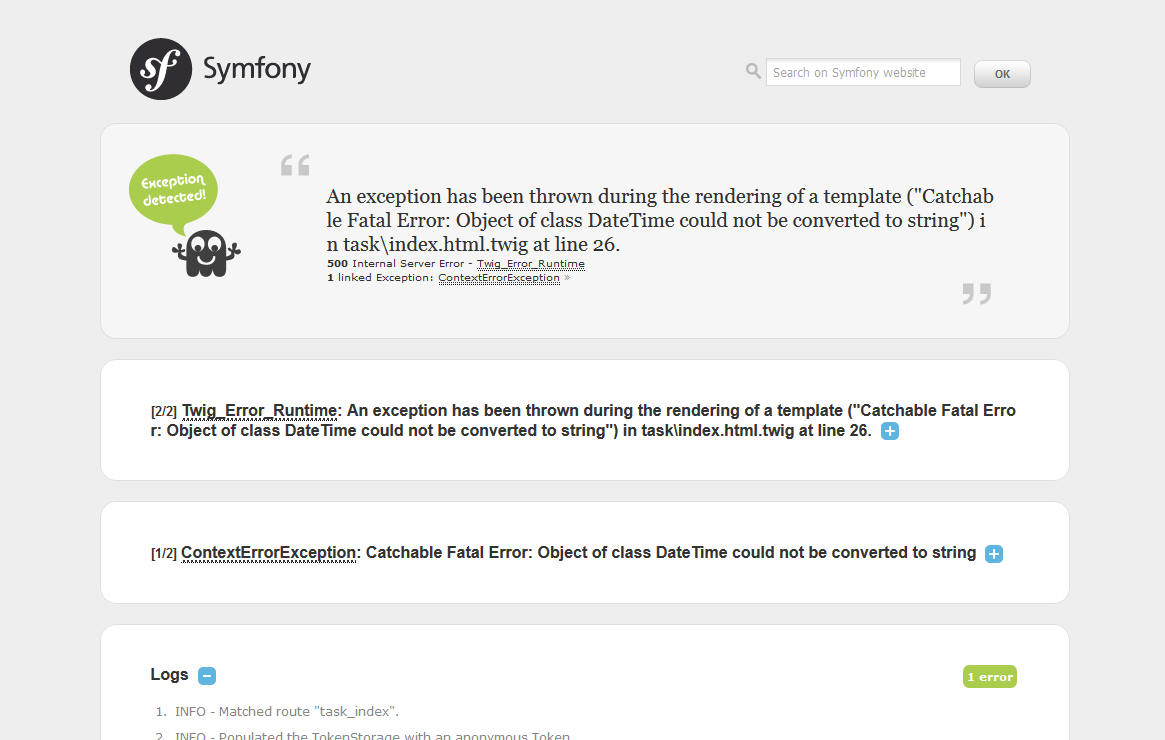
Generating the CRUD code: OK

Generating the Form code: OK

Updating the routing: OK

Everything is OK! Now get to work :).

1. Run the server and go to <http://localhost:8000/app_dev.php/task/> and you get this error:



1. Modify app\Resources\views\task\index.html.twig file to add the following:

<tr>

<td><a href="{{ path('task\_show', { 'id': task.id }) }}">{{ task.id }}</a></td>

<td>{{ task.name }}</td>

<td>{{ task.description }}</td>

**<td>{{ task.dueDate | date('d-m-Y h:i A') }}</td>**

**<td>{{ task.creationDate | date('d-m-Y h:i A') }}</td>**

1. Also in app\Resources\views\task\show.html.twig template

<tr>

<th>Description</th>

<td>{{ task.description }}</td>

</tr>

<tr>

<th>Duedate</th>

**<td>{{ task.dueDate | date('d-m-Y h:I A) }}</td>**

</tr>

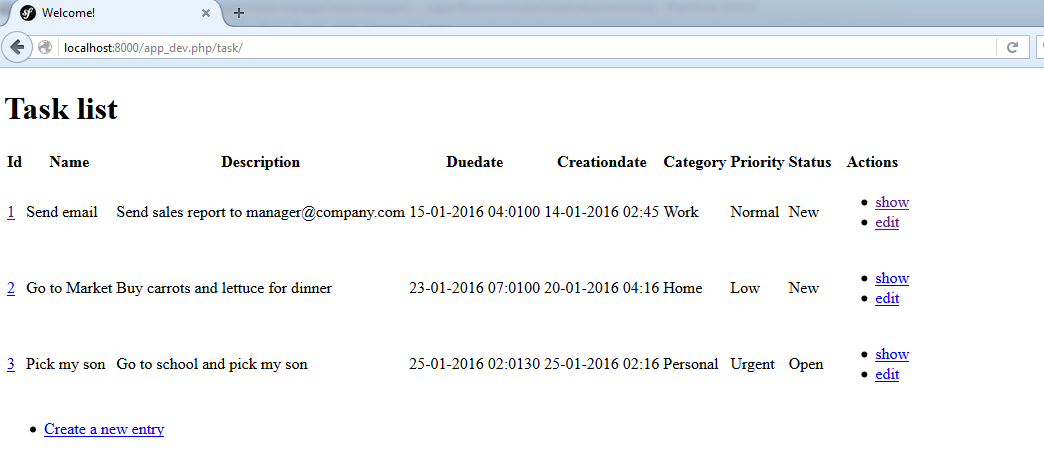
<tr>

<th>Creationdate</th>

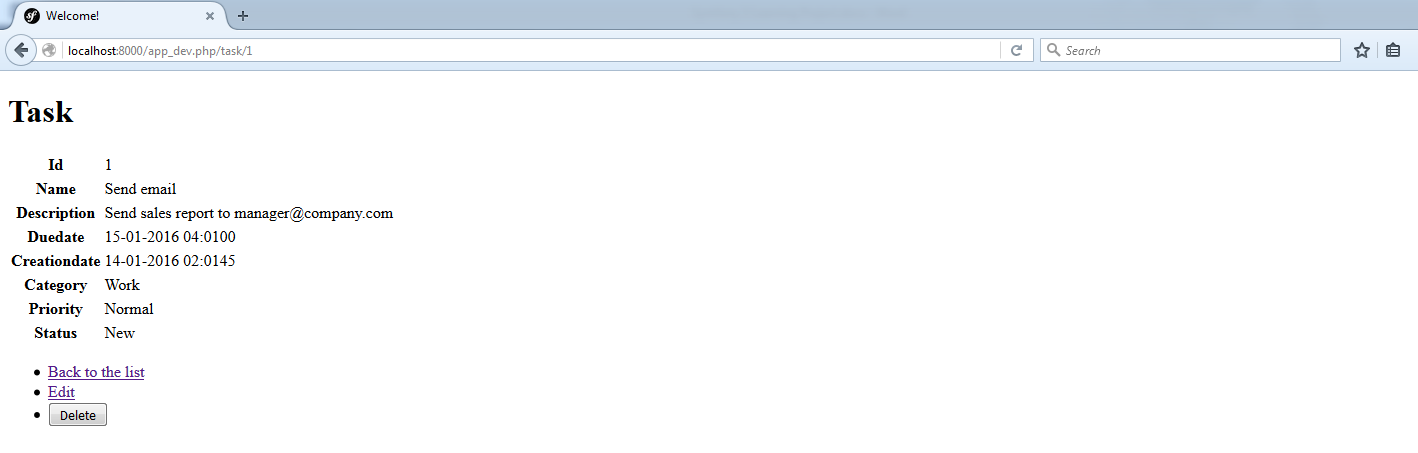
**<td>{{ task.creationDate | date('d-m-Y h:i A') }}</td>**

1. Refresh the page and check the result

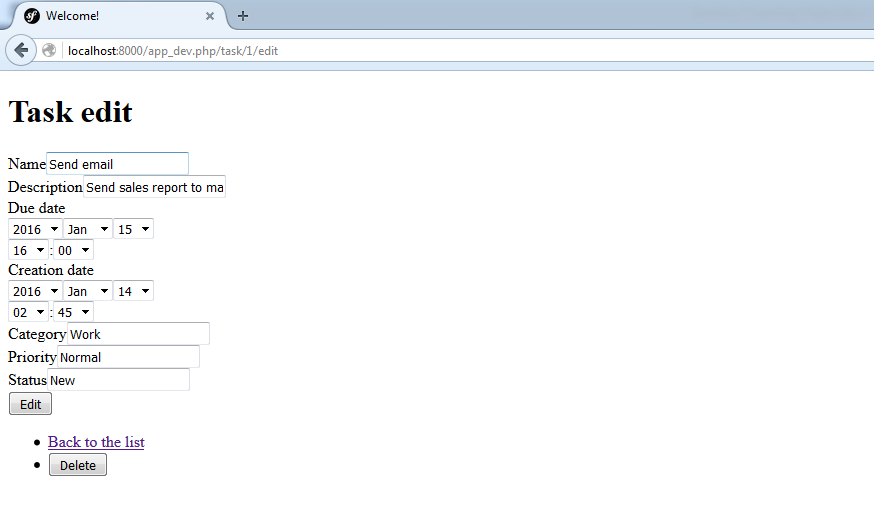
List page (app/Resources/views/task/index.html.twig)



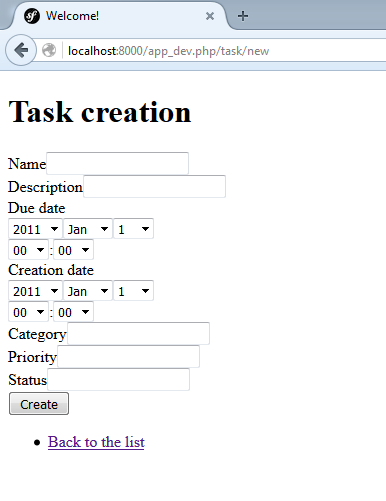
Show details page (app/Resources/views/task/show.html.twig)



Edit task page (app/Resources/views/task/edit.html.twig)



New task page (app/Resources/views/task/new.html.twig)



1. Also this command generates automatically the following files:

src/TaskManagerBundle/Controller/TaskController.php

<?php

namespace TaskManagerBundle\Controller;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

use TaskManagerBundle\Entity\Task;

use TaskManagerBundle\Form\TaskType;

/\*\*

\* Task controller.

\*

\* @Route("/task")

\*/

class TaskController extends Controller

{

/\*\*

\* Lists all Task entities.

\*

\* @Route("/", name="task\_index")

\* @Method("GET")

\*/

public function indexAction()

{

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

$tasks = $em->getRepository('TaskManagerBundle:Task')->findAll();

return $this->render('task/index.html.twig', array(

'tasks' => $tasks,

));

}

/\*\*

\* Creates a new Task entity.

\*

\* @Route("/new", name="task\_new")

\* @Method({"GET", "POST"})

\*/

public function newAction(Request $request)

{

$task = new Task();

$form = $this->createForm('TaskManagerBundle\Form\TaskType', $task);

$form->handleRequest($request);

if ($form->isSubmitted() && $form->isValid()) {

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

$em->persist($task);

$em->flush();

return $this->redirectToRoute('task\_show', array('id' => $task->getId()));

}

return $this->render('task/new.html.twig', array(

'task' => $task,

'form' => $form->createView(),

));

}

/\*\*

\* Finds and displays a Task entity.

\*

\* @Route("/{id}", name="task\_show")

\* @Method("GET")

\*/

public function showAction(Task $task)

{

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

return $this->render('task/show.html.twig', array(

'task' => $task,

'delete\_form' => $deleteForm->createView(),

));

}

/\*\*

\* Displays a form to edit an existing Task entity.

\*

\* @Route("/{id}/edit", name="task\_edit")

\* @Method({"GET", "POST"})

\*/

public function editAction(Request $request, Task $task)

{

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

$editForm = $this->createForm('TaskManagerBundle\Form\TaskType', $task);

$editForm->handleRequest($request);

if ($editForm->isSubmitted() && $editForm->isValid()) {

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

$em->persist($task);

$em->flush();

return $this->redirectToRoute('task\_edit', array('id' => $task->getId()));

}

return $this->render('task/edit.html.twig', array(

'task' => $task,

'edit\_form' => $editForm->createView(),

'delete\_form' => $deleteForm->createView(),

));

}

/\*\*

\* Deletes a Task entity.

\*

\* @Route("/{id}", name="task\_delete")

\* @Method("DELETE")

\*/

public function deleteAction(Request $request, Task $task)

{

$form = $this->createDeleteForm($task);

$form->handleRequest($request);

if ($form->isSubmitted() && $form->isValid()) {

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

$em->remove($task);

$em->flush();

}

return $this->redirectToRoute('task\_index');

}

/\*\*

\* Creates a form to delete a Task entity.

\*

\* @param Task $task The Task entity

\*

\* @return \Symfony\Component\Form\Form The form

\*/

private function createDeleteForm(Task $task)

{

return $this->createFormBuilder()

->setAction($this->generateUrl('task\_delete', array('id' => $task->getId())))

->setMethod('DELETE')

->getForm()

;

}

}

src/TaskManagerBundle/Form/TaskType.php

<?php

namespace TaskManagerBundle\Form;

use Symfony\Component\Form\AbstractType;

use Symfony\Component\Form\FormBuilderInterface;

use Symfony\Component\OptionsResolver\OptionsResolver;

class TaskType extends AbstractType

{

/\*\*

\* @param FormBuilderInterface $builder

\* @param array $options

\*/

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

{

$builder

->add('name')

->add('description')

->add('dueDate')

->add('creationDate')

->add('category')

->add('priority')

->add('status')

;

}

/\*\*

\* @param OptionsResolver $resolver

\*/

public function configureOptions(OptionsResolver $resolver)

{

$resolver->setDefaults(array(

'data\_class' => 'TaskManagerBundle\Entity\Task'

));

}

}

And src/TaskManagerBundle/Tests/Controller/TaskControllerTest.php

<?php

namespace TaskManagerBundle\Tests\Controller;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

class TaskControllerTest extends WebTestCase

{

/\*

public function testCompleteScenario()

{

// Create a new client to browse the application

$client = static::createClient();

// Create a new entry in the database

$crawler = $client->request('GET', '/task/');

$this->assertEquals(200, $client->getResponse()->getStatusCode(), "Unexpected HTTP status code for GET /task/");

$crawler = $client->click($crawler->selectLink('Create a new entry')->link());

// Fill in the form and submit it

$form = $crawler->selectButton('Create')->form(array(

'taskmanagerbundle\_task[field\_name]' => 'Test',

// ... other fields to fill

));

$client->submit($form);

$crawler = $client->followRedirect();

// Check data in the show view

$this->assertGreaterThan(0, $crawler->filter('td:contains("Test")')->count(), 'Missing element td:contains("Test")');

// Edit the entity

$crawler = $client->click($crawler->selectLink('Edit')->link());

$form = $crawler->selectButton('Update')->form(array(

'taskmanagerbundle\_task[field\_name]' => 'Foo',

// ... other fields to fill

));

$client->submit($form);

$crawler = $client->followRedirect();

// Check the element contains an attribute with value equals "Foo"

$this->assertGreaterThan(0, $crawler->filter('[value="Foo"]')->count(), 'Missing element [value="Foo"]');

// Delete the entity

$client->submit($crawler->selectButton('Delete')->form());

$crawler = $client->followRedirect();

// Check the entity has been delete on the list

$this->assertNotRegExp('/Foo/', $client->getResponse()->getContent());

}

\*/

}

1. Modify app/Resources/views/base.html.twig to add header and footer sections

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<title>{% block title %}Welcome!{% endblock %}</title>

{% block stylesheets %}{% endblock %}

<link rel="icon" type="image/x-icon" href="{{ asset('favicon.ico') }}" />

</head>

<body>

**<header>**

**<h1>Task Manager</h1>**

**</header>**

{% block body %}{% endblock %}

**<footer>**

**<address>Developers Delicias 2016. All rights reserved.</address>**

**</footer>**

{% block javascripts %}{% endblock %}

</body>

</html>

1. Modify **/app/Resources/views/task/index.html.twig** to reflect the proposed data:

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

{% block body %}

**<h2>All Tasks</h2>**

**<ul>**

**<li>**

**<a href="{{ path('task\_new') }}">New task</a>**

**</li>**

**</ul>**

<table>

<thead>

<tr>

<th>Name</th>

<th>Description</th>

**<th>Due date</th>**

<th>Actions</th>

</tr>

</thead>

<tbody>

{% for task in tasks %}

<tr>

<td>{{ task.name }}</td>

<td>{{ task.description }}</td>

**<td>{{ task.dueDate | date('Y-m-d h:i A') }}</td>**

<td>

<ul>

<li>

**<a href="{{ path('task\_show', { 'id': task.id }) }}">Details</a>**

</li>

<li>

**<a href="{{ path('task\_edit', { 'id': task.id }) }}">Edit</a>**

</li>

**<li><a href="#">Remove</a> </li>**

</ul>

</td>

</tr>

{% endfor %}

</tbody>

**<tfoot>**

**<tr>**

**<td colspan="4">Total: {{ tasks | length }}</td>**

**</tr>**

**</tfoot>**

</table>

{% endblock %}

1. Modify app/Resources/views/task/show.html.twig to reflect the required format

...

<table>

<tbody>

...

<tr>

**<th>Due date</th>**

**<td>{{ task.dueDate | date('d-m-Y h:i A') }}</td>**

</tr>

<tr>

**<th>Creation date</th>**

**<td>{{ task.creationDate | date('d-m-Y h:i A') }}</td>**

</tr>

<tr>

<th>Category</th>

<td>{{ task.category }}</td>

</tr>

...

{% endblock %}

1. Move /src/TaskManagerBundle/Tests/Controller/TaskControllerTest.php to tests/TaskManagerBundle/Controller/TaskControllerTest.php.

##### FR-01-001 Task List

###### Requirement 1: Task list should be ordered by due date ascendent

For now it could see this requirement is accomplished but if you add a new task with an older due date you can see it is shown at last position.

So for this we are going to modify our repository as follow:

<?php

namespace TaskManagerBundle\Repository;

/\*\*

\* TaskRepository

\*

\* This class was generated by the Doctrine ORM. Add your own custom

\* repository methods below.

\*/

class TaskRepository extends \Doctrine\ORM\EntityRepository

{

**public function findAll()**

**{**

**return $this->findBy(array(), array('dueDate' => 'ASC'));**

**}**

}

###### Requirement 2: Actions could be links or buttons

It was already accomplished when we edit the template

<td>

<ul>

<li>

<a href="{{ path('task\_show', { 'id': task.id }) }}">Details</a>

</li>

<li>

<a href="{{ path('task\_edit', { 'id': task.id }) }}">Edit</a>

</li>

<li><a href="#">Remove</a> </li>

</ul>

</td>

###### Requirement 3: Should show the total number of tasks

We add this requirement using a <tfoot> tag as shown

<tfoot>

<tr>

<td colspan="4">Total: {{ tasks | length }}</td>

</tr>

</tfoot>

###### Requirement 4: Due date should apply date format as shown

Add this format to the due date

<tr>

<td>{{ task.name }}</td>

<td>{{ task.description }}</td>

**<td>{{ task.dueDate | date('Y-m-d h:i A') }}</td>**

##### FR-01-001 Create new task

###### Requirement 1: Required fields should be have different style and be differentiate from the rest

Modify /app/Resources/views/base.html.twig to add this stylesheet

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<title>{% block title %}Welcome!{% endblock %}</title>

{% block stylesheets %}

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

Create this new stylesheet

Bootstrap

$ composer require twbs/bootstrap

Project configuration

Parameters.yml

config\_dev.yml

### Version 1.1

#### Release Overview

For this release we need to implement a workflow for the tasks.

The basic workflow is the following:

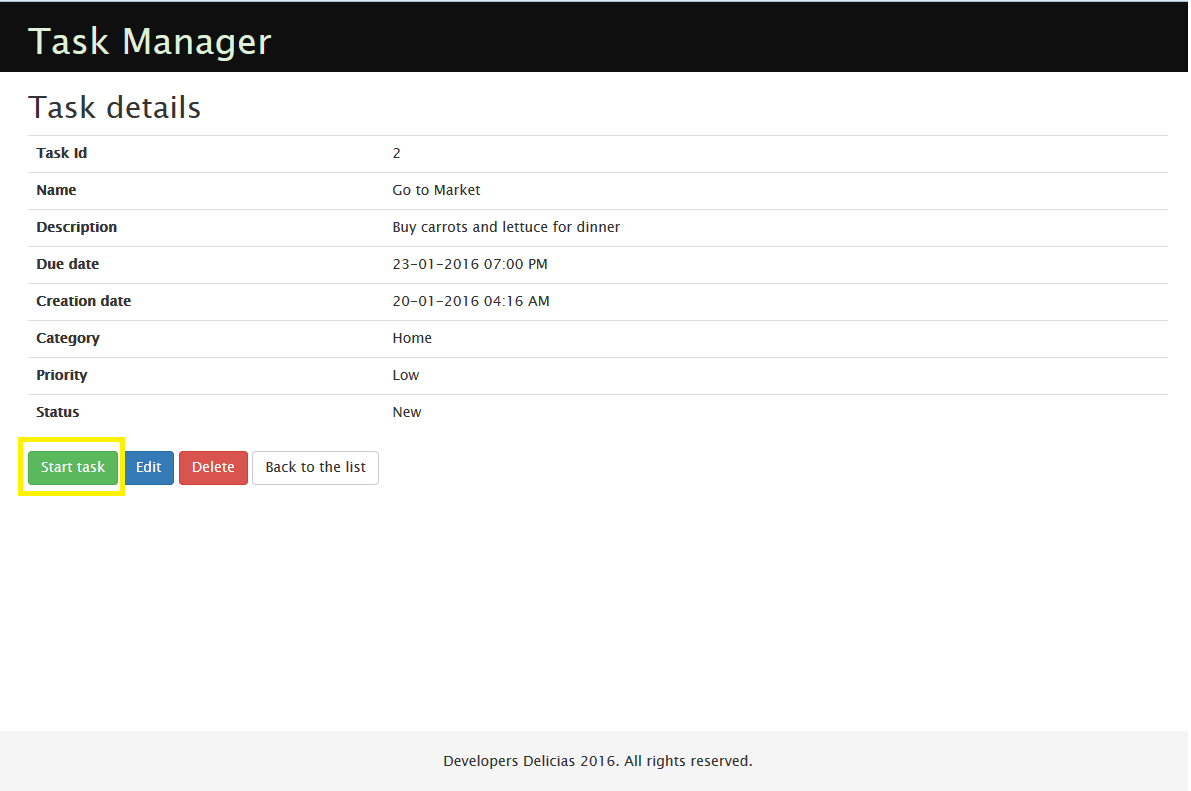
1. A user create a task and by default start in New state
2. In details page she or he can start the task and it changes to In progress state
3. When the user finishes the task he can close the task.

Also is required to add comments to the tasks.

#### Detailed requirements specifications

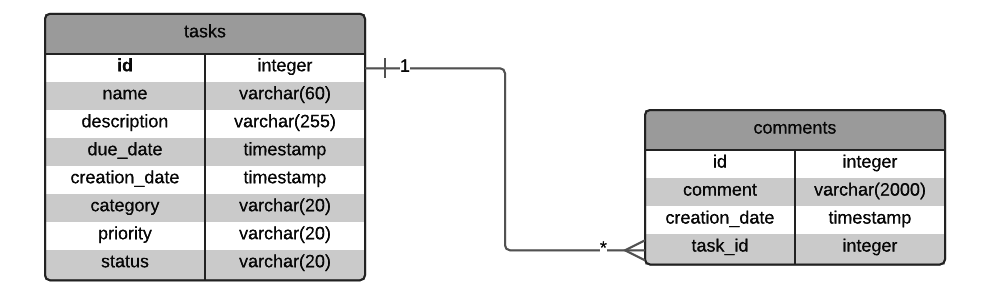
FR-1.1-001 Implement Task Workflow

Add a new button in task actions to start the task



When clicked the start button change the status to In progress and now the button should show Close tasks label

FR-1.1-002 Comments on tasks



Generate new entity

Modify values

Update entities to add setters and getters

php bin/console doctrine:generate:entities TaskManagerBundle

Create table in dev environment

php bin/console doctrine:schema:update --env=dev --force

php bin/console doctrine:fixtures:load

change code to add this list

Functional tests

$ composer require --dev liip/functional-test-bundle

### Version 1.2

#### Release Overview

Improvements in page:

* Due date time selector
* Closed tasks does not appear in task lists

### Version 1.3

#### Release Overview

User Login and assign task to users.

User login

Page should be protected by a user-login page

### Version 1.4

Release Overview

User registration

Version 1.5

1. Welcome user message
2. Associate tasks to a user
3. Only show user tasks not all
4. When a task is created it is assigned to the logged in user

Version 1.6

1. Spanish translations
2. Language selection