Introduction

Here are some file to present my Symfony experience. They are from a project
Which which you can send log statement to this server via an API.
On the web side you can read the message and set mail alerts for certain type logs.

Services.yml

```
<?php


services:

#####################################################################
# Base class of all manager classes in folder AppBundle\Manager\Manager
# All Manager classes, except memeber of AppBundle\Manager\Manager
# should inherit from this
#
     AppBundle.Manager:
          class: AppBundle\Manager\Manager
          abstract: true
#####################################################################
# Member classes of class AppBundle.Manager
#
#
#http exeption manager
     AppBundle.httpExceptionManager:
           class: AppBundle\Manager\HttpExceptionManager

# Manager for the table log staments on this server to the log files
     AppBundle.MonoLogManager:
          class: AppBundle\Manager\MonoLog
          arguments: ['@logger']

#####################################################################
# Base class of all entity Managers
# All entity managers should inherit form this class.

     AppBundle.EntityManager:
          class: AppBundle\Manager\Manager\EntityManager
          abstract: true



#Manager for the table clients_systems
     AppBundle.ClientSystemsManager:
          class: AppBundle\Manager\Manager\Entity\ClientSystems
          calls:
```

```
            - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]
            - [setEntityManagerParams, ['@validator', 'systemid',
'AppBundle:ClientSystems']]

# Manager for the table logs
    AppBundle.LogsManager:
        class: AppBundle\Manager\Manager\Entity\Logs
        calls:
            - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]
            - [setEntityManagerParams, ['@validator', 'logid', 'AppBundle:Logs']]

# Manager for the table logs_types
    AppBundle.LogsTypesManager:
        class: AppBundle\Manager\Manager\Entity\LogsTypes
        calls:
            - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]
            - [setEntityManagerParams, ['@validator', 'id', 'AppBundle:LogsTypes']]

# Manager for the table logs_levels
    AppBundle.LogsLevelsManager:
        class: AppBundle\Manager\Manager\Entity\LogsLevels
        calls:
            - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]
            - [setEntityManagerParams, ['@validator', 'id', 'AppBundle:LogsLevels']]



# Manager for the table app_users
    AppBundle.UserManager:
        class: AppBundle\Manager\Manager\Entity\User
        calls:
            - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]
            - [setEntityManagerParams, ['@validator', 'id', 'AppBundle:User']]



# Manager for the table mail_alert
    AppBundle.MailAlert:
        class: AppBundle\Manager\Manager\Entity\MailAlert
        arguments: ['@AppBundle.MailAlertFilter']
        calls:
            - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]
            - [setEntityManagerParams, ['@validator', 'mailAlertId',
'AppBundle:MailAlert']]
```

```
# Manager for the table mail_alert_filter
     AppBundle.MailAlertFilter:
          class: AppBundle\Manager\Manager\Entity\MailAlertFilter
          calls:
             - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]
             - [setEntityManagerParams, ['@validator', 'mailAlertFilterId',
'AppBundle:MailAlertFilter']]



# Manager for the table roles
     AppBundle.RolesManager:
          class: AppBundle\Manager\Manager\Entity\Roles
          calls:
             - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]
             - [setEntityManagerParams, ['@validator', 'id', 'AppBundle:Roles']]

#################################################################
# Base class of all controller managers
# Contorller manager include functions of the controllers
#
# All contoller managera should inherit form this class.
#
     AppBundle.ControllerManager:
          class: AppBundle\Manager\Manager\ControllerManager
          abstract: true

     AppBundle.LogsListControllerManager:
          class: AppBundle\Manager\Manager\Controller\LogsListController
          calls:
             - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]

     AppBundle.MailAlertsController:
          class: AppBundle\Manager\Manager\Controller\MailAlertsController
          calls:
             - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]

     AppBundle.LoginController:
          class: AppBundle\Manager\Manager\Controller\LoginController
          calls:
             - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]

     AppBundle.UsersControllerManager:
          class: AppBundle\Manager\Manager\Controller\UsersController
          calls:
```

```
            - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]

    AppBundle.SystemsControllerManager:
        class: AppBundle\Manager\Manager\Controller\SystemsController
        calls:
            - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]

    AppBundle.LogLevelsControllerManager:
        class: AppBundle\Manager\Manager\Controller\LogLevelsController
        calls:
            - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]


#################################################################
# Misc classes in folder AppBundle\Manager\Manager
#


#Authorization manager
#A class for API auhtorization
    AppBundle.ApiAuthorizationManager:
         class: AppBundle\Manager\Manager\ApiAuthorization
         arguments: ['@AppBundle.ClientSystemsManager']
         calls:
            - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]

#FrontEndPaging manager
#A class to manage an item list with paging and sort
    AppBundle.FrontEndPagingManager:
         class: AppBundle\Manager\Manager\FrontEndPaging
         calls:
            - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]


#Form manager
#A class to render symfony forms
    AppBundle.FormManager:
         class: AppBundle\Manager\Manager\Form
         calls:
            - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]



#Mail manager
#A class to send emails
    AppBundle.EmailManager:
```

```
          class: AppBundle\Manager\Manager\Email
          calls:
             - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]


#Menu manager
#A class to manage the menus
     AppBundle.menuManager:
          class: AppBundle\Manager\Manager\Menu
          calls:
             - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]

#Bootstrap manager
#A class to manage the bootsrap classes
     AppBundle.bootstrapManager:
          class: AppBundle\Manager\Manager\Bootstrap
          calls:
             - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]


#Command manager
#A class to excecute Symfony console commands
     AppBundle.consoleCommandManager:
          class: AppBundle\Manager\Manager\ConsoleCommand
          calls:
             - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]



#Excel manager
#A class to export to Excel
     AppBundle.excelManager:
          class: AppBundle\Manager\Manager\Excel
          calls:
             - [setManagerParams, ['@doctrine.orm.entity_manager',
'@AppBundle.MonoLogManager', '@AppBundle.httpExceptionManager', '@service_container']]
?>
```

Users List templates

Here are the most important templates I use on page ‘users list’ to render the list of current users.

Userslist.html.twig

```
{% extends 'base.html.twig' %}
{% block body %}

    <p>
        <a href="/edituser/new" class="btn btn-success btn-sm" role="button" >
            Insert User
        </a>
    </p>

    <table class="{{ tableTypeClass }}">
        {% include 'frontEnd/partials/standard/tableHeader.html.twig'
            with {'headerItems': [ 'id' , 'Name' , 'Role', 'Email', ' ' , ' '] } only
        %}


        {% for user in list %}
            <tr>
                <td>{{ user.id }}</td>
                <td>{{ user.username }}</td>
                <td>{{ user.email }}</td>
                <td>{{ user.role }}</td>
                <td>
                    <a href="/edituser/{{ user.id }}"  class="btn btn-primary btn-sm"
role="button">
                                Edit
                    </a>
                </td>

                <td>
                    <a href="/deleteuser/{{ user.id }}"  class="btn btn-danger btn-sm"
role="button">
                        Delete
                    </a>
                </td>

            </tr>
        {% endfor %}
    </table>
{% endblock %}
```

Basetemplate.html.twig

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <title>{% block title %}Welcome!{% endblock %}</title>
        {% block stylesheets %}{% endblock %}
        <link rel="icon" type="image/x-icon" href="{{ asset('favicon.ico') }}" />
```

```
        <!-- external css -->
        <link rel="stylesheet" href="/js/jquery-ui-1.12.0/jquery-ui.min.css">
        <link rel="stylesheet" href="/bootstrap-4.0.0-alpha.3/dist/css/
bootstrap.min.css">

        <!-- css -->
        <link rel="stylesheet" href="/css/main.css">


        <!-- external Javascript -->
        <script src="/js/jquery-3.1.0.min.js"></script>
        <script src="/js/jquery-ui-1.12.0/jquery-ui.js"></script>
        <script src="/js/jquery-ui-1.12.0/jquery-ui.min.js"></script>
        <script src="https://www.atlasestateagents.co.uk/javascript/tether.min.js"></
script>
        <script src="/bootstrap-4.0.0-alpha.3/docs/dist/js/bootstrap.min.js"></script>

    </head>
    <body>


    {% include 'basePartials/menu.html.twig'   %}


    <div class="container theme-showcase" role="main">
        {% block body %}{% endblock %}
        {% block javascripts %}{% endblock %}
    </div>
    </body>
</html>
```

Tableheader.html.twig

```
{#
    A standard table header.
    As parameter you give an array of header items

    Example:

    {% include 'frontEnd/partials/standardTable/tableHeader.html.twig'
                with {'headerItems': [ 'id' , 'Name' , 'Role', 'Email', ' ' , ' '] } only
    %}

#}
<thead class="thead-inverse">
    {% for item in headerItems %}
        <th>
            <span class="tag">{{ item }}</span>
        </th>
```

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

Entity class voor de table ‘logs’

```
<?php
/**
 * Entity for table logs
 *
 */
namespace AppBundle\Entity;
use AppBundle\Entity\Entity;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;


/**
 * @ORM\Entity
 * @ORM\Table(name="logs")
 */
class Logs extends Entity
{
    /**
     * @ORM\Column(
     *      type="integer",
     *      name="logid"
     *  )
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $logid;

    /**
     * @ORM\Column(
     *      type="integer"
     *  )
     * @Assert\NotBlank(
     *      message = "'level' is required"
     *  )
     * @Assert\Range(
     *       min = 1,
     *       max = 3,
     *       minMessage = "Level must be 1,2 or 3 ( 1 = 'Error', 2 =  'Warn', 3 = 'Notice'
)",
     *       maxMessage = "Level must be 1,2 or 3 ( 1 = 'Error', 2 =  'Warn', 3 = 'Notice'
)"
     * )
     */
    protected $level;
```

```
    /**
     * @ORM\Column(
     *     type="integer"
     * )
     *  @Assert\NotBlank(
     *     message = "'type' is required"
     *  )
     *
     *
     * @Assert\Range(
     *      min = 1,
     *      max = 3,
     *      minMessage = "Type must be 1,2 or 3 ( 1 = 'Application', 2 =  'Security', 3 = 'System' )",
     *      maxMessage =  "Type must be 1,2 or 3 ( 1 = 'Application', 2 =  'Security', 3 = 'System' )"
     *
     * )
     */
    protected $type;

    /**
     * @ORM\Column(
     *     type="integer",
     *     name="client_systems_id"
     *  )
     *  @Assert\NotBlank(
     *     message = "'clientSystemsId' is required"
     *  )
     */
    protected $clientSystemsId;


    /**
     * @ORM\Column(type="text")
     * @Assert\NotBlank(
     *     message = "'message' is required"
     *  )
     */
    protected $message;

    /**
     * @ORM\Column(type="string", length=20)
     *
     */
    protected $timestamp;


}
```

Class to generate Symphony forms

Here is a class I have to generate symphony forms. The most important function is ‘public function get’ see it to get idea of this class.

```
<?php
/**
 * A generic class to create forms with Symfony classes
 *
 *
 */
namespace AppBundle\Manager\Manager;

use AppBundle\Manager\Manager;

use Doctrine\ORM\Tools\Pagination\Paginator;
use Symfony\Component\DependencyInjection\Tests\Compiler\C;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
use Symfony\Component\Form\Extension\Core\Type\HiddenType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\PasswordType;
use Symfony\Component\Form\Extension\Core\Type\CheckboxType;
use Symfony\Component\Form\Extension\Core\Type\FormType;
use Symfony\Component\Form\Forms;
use AppBundle\Manager\Manager\Bootstrap;
class Form extends Manager {

    /**
     * Text field
     *
```

```
     * @var Symfony\Component\Form\Extension\Core\Type\TextType
     */
    public static $TEXT;

    /**
     * Select field
     *
     * @var Symfony\Component\Form\Extension\Core\Type\ChoiceType
     */
    public static $SELECT;

     /**
     * Hidden field
     *
     * @var Symfony\Component\Form\Extension\Core\Type\HiddenType
     */
    public static $HIDDEN;

    /**
     * Password field
     *
     * @var Symfony\Component\Form\Extension\Core\Type\PasswordType
     */
    public static $PASSWORD;

    /**
     * Checkbox field
     *
     * @var Symfony\Component\Form\Extension\Core\Type\CheckboxType
     */
    public static $CHECKBOX;


    /**
     * Submit
     *
     * @var Symfony\Component\Form\Extension\Core\Type\HiddenType
     */
    public static $SUBMIT;


    /**
     * __construct
     */
    public function __construct() {
      self::setFormTypes();
   }


   /**
```

```
     * setFormTypes
     *
     */
    public static function setFormTypes() {
        self::$TEXT = TextType::class;
        self::$SELECT = ChoiceType::class;
        self::$HIDDEN = HiddenType::class;
        self::$PASSWORD = PasswordType::class;
        self::$CHECKBOX = CheckboxType::class;
        self::$SUBMIT = SubmitType::class;

    }


    /**
     *  Get form , get symofony form for given fiels
     *
     * Example of param $fields:
     *
     * [
     *   'startDate' => [
     *       'label' => 'Start Date',
     *       'key' => 'startDate',
     *       'type' => self::$TEXT,
     *       'value' => '2016-07-01',
     *   ],
     *
     *   'systemId' => [
     *       'label' => 'System',
     *       'key' => 'systemId',
     *       'type' => self::$SELECT,
     *       'value' => '2',
     *       'options' => [
     *           0 => [
     *               'value' => 1,
     *               'name' => 'Test 1',
     *           ],
    *           1 => [
     *               'value' => 2,
     *               'name' => 'Test 2',
     *           ],
     *   ],
     *
     *   'submit' => [
     *       'label' => 'Search',
     *       'key' => 'search',
     *       'default' => '0',
     *       'type' => self::$SUBMIT ,
     *       'value' => '',
     *   ],
     *
     * ]
```

```
    *
    *
    * @param array $fields   See above
    * @param array $options  Options array
    *                            'formName'      Form name
    *                            'action'        Current page is default
    *                            'method'        GET or POST. GET is default
    *                            'entityType'    One of the databse entites
    *                                            Default is 'FormType::class'
    *                                            Which is not bound to any table
    *
    *
    *
    *
    */
    public function get($fields, $options = []) {

        //
        // Set options as user defined or by default values
        //
        $defaultOptions = [
            'formName' => '',
            'action' => $this->getBaseLink(),
            'method' => 'GET',
            'entityType' =>  FormType::class,
            'id' => false
        ];

        foreach ($defaultOptions  as $key =>$default) {
            $$key = isset($options[$key]) ? $options[$key] : $defaultOptions[$key];
        }

        $formOptions = [];

        // set id
        if ($id) {
            $formOptions['attr'] = ['id' => $id];
        }

        //Initialize form
        $form = $this->intializeForm($formName, $entityType, $fields, $action, $method,
$formOptions);

        //
        // Set fields
        //
        foreach ($fields as $key => $field) {

            // Add standard input class for all fields
           $this->addStandardInputClassForField($field);

            $options =  $this->addFieldOptions($field);
```

```
            // Add select options
             if ($field['type'] == self::$SELECT) {
                $options['choices'] = $this->convertSelectOptions($field['options']);
             }

              // Add field
              $form->add($key, $field['type'], $options );
          };

         return  $form->getForm()->createView();
    }


    /**
     * intializeForm
     *
     *
     * @param string $formName
     * @param string $entityType
     * @param array $fields
     * @param string $action
     * @param string $method
     * @param array $formOptions
     * @return Symfony\Component\Form\FormBuilder
     */
   private function intializeForm($formName, $entityType, $fields, $action, $method,
$formOptions) {

        $formValues = array_column($fields, 'value', 'key');
        $form = $this->getContainer()
            ->get('form.factory')
            ->createNamedBuilder($formName, $entityType, $formValues,
                $formOptions );

        // set action and method
        $form->setAction($action);
        $form->setMethod($method);

        return $form;
   }


    /**
     * convertSelectOptions
     *
     * Example of $optionsArr:
     *
     * [
     *      0 => [
     *              'name' => 'Test 1',
     *              'value' => 1,
```

```
     *           ],
     *       1 => [
     *               'name' => 'Test 2',
     *               'value' => 2,
     *           ],
     * ]
     *
     *
     * @param $optionsArr  array of select options in format above
     *
     * @return array        Array of options in Symfony form format
     *
     */
    private function convertSelectOptions($optionsArr) {

        $choices = array_column($optionsArr, 'value' ,'name' );
        $firstItem = ['Select' => 0  ];
        $choices = array_merge($firstItem, $choices);
        return $choices;
    }


    private function addFieldOptions($field) {

        $options = [];

        // set user defined options
        $optionOptions = ['label' , 'attr'];
        foreach ($optionOptions as $option) {
            if (isset($field[$option])) {
                $options[$option] =  $field[$option];
            }
        }

        // ignore HTML 5 required option
        $fieldTypesToIgnore = [self::$CHECKBOX, self::$TEXT, self::$PASSWORD];
        if (in_array($field['type'], $fieldTypesToIgnore )) {
            $options['required'] = false;

        }

        return $options;
    }


    /**
     * This function adds a the standard input classes for all input fields.
     *
     * The meaning is that there is an entity. All are similar.
     *
     * @param array $field
     */
```

```
    private function addStandardInputClassForField(&$field) {
        $field['attr']['class'] = isset($field['attr']['class'])
            ? $field['attr']['class'] . Bootstrap::$FORM_FIELD_STANDARD
            : Bootstrap::$FORM_FIELD_STANDARD;
    }

}
<?php
/**
 * A generic class to create forms with Symfony classes
 *
 *
 */
namespace AppBundle\Manager\Manager;

use AppBundle\Manager\Manager;

use Doctrine\ORM\Tools\Pagination\Paginator;
use Symfony\Component\DependencyInjection\Tests\Compiler\C;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
use Symfony\Component\Form\Extension\Core\Type\HiddenType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\PasswordType;
use Symfony\Component\Form\Extension\Core\Type\CheckboxType;
use Symfony\Component\Form\Extension\Core\Type\FormType;
use Symfony\Component\Form\Forms;
use AppBundle\Manager\Manager\Bootstrap;
class Form extends Manager {

    /**
     * Text field
     *
     * @var Symfony\Component\Form\Extension\Core\Type\TextType
     */
    public static $TEXT;

    /**
     * Select field
     *
     * @var Symfony\Component\Form\Extension\Core\Type\ChoiceType
     */
    public static $SELECT;

    /**
     * Hidden field
     *
     * @var Symfony\Component\Form\Extension\Core\Type\HiddenType
     */
    public static $HIDDEN;
```

```
    /**
     * Password field
     *
     * @var Symfony\Component\Form\Extension\Core\Type\PasswordType
     */
    public static $PASSWORD;

    /**
     * Checkbox field
     *
     * @var Symfony\Component\Form\Extension\Core\Type\CheckboxType
     */
    public static $CHECKBOX;


    /**
     * Submit
     *
     * @var Symfony\Component\Form\Extension\Core\Type\HiddenType
     */
    public static $SUBMIT;


    /**
     * __construct
     */
    public function __construct() {
      self::setFormTypes();
   }


   /**
    * setFormTypes
    *
    */
   public static function setFormTypes() {
       self::$TEXT = TextType::class;
       self::$SELECT = ChoiceType::class;
       self::$HIDDEN = HiddenType::class;
       self::$PASSWORD = PasswordType::class;
       self::$CHECKBOX = CheckboxType::class;
       self::$SUBMIT = SubmitType::class;

   }


   /**
    *  Get form , get symofony form for given fiels
    *
    * Example of param $fields:
```

```
     *
     * [
     *   'startDate' => [
     *       'label' => 'Start Date',
     *       'key' => 'startDate',
     *       'type' => self::$TEXT,
     *       'value' => '2016-07-01',
     *   ],
     *
     *   'systemId' => [
     *       'label' => 'System',
     *       'key' => 'systemId',
     *       'type' => self::$SELECT,
     *       'value' => '2',
     *       'options' => [
     *           0 => [
     *               'value' => 1,
     *               'name' => 'Test 1',
     *           ],
    *            1 => [
     *               'value' => 2,
     *               'name' => 'Test 2',
     *           ],
     *   ],
     *
     *   'submit' => [
     *       'label' => 'Search',
     *       'key' => 'search',
     *       'default' => '0',
     *       'type' => self::$SUBMIT ,
     *       'value' => '',
     *   ],
     *
     * ]
     *
     *
     * @param array $fields   See above
     * @param array $options  Options array
     *                           'formName'      Form name
     *                           'action'        Current page is default
     *                           'method'        GET or POST. GET is default
     *                           'entityType'    One of the databse entites
     *                                           Default is 'FormType::class'
     *                                           Which is not bound to any table
     *
     *
     *
     *
     */
    public function get($fields, $options = []) {

        //
```

```
        // Set options as user defined or by default values
        //
        $defaultOptions = [
            'formName' => '',
            'action' => $this->getBaseLink(),
            'method' => 'GET',
            'entityType' =>  FormType::class,
            'id' => false
        ];

        foreach ($defaultOptions  as $key =>$default) {
            $$key = isset($options[$key]) ? $options[$key] : $defaultOptions[$key];
        }

        $formOptions = [];

        // set id
        if ($id) {
            $formOptions['attr'] = ['id' => $id];
        }

        //Initialize form
        $form = $this->intializeForm($formName, $entityType, $fields, $action, $method,
$formOptions);

        //
        // Set fields
        //
        foreach ($fields as $key => $field) {

            // Add standard input class for all fields
           $this->addStandardInputClassForField($field);

            $options =  $this->addFieldOptions($field);

           // Add select options
            if ($field['type'] == self::$SELECT) {
               $options['choices'] = $this->convertSelectOptions($field['options']);
            }

             // Add field
             $form->add($key, $field['type'], $options );
         };

        return  $form->getForm()->createView();
    }


    /**
     * intializeForm
     *
     *
```

```
     * @param string $formName
     * @param string $entityType
     * @param array $fields
     * @param string $action
     * @param string $method
     * @param array $formOptions
     * @return Symfony\Component\Form\FormBuilder
     */
    private function intializeForm($formName, $entityType, $fields, $action, $method,
$formOptions) {

        $formValues = array_column($fields, 'value', 'key');
        $form = $this->getContainer()
            ->get('form.factory')
            ->createNamedBuilder($formName, $entityType, $formValues,
                $formOptions );

        // set action and method
        $form->setAction($action);
        $form->setMethod($method);

        return $form;
    }


    /**
     * convertSelectOptions
     *
     * Example of $optionsArr:
     *
     * [
     *      0 => [
     *              'name' => 'Test 1',
     *              'value' => 1,
     *          ],
     *      1 => [
     *              'name' => 'Test 2',
     *              'value' => 2,
     *          ],
     * ]
     *
     *
     * @param $optionsArr  array of select options in format above
     *
     * @return array        Array of options in Symfony form format
     *
     */
    private function convertSelectOptions($optionsArr) {

        $choices = array_column($optionsArr, 'value' ,'name' );
        $firstItem = ['Select' => 0  ];
        $choices = array_merge($firstItem, $choices);
```

```
        return $choices;
    }


    private function addFieldOptions($field) {

        $options = [];

        // set user defined options
        $optionOptions = ['label' , 'attr'];
        foreach ($optionOptions as $option) {
            if (isset($field[$option])) {
                $options[$option] =  $field[$option];
            }
        }

        // ignore HTML 5 required option
        $fieldTypesToIgnore = [self::$CHECKBOX, self::$TEXT, self::$PASSWORD];
        if (in_array($field['type'], $fieldTypesToIgnore )) {
            $options['required'] = false;

        }

        return $options;
    }


    /**
     * This function adds a the standard input classes for all input fields.
     *
     * The meaning is that there is an entity. All are similar.
     *
     * @param array $field
     */
    private function addStandardInputClassForField(&$field) {
        $field['attr']['class'] = isset($field['attr']['class'])
            ? $field['attr']['class'] . Bootstrap::$FORM_FIELD_STANDARD
            : Bootstrap::$FORM_FIELD_STANDARD;
    }


}
```