5. Forms - follwing

I will ask you to get the latest code from chapter 4. It contains the source code for the homework that we are going to discuss before proceeding.

```
$ git clone git@github.com:skafandri/symfony-tutorial.git --branch ch4
Cloning into 'symfony-tutorial'...
remote: Counting objects: 566, done.
remote: Compressing objects: 100% (119/119), done.
remote: Total 566 (delta 42), reused 0 (delta 0), pack-reused 443
Receiving objects: 100% (566/566), 400.63 KiB | 119.00 KiB/s, done.
Resolving deltas: 100% (255/255), done.
Checking connectivity... done.
```

Don't forget to `composer install`.

5.1 Homework 1+2 from chapter 4

We will take a look at the source code for the dashboard. You don't need to write the follwing code as it comes with the last code you checked out previously in this chapter.

We want the dashboard links to be customizable and configurable. Having only one mandatory configuration (route) and optional configurations for title, active and icon.

I would like to have a dashboard configuration file that looks like this:

- **app/config/dashboard.yml**

```
app:
    dashboard:
        items:
            account:
                route: account
                icon: account.png
            address:
                route: address
                active: false
            category:
                route: category
                icon: category.png
            contact:
                route: contact
                icon: contact.png
            country:
                route: country
                icon: country.png
            customer:
                title: clients
                route: customer
                icon: customer.png
            order:
                title: order management
                route: order
                icon: order.png
            product:
                title: product management
                route: product
                icon: product.png
            productacquisition:
                title: product acquisition
                route: productacquisition
                icon: acquisition.png
            productsale:
```

```
                title: product sales
                route: productsale
                icon: product_sale.png
            vendor:
                title: suppliers
                route: vendor
            warehouse:
                title: warehouses
                route: warehouse
                icon: warehouse.png
```

Good, to include this configuration, we will edit **app/config/dashboard.yml** and add `- { resource: dashboard.yml }` under **imports** section.

If you try to run the application now, you should get an exception `FileLoaderLoadException in FileLoader.php line 125: There is no extension able to load the configuration for "app"`.

Indeed, we must register our bundle to provide the configuration format it expects. To do this, we need to add a configuration class.

- **src/AppBundle/DependencyInjection/Configuration.php**

```php
<?php

namespace AppBundle\DependencyInjection;

use Symfony\Component\Config\Definition\Builder\TreeBuilder;
use Symfony\Component\Config\Definition\ConfigurationInterface;

class Configuration implements ConfigurationInterface
{

    public function getConfigTreeBuilder()
    {
        $treeBuilder = new TreeBuilder();
        $rootNode = $treeBuilder->root('app');

        $rootNode
            ->children()
                ->arrayNode('dashboard')
                    ->children()
                        ->arrayNode('items')
                            ->prototype('array')
                                ->children()
                                    ->scalarNode('title')
                                        ->defaultValue(null)
                                    ->end()
                                    ->scalarNode('route')->end()
                                    ->scalarNode('icon')
                                        ->defaultValue(null)
                                    ->end()
                                    ->booleanNode('active')
                                        ->defaultValue(true)
                                    ->end()
                                ->end()
                            ->end()
                        ->end()
                    ->end()
                ->end()
            ->end();

        return $treeBuilder;
    }

}
```

The exception should disappear now. We are ready to read the configuration values, but before that let's prepare some

classes that will represent a dashboard.

We will define Dashboard and DashboardItem classes so we can work with objects and not with configuration arrays.

- **src/AppBundle/Dashboard/Dashboard.php**

```php
<?php

namespace AppBundle\Dashboard;

class Dashboard
{

    private $items = array();

    public function addItems(DashboardItem $item)
    {
        $this->items[$item->getKey()] = $item;
    }

    public function getItems($includeInactive = false)
    {
        $items = array();
        foreach ($this->items as $item) {
            if ($item->isActive() || $includeInactive) {
                $items[] = $item;
            }
        }
        return $items;
    }

    public function loadFromConfiguration($configuration)
    {
        foreach ($configuration['dashboard']['items'] as $key => $item) {
            $title = $item['title'] ? $item['title'] : $key;
            $icon = $item['icon'] ? $item['icon'] : 'default.png';
            $dashboardItem = new DashboardItem();
            $dashboardItem
                    ->setKey($key)
                    ->setActive($item['active'])
                    ->setTitle($title)
                    ->setRoute($item['route'])
                    ->setIcon($icon);
            $this->items[$dashboardItem->getKey()] = $dashboardItem;
        }
    }

}
```

- **src/AppBundle/Dashboard/DashboardItem.php**

```php
<?php

namespace AppBundle\Dashboard;

class DashboardItem
{

    private $key;
    private $title;
    private $route;
    private $icon;
    private $active;

    public function getKey()
    {
        return $this->key;
    }
```

```
    public function getTitle()
    {
        return $this->title;
    }

    public function getRoute()
    {
        return $this->route;
    }

    public function getIcon()
    {
        return $this->icon;
    }

    public function isActive()
    {
        return $this->active;
    }

    public function setKey($key)
    {
        $this->key = $key;
        return $this;
    }

    public function setTitle($title)
    {
        $this->title = $title;
        return $this;
    }

    public function setRoute($route)
    {
        $this->route = $route;
        return $this;
    }

    public function setIcon($icon)
    {
        $this->icon = $icon;
        return $this;
    }

    public function setActive($active)
    {
        $this->active = $active;
        return $this;
    }

}
```

Now we will parse the config values and construct a Dashboard object from it.

- **src/AppBundle/DependencyInjection/AppExtension.php**

```
<?php

namespace AppBundle\DependencyInjection;

use Symfony\Component\DependencyInjection\ContainerBuilder;
use Symfony\Component\HttpKernel\DependencyInjection\Extension;

class AppExtension extends Extension
{

    public function load(array $configs, ContainerBuilder $container)
    {
```

```
        $configuration = new Configuration();
        $appConfig = $this->processConfiguration($configuration, $configs);

        $dashboardDefinition = $container->register('app.dashboard', 'AppBundle\Dashboard\Dashboard');
        $dashboardDefinition->addMethodCall('loadFromConfiguration', array($appConfig));
    }

}
```

Let's define a simple controller and a route to handle the dashboard action:

- **src/AppBundle/Controller/DashboardController.php**

```
<?php

namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class DashboardController extends Controller
{

    public function indexAction()
    {
        if ($this->container->has('profiler')) {
            $this->container->get('profiler')->disable();
        }

        $dashboard = $this->get('app.dashboard');

        return $this->render(
                        'AppBundle:Dashboard:index.html.twig', array('items' => $dashboard->getItems())
        );
    }

}
```

- **src/AppBundle/Resources/config/routing/dashboard.yml**

```
dashboard:
    path:     /
    defaults: { _controller: "AppBundle:Dashboard:index" }
```

- Edit **src/AppBundle/Resources/config/routing.yml** and add:

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

We need a simple template to render our dashboard

- **src/AppBundle/Resources/views/Dashboard/index.html.twig**

```
{% extends 'base.html.twig' %}
{% block javascripts %}
    {{ parent() }}
    <script type="text/javascript"
            src="{{asset('bundles/app/js/jqDesktop.js')}}" >
    </script>
    <script type="text/javascript"
            src="{{asset('bundles/app/js/dashboard.js')}}" >
    </script>
{% endblock %}

{% block stylesheets %}
    {{parent()}}
    <link rel="stylesheet"
```

```
          href="{{asset('bundles/app/css/dashboard.css')}}" >
{% endblock %}

{% block body %}
    <div id="dashboard"></div>
    {% for item in items %}
        <div class="window"
          {% include 'AppBundle:Dashboard:item_attributes.html.twig'
          with {item:item} %} >
        </div>
    {% endfor %}
{% endblock %}
```

- **src/AppBundle/Resources/views/Dashboard/item_attributes.html.twig**

```
title="{{ item.title }}"
data-route="{{ path(item.route)}}"
data-icon="{{ asset('bundles/app/images/dashboard/' ~ item.icon) }}"
```

For this task, we are going to use JQDesktop, a JQuery plugin developed by me. Is open source and available at https://github.com/skafandri/jqdesktop

Download https://raw.githubusercontent.com/skafandri/jqdesktop/master/min.jquery.jqDesktop.js and save it under **src/AppBundle/Resources/public/js/jqDesktop.js**

Some JavaScript to use JQDesktop plugin

- **src/AppBundle/Resources/public/js/dashboard.js**

```
$(function () {
    $('#dashboard').jqDesktop({
        iconWidth: 70,
        iconHeight: 70
    });
    $('.window').each(function () {
        var route = $(this).data('route');
        var jqWindow = $(this).jqWindow({
            icon: $(this).data('icon'),
            width: 800,
            height: 500}
                );
        $('#dashboard').jqDesktop('addWindow', jqWindow);
        jqWindow[0].bind('windowFirstOpen', function () {
            $(this).jqWindow('setContent', '<iframe src="' + route + '" ></iframe>');
        });
    });

});
```

Some CSS to style our dashboard

- **src/AppBundle/Resources/public/css/dashboard.css**

```
.jqdesktop-content {
    background: -webkit-radial-gradient(
        closest-side,
        rgba(232,233,242,1) 0,
        rgba(167,172,201,1) 53%,
        rgba(126,135,178,1) 98%,
        rgba(126,135,178,1) 100%
        );
    background: -moz-radial-gradient(
        closest-side,
        rgba(232,233,242,1) 0,
        rgba(167,172,201,1) 53%,
        rgba(126,135,178,1) 98%,
        rgba(126,135,178,1) 100%
```

```
        );
    background: radial-gradient(
        closest-side,
        rgba(232,233,242,1) 0,
        rgba(167,172,201,1) 53%,
        rgba(126,135,178,1) 98%,
        rgba(126,135,178,1) 100%
        );
}

#dashboard {
    height: 100%;
}

html, body {
    height:100%;
    overflow: hidden;
    margin:0px;
    padding:0px;
    font-size: 12px;
}

.jqdesktop-window-content iframe {
    width: 100%;
    height: 100%;
    border: none;
}

.jqdesktop-icon-container {
    margin: 5px;
}
```

Done, we will just add some images to give our dashboard a style. You can find the images used at https://github.com/skafandri/symfony-tutorial/tree/ch4/src/AppBundle/Resources/public/images/dashboard

You can visit http://127.0.0.1:8000/dashboard/ and see the result. You can rearrange the icons, open/close/move/maximize/minimize the windows. You will notice that we don't need a link to the dashboard from every listing anymore.

5.2 Datepicker

In our application we have select to enter a date, which is not very handy. We will use the JQuery datepicker widget to edit dates. We will create a new form type that handles date fields.

- Create **src/AppBundle/Form/DatePickerType.php**

```
<?php

namespace AppBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\OptionsResolver\OptionsResolverInterface;

class DatePickerType extends AbstractType
{

    public function setDefaultOptions(OptionsResolverInterface $resolver)
    {
        $resolver->setDefaults(array(
            'widget' => 'single_text'
        ));
    }

    public function getParent()
    {
        return 'date';
```

```
    }

    public function getName()
    {
        return 'datePicker';
    }
}
```

- Edit **src/AppBundle/Form/ProductAcquisitionType.php** change `->add('date')` to `->add('date', new DatePickerType())`
- Edit **src/AppBundle/Resources/views/Form/fields.html.twig**
  add the following block

```
{% block datePicker_widget %}
        <input type="text" {{ block('widget_attributes') }}
               {% if value is not empty %}value="{{ value }}" {% endif %} class="datepicker"/>
{% endblock datePicker_widget %}
```

- Edit **src/AppBundle/Resources/public/js/app.js** and update setupUi method so it looks like

```
function setupUi() {
    $(':input').addClass('ui-widget-content');
    $('input[type="submit"], .button, #footer a').button();
    $('th').addClass('ui-widget-header');
    $('.tree').menu();
    $('.link-dialog').click(function () {
        showLinkDialog(this);
        return false;
    });

    $('.datepicker').on('focus', function () {
        $(this).datepicker({
            dateFormat: "yy-mm-dd"
        }).datepicker('show');
    });
}
```

Done, now the product acquisition forms use datepicker widget to input the date.

Let's transform all checkboxes into buttons.

- Edit **src/AppBundle/Resources/public/js/app.js**
  add `$( ":checkbox" ).button();` at the end of setupUi method.

5.3 Prevent deleting a category that has products

We want to prevent the user from deleting a category that has at least one product.

Let's start by adding this constraint to the entity.

- Edit **src/AppBundle/Entity/Category.php**

Add `@ORM\HasLifecycleCallbacks` after `@ORM\Entity`

Add the follwing method

```
/**
 * @ORM\PreRemove
 */
public function preRemove()
{
    if (count($this->getProduct())) {
        throw new LogicException('Cannot remove a category that has products');
    }
}
```

- Create **src/AppBundle/Exception/AppException.php**

```
<?php

namespace AppBundle\Exception;

class AppException extends \Exception
{

}
```

- Create **src/AppBundle/Exception/LogicException.php**

```
<?php

namespace AppBundle\Exception;

class LogicException extends AppException
{

}
```

Good, now a user cannot delete a category if it has at least one product. The user will get an ugly exception if the application runs in dev mode or a 500 HTTP error if it is running in prod mode.

We want to display an error message for the end user. We will cath the exception in the controller and add it as a flash message.

- Edit **src/AppBundle/Controller/CategoryController.php** and update the deleteAction as following

```
public function deleteAction(Request $request, $id)
{
    $form = $this->createDeleteForm($id);
    $form->handleRequest($request);

    if ($form->isValid()) {
        $em = $this->getDoctrine()->getManager();
        $entity = $em->getRepository('AppBundle:Category')->find($id);

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

        try {
            $em->remove($entity);
            $em->flush();
        } catch (AppException $exception) {
            $this->addFlash('warning', $exception->getMessage());
            return $this->updateAction($request, $id);
        }
    }

    return $this->redirect($this->generateUrl('category'));
}
```

Don't forget to `use AppBundle\Exception\AppException;`

- Edit **app/Resources/views/common/app.html.twig** and add `{% include 'common/flash_messages.html.twig' %}` before `{% block content %}`

- Create **app/Resources/views/common/flash_messages.html.twig**

```
{% for flashMessage in app.session.flashbag.get('warning') %}
    <div class="ui-state-error">
        {{ flashMessage }}
```

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

Now you should see a red error message if you try to remove a category which has at least one product.

5.4 Soft delete

When we created the address table, we added a *deleted* column to achieve a soft delete effect. We will implement this feature now.

First, let's enable the address item to be visible in our dashboard. This step is not mandatory, but we need it to test the new functionality.

- Edit **app/config/dashboard.yml** and set `active: true` for the *address* item.
- Create **src/AppBundle/Event/Listener/SoftDelete.php**

```
<?php

namespace AppBundle\Event\Listener;

use Doctrine\ORM\Event\OnFlushEventArgs;

class SoftDelete
{
    public function onFlush(OnFlushEventArgs $args)
    {
        $unitOfWork = $args->getEntityManager()->getUnitOfWork();
        foreach ($unitOfWork->getScheduledEntityDeletions() as $entity) {
            if (is_callable(array($entity, 'setDeleted'))) {
                $entity->setDeleted(true);
                $unitOfWork->propertyChanged($entity, 'deleted', false, true);
                $unitOfWork->scheduleExtraUpdate($entity, array(
                    'deleted' => array(false, true)
                ));
                $args->getEntityManager()->persist($entity);
            }
        }
    }
}
```

- Edit **app/config/services.yml** empty the file and put:

```
imports:
    - { resource: "@AppBundle/Resources/config/services.yml" }
```

- Create **src/AppBundle/Resources/config/services.yml**

```
services:
    app.listener.soft_delete:
        class: AppBundle\Event\Listener\SoftDelete
        tags:
            - { name: doctrine.event_listener, event: onFlush }
```