TD n°6

Ce projet utilise le repository github : boards-symfony.

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

Objectifs

1. Factorisation du code
2. Réutilisation

Prise en main

Vous pouvez au choix :

- Partir de votre propre projet et y inclure les éléments du projet Github
- Partir du projet GitHub pour y intégrer vos propres fonctionnalités (recommandé)

A partir du dossier du projet, exécuter :

```
composer update
```

Dans le fichier **.env**, ajuster la valeur de la variable **DATABASE_URL**.

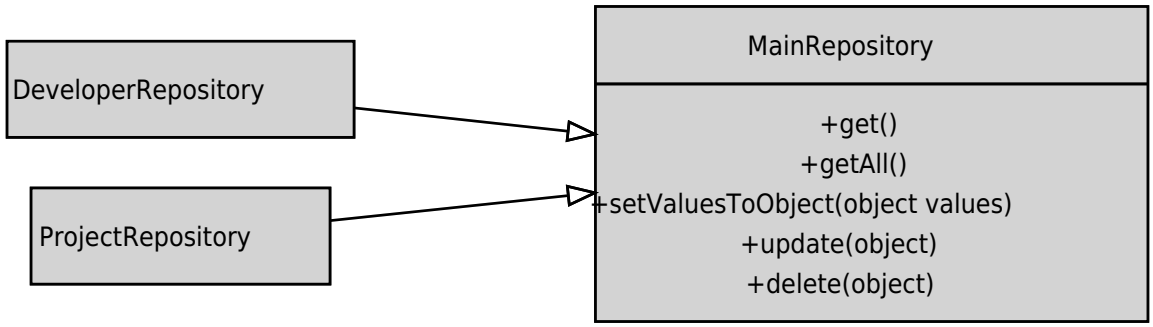
Démarrer le serveur (Mysql) et le serveur http de dévelopement :

```
php bin/console server:run
```

Factorisation du code

Doctrine repositories

Les repositories de chacun des models héritent de **MainRepository**, définissant les opérations de base CRUD sur les objets.



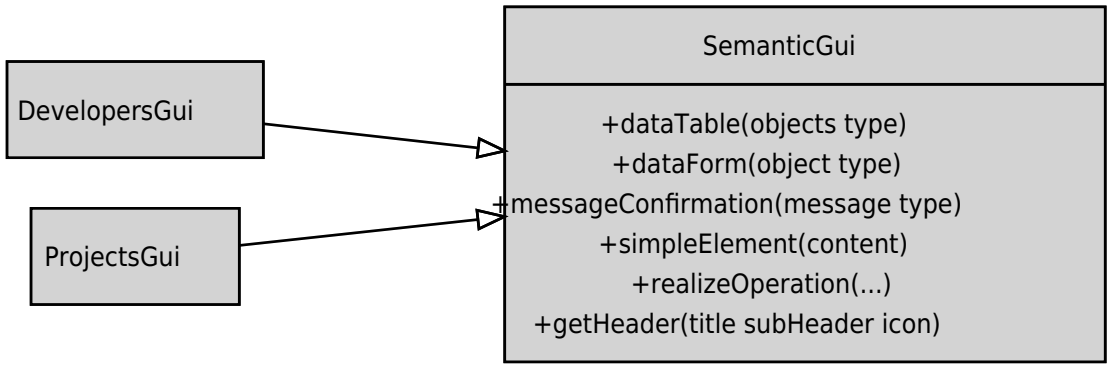
CREATED WITH YUML

Chaque Repository héritant de MainRepository a juste à définir dans son constructeur la classe métier auquel il correspond :

```
<?php

namespace App\Repository;

use Symfony\Bridge\Doctrine\RegistryInterface;
use App\Entity\Developer;

class DeveloperRepository extends MainRepository{
    public function __construct(RegistryInterface $registry){
        parent::__construct($registry, Developer::class);
    }
}
```

Services Semantic

Les services Semantic de chacun des models héritent de **SemanticGui**, classe définissant les opérations utilisables dans tous les contrôleurs.

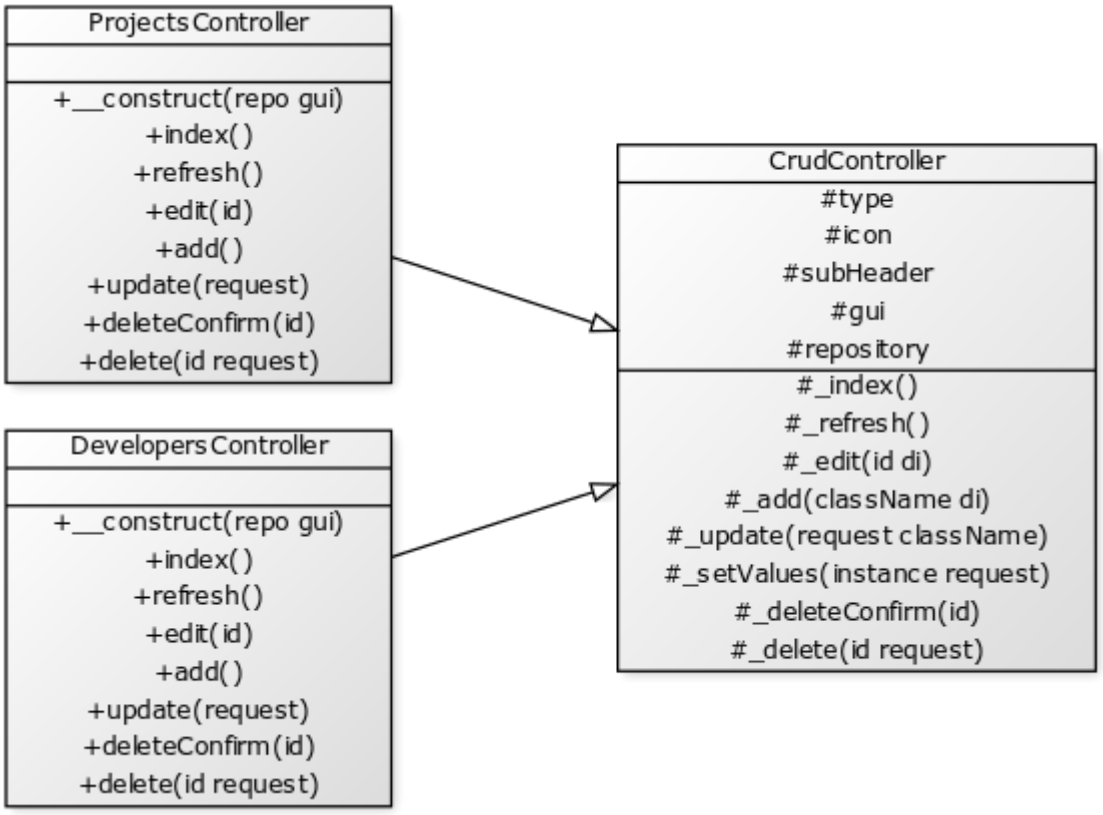


CREATED WITH YUML

Chaque Classe Gui dérivée de **SemanticGui** doit logiquement redéfinir les méthodes **dataTable** et **dataForm**.

CRUD contrôleurs

Chaque contrôleur pilotant une classe métier hérite de **CrudController**, qui définit l'ensemble des opérations CRUD.



- Chaque Repository héritant de MainRepository utilise les méthodes protégées de **CrudController**.
- Le constructeur doit initialiser par injection de dépendances une instance de **MainRepository** et de **SemanticGui** dérivées dans son constructeur.

```
<?php
class DevelopersController extends CrudController{
    public function __construct(DevelopersGui $gui,DeveloperRepository $repo){
        $this->gui=$gui;
        $this->repository=$repo;
        $this->type="developers";
        $this->subHeader="Developer list";
        $this->icon="users";
    }
    /**
     * @Route("/developers", name="developers")
     */
    public function index(){
        return $this->_index();
    }
    /**
     * @Route("/developers/refresh", name="developers_refresh")
     */
    public function refresh(){
        return $this->_refresh();
    }
```

```
    /**
     * @Route("/developers/edit/{id}", name="developers_edit")
     */
    public function edit($id){
        return $this->_edit($id);
    }
    /**
     * @Route("/developers/new", name="developers_new")
     */
    public function add(){
        return $this->_add("\App\Entity\Developer");
    }

    /**
     * @Route("/developers/update", name="developers_update")
     */
    public function update(Request $request){
        return $this->_update($request, "\App\Entity\Developer");
    }
    /**
     * @Route("/developers/confirmDelete/{id}", name="developers_confirm_delete")
     */
    public function deleteConfirm($id){
        return $this->_deleteConfirm($id);
    }
    /**
     * @Route("/developers/delete/{id}", name="developers_delete")
     */
    public function delete($id,Request $request){
        return $this->_delete($id, $request);
    }
}
```

Fonctionnalités à implémenter

CRUD

//TODO 1.1

Pour les models **Tag**, **Step**, **Task**, ajouter les fonctionnalités de base CRUD :

- Listage des instances dans une table
- Suppression
- Modification
- Ajout

Consignes :

- Respecter la logique fonctionnelle et structurelle (implémentation) mise en place dans le projet initial
- Factoriser au mieux le code

Route index

//TODO 1.2

Modifier la route **index**, pour qu'elle affiche les éléments suivants, et qu'elle permette d'accéder à chacune des parties :