

## TD n°3



- Projet **boards**
- Application gestion de projets SCRUM

### Objectifs

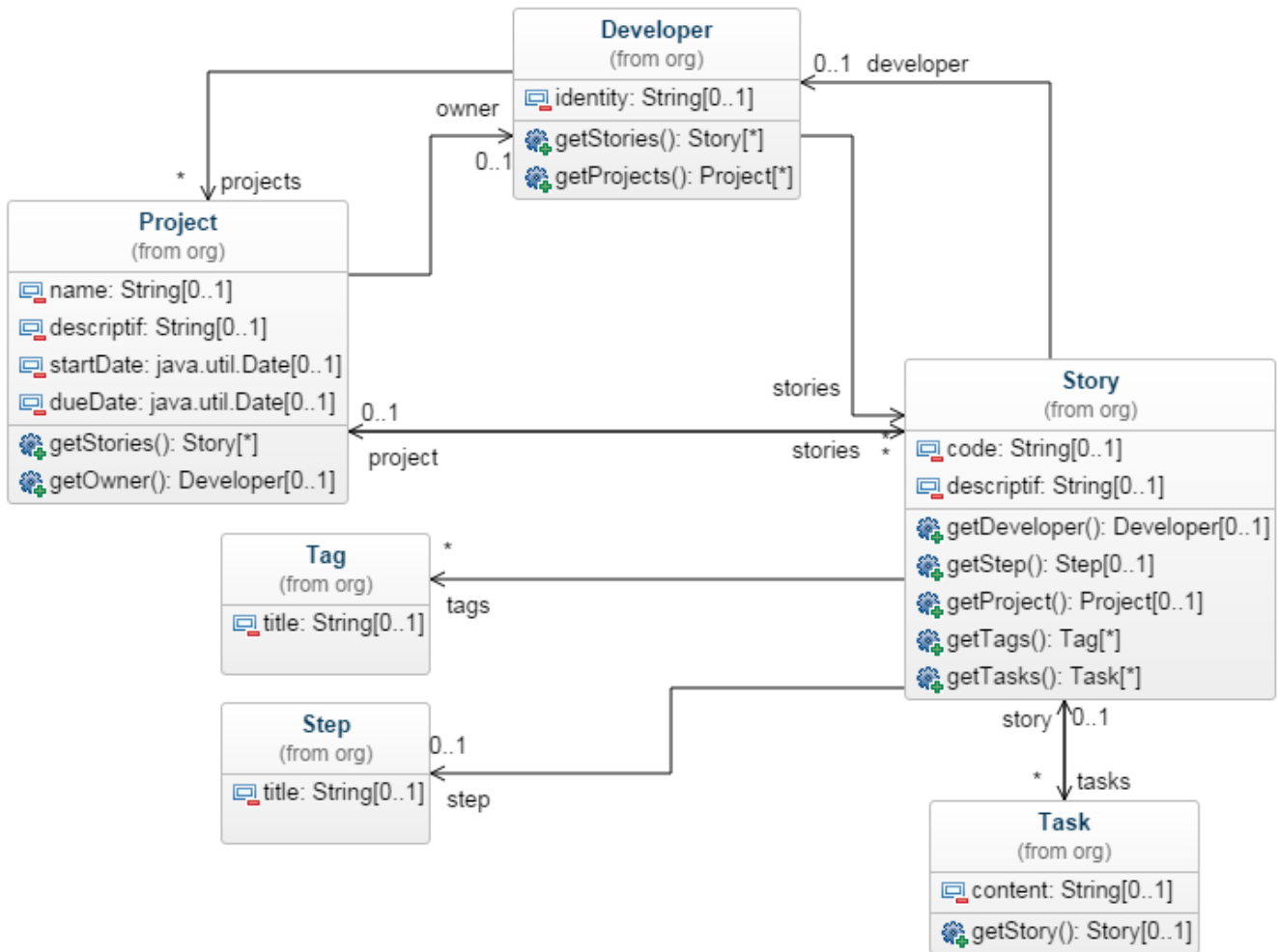
1. Bases symfony + Doctrine
2. Requêtes Ajax, Composants

### Contexte

Vous travaillez sur un outil permettant de gérer des projets.

Voici les principales caractéristiques du système d'information :

- Chaque projet [**project**] possède un nom, un descriptif, une date de début et de fin, et un propriétaire (owner, qui est un développeur)
- L'équipe est constituée d'un ensemble de développeurs [**developer**].
- Chaque User story [**story**] a un code et un descriptif, et appartient à un projet.
- Il est possible de lui apposer des tags [**tags**], composés d'une couleur et d'un label.
- Elle peut être affectée à un développeur [**dev**] (qui a juste une identité).
- Elle peut contenir une liste de tâches [**tasks**], à réaliser ou réalisées.



## Création du projet, intégration des composants

Exécuter le script de création de la base de données :

```
mysql -u root < path/to/boards.sql
```

Créer le projet **boards**

```
composer create-project symfony/skeleton boards
```

A partir du dossier de l'application, intégrer les principaux composants :

```
composer require annotations twig doctrine maker asset
```

Configurer la connexion à la base de données dans le fichier **.env**

Générer les classes métier :

```
php bin/console doctrine:mapping:convert --from-database annotation ./src/Entity
```

Ajouter et lancer le serveur de développement :

```
composer require server --dev
php bin/console server:run
```

## Intégration Semantic

Intégrer **phpMv-UI toolkit** :

```
composer require phpmv/php-mv-ui 2.3.x-dev
```

- Télécharger [Semantic-UI](#), dézipper l'archive dans le dossier **public/assets** (à créer)
- Télécharger [Jquery](#)

Modifier le template de base : **templates/base.html.twig**

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>{% block title %}Welcome!{% endblock %}</title>
    {% block stylesheets %}
    <link rel="stylesheet" type="text/css" href="{{
asset('assets/semantic.min.css') }}">
    {% endblock %}
  </head>
  <body>
    <div class="ui container">
    {% block body %}{% endblock %}
    </div>
    {% block javascripts %}
    <script src="{{ asset('assets/jquery-3.3.1.min.js') }}"></script>
    <script src="{{ asset('assets/semantic.min.js') }}"></script>
    {% endblock %}
  </body>
</html>
```

## Classes métier

Modifier les classes métier :

- Renommer les instances/collections associées ex: **\$idowner** devient **\$owner** dans la classe **Project**
- Intégrer les classes au namespace **App\Entity**
- Générer les accesseurs

Exemple pour la classe **Project** :

```
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use Doctrine\Common\Collections\ArrayCollection;

/**
 * Project
 *
 * @ORM\Table(name="project",
uniqueConstraints={@ORM\UniqueConstraint(name="projectName", columns={"name"})},
indexes={@ORM\Index(name="idOwner", columns={"idOwner"})})
 * @ORM\Entity
 */
class Project
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="name", type="string", length=100, nullable=false)
     */
    private $name;

    /**
     * @var string
     *
     * @ORM\Column(name="descriptif", type="text", length=65535, nullable=false)
     */
    private $descriptif;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="startDate", type="date", nullable=false)
     */
    private $startdate;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="dueDate", type="date", nullable=false)
     */
    private $duedate;

    /**
     * @var \Developer
     *
     * @ORM\ManyToOne(targetEntity="Developer")
     */
}
```

```

    * @ORM\JoinColumns({
    *   @ORM\JoinColumn(name="idOwner", referencedColumnName="id")
    * })
    */
private $owner;

/**
 * @ORM\OneToMany(targetEntity="App\Entity\Story", mappedBy="project")
 */
private $stories;
public function __construct(){
    $this->stories=new ArrayCollection();
}
}

```

Créer le repository **ProjectRepository** :

```

namespace App\Repository;

use App\Entity\Project;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Symfony\Bridge\Doctrine\RegistryInterface;

class ProjectRepository extends ServiceEntityRepository
{
    public function __construct(RegistryInterface $registry)
    {
        parent::__construct($registry, Project::class);
    }
}

```

## Tests phpMv-UI

### Service

Créer la classe **ProjectsGui** dans le dossier **src/Services/semantic**

```

namespace App\Services\semantic;

use Ajax\php\symfony\JquerySemantic;

class ProjectsGui extends JquerySemantic{
}

```

### Contrôleur

Créer le contrôleur Projects :



```
php bin/console make:controller Projects
```

## Test Ajax

Créer une méthode dans le service `ProjectsGui` permettant de générer un bouton :

Le click sur le bouton effectuera une requête ajax vers l'adresse `/projects` dont le résultat sera affiché dans une zone HTML d'id `response` :

```
namespace App\Services\semantic;

use Ajax\php\symfony\JquerySemantic;

class ProjectsGui extends JquerySemantic{
    public function button(){
        $bt=$this->semantic()->htmlButton("btProjects","Projets","orange");
        $bt->getOnClick($this->getUrl("/projects"),"#response",["attr"=>""]);
        return $bt;
    }
}
```

Injecter le service `ProjectsGui` en le passant en paramètre de la méthode `index` :

La méthode `index` appelle la génération du bouton et affiche la vue `Projets/index.html.twig`

```
namespace App\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Response;
use App\Services\semantic\ProjectsGui;

class ProjectsController extends Controller{
    /**
     * @Route("/index", name="index")
     */
    public function index(ProjectsGui $gui){
        $gui->button();
        return $gui->renderView('Projects/index.html.twig');
    }
}
```

Créer le template `Projects/index.html.twig` :

```
{% extends "base.html.twig" %}
{% block body %}
    {{ q["btProjects"] | raw }}
    <div id="response" class="ui segment"></div>
{% endblock %}
{% block javascripts %}
    {{ parent() }}
    {{ script_foot | raw }}
{% endblock %}
```

Ajouter la route **/projects** pour afficher les projets :

```
...
/**
 * @Route("/projects", name="projects")
 */
public function all(ProjectRepository $projectRepo){
    $projects=$projectRepo->findAll();
    return $this->render('Projects/all.html.twig',["projects"=>$projects]);
}
```

Le template **Projects/all.html.twig** affiche la liste des projets :

```
<div class="ui inverted segment">
  <div class="ui inverted relaxed divided list">
    {% for project in projects %}
      <div class="item">
        <div class="content">
          <div class="header">{{ project.name }}</div>
          {{ project.descriptif }}
        </div>
      </div>
    {% endfor %}
  </div>
</div>
```

## Composant HtmlButtonGroups

Créer la méthode **buttons** dans la classe **ProjectsGui** :

```
...
public function buttons(){
    $bts=$this->_semantic->htmlButtonGroups("bts",["Projects","Tags"]);
    $bts->addIcons(["folder","tags"]);
    $bts->setPropertyValues("data-url", ["projects","tags"]);
    $bts->getOnClick("", "#response", ["attr"=>"data-url"]);
}
```

Le click sur chaque bouton effectue une requête ajax vers l'url définie dans **data-url** dont le résultat est afficher dans l'élément HTML d'id **response**

Modifier le template **Projects/index.html.twig** pour qu'il affiche les boutons :

```
{% extends "base.html.twig" %}
{% block body %}
    {{ q["bts"] | raw }}
    <div id="response" class="ui segment"></div>
{% endblock %}
```

```
{% block javascripts %}
    {{ parent() }}
    {{ script_foot | raw }}
{% endblock %}
```

Modifier l'action **index** pour qu'elle appelle la méthode **buttons** de **gui** :

```
class ProjectsController extends Controller{
    /**
     * @Route("/index", name="index")
     */
    public function index(ProjectsGui $gui){
        $gui->buttons();
        return $gui->renderView('Projects/index.html.twig');
    }
    ...
}
```

## Composant DataTable

Créer la classe **TagsGui** dans **src/Services** :

```
namespace App\Services\semantic;

use Ajax\php\symfony\JquerySemantic;

class TagsGui extends JquerySemantic{
    public function dataTable($tags){
        $dt=$this->_semantic->dataTable("dtTags", "App\Entity\Tag", $tags);
        return $dt;
    }
}
```

Créer la classe Repository pour les Tags :

```
namespace App\Repository;

use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Symfony\Bridge\Doctrine\RegistryInterface;
use App\Entity\Tag;

class TagRepository extends ServiceEntityRepository
{
    public function __construct(RegistryInterface $registry)
    {
        parent::__construct($registry, Tag::class);
    }
}
```

Ajouter la route **tags** dans le contrôleur **ProjectsController** :

```
...  
/**  
 * @Route("/tags", name="tags")  
 */  
public function tags(TagsGui $gui, TagRepository $tagRepo){  
    $tags=$tagRepo->findAll();  
    $dt=$gui->dataTable($tags);  
    return new Response($dt);  
}
```

Amélioration de l'affichage des tags :

```
namespace App\Services\semantic;  
  
use Ajax\php\symfony\JquerySemantic;  
use Ajax\semantic\html\elements\HtmlLabel;  
  
class TagsGui extends JquerySemantic{  
    public function dataTable($tags){  
        $dt=$this->_semantic->dataTable("dtTags", "App\Entity\Tag", $tags);  
        $dt->setFields(["tag"]);  
        $dt->setCaptions(["Tag"]);  
        $dt->setValueFunction("tag", function($v,$tag){  
            $lbl=new HtmlLabel("", $tag->getTitle());  
            $lbl->setColor($tag->getColor());  
            return $lbl;  
        });  
        return $dt;  
    }  
}
```

From:  
<http://slamwiki2.kobject.net/> - **SlamWiki 2.1**

Permanent link:  
<http://slamwiki2.kobject.net/framework-web/symfony/td3?rev=1519171064>

Last update: **2019/08/31 14:43**

