Introducción

El componente de Workflow proporciona utilidades para gestionar un flujo de trabajo o una máquina de estados.

Para instalar el componente de Workflow con Flex, hay que utilizar el siguiente comando:

> *composer require symfony/workflow*

Creación de un Workflow

Un Workflow es un proceso o un ciclo de vida a través del cual se mueven nuestros objetos. Cada paso, fase, estado, etc en el proceso se denomina **place**. Los objetos cambian de un *place* a otro mediante **transitions**.

Para definir un Workflow por tanto, es necesario definir:

- **places**
- **transitions**

Veamos un ejemplo de configuración de un workflow:

```
# config/packages/workflow.yaml
framework:
    workflows:
      publicacion_articulo:
            type: 'workflow' # o 'state_machine'
            marking_store:
                type: 'multiple_state' # o 'single_state'
                arguments:
                    - 'estado'
            supports:
                - App\Entity\Articulo
            places:
                - borrador
```

```
                - pendiente_de_correccion
                - pendiente_de_aprobacion
                - corregido
                - aprobado
                - rechazado
                - publicado
            initial_place: borrador
            transitions:
                revisar:
                    from: borrador
                    to:   [pendiente_de_correccion, pendie
nte_de_aprobacion]
                corregido:
                    from: 'pendiente_de_correccion'
                    to:   corregido
                aprobar:
                    from: 'pendiente_de_aprobacion'
                    to:   aprobado
                publicar:
                    from: [corregido, aprobado]
                    to:   publicado
                rechazar:
                    from: 'pendiente_de_aprobacion'
                    to:   rechazado
```

Este Workflow actuará sobre entidades *App|Entity|Articulo* y utilizará la propiedad *estado* para almacenar el *place* o los *places* de la entidad.

Si **marking_store** está definida como **multiple_state**, la entidad podrá estar en más de un place simultáneamente.

Si **marking_store** está definida como **single_state**, la entidad solamente podrá estar en un place simultáneamente.

Un ejemplo de entidad Articulo podría ser el siguiente:

```
/**
 * Articulo
 *
 * @ORM\Table(name="articulo")}
 * @ORM\Entity
 */
class Articulo
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer", nullable=fal
se)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $id;

    /**
```

```
     * @var string
     *
     * @ORM\Column(name="titulo", type="string", length=25
5, nullable=false)
     */
    private $titulo;

    /**
     * @var string
     *
     * @ORM\Column(name="contenido", type="string", length
=255, nullable=false)
     */
    private $contenido;

    /**
     * Esta es la propiedad utilizada por el marking_store
     *
     * @var array
     *
     * @ORM\Column(type="json_array", nullable=true)
     */
    private $estado;
```

Si el workflow es single_state o state_machine, entonces la propiedad estado se definiría como string:

```
/**
 * Articulo
 *
 * @ORM\Table(name="articulo")}
 * @ORM\Entity
 */
class Articulo
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer", nullable=fal
se)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $id;


    /**
     * @var string
     *
     * @ORM\Column(name="titulo", type="string", length=25
5, nullable=false)
     */
    private $titulo;


    /**
```

```
     * @var string
     *
     * @ORM\Column(name="contenido", type="string", length
=255, nullable=false)
     */
    private $contenido;

    /**
     * Esta es la propiedad utilizada por el marking_store
     *
     * @var string
     *
     * @ORM\Column(type="string", nullable=true)
     */
    private $estado;
```

Cómo trabajar con Workflows

Un workflow tiene varios métodos para facilitarnos trabajar con él. Inyectando el servicio correspondiente, podemos obtener el workflow asociado a un objeto:

```
public function edit(Registry $workflows) {
    $workflow = $workflows->get($articulo);
}
```

Una vez tenemos el workflow, podemos utilizar sus métodos

- can($objeto, $transicion)

Devuelve *true* si se puede realizar la transición sobre el objeto.

- apply($objeto, $transicion)

Aplica una transición a un objeto

- getEnabledTransitions($objeto);

Devuelve un array con las posibles transiciones según el *place* actual del objeto.

Veamos un ejemplo:

```
use Symfony\Component\Workflow\Registry;
```

```php
use App\Entity\Articulo;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\Workflow\Exception\LogicException;

//...

/**
 * @Route("articulo/{id}/revisar", name="articulo_revisar"
)
 */
public function revisar(Registry $workflows, Articulo $art
iculo)
{
    $workflow = $workflows->get($articulo);

    if($workflow->can($articulo, 'revisar')) {
        try {
            $workflow->apply($articulo, 'revisar');
            $this->getDoctrine()->getManager()->flush();
        } catch (LogicException $exception) {
            // ... Si se intenta hacer una transición no v
álida
        }
    }

    return $this->redirectToRoute('articulo_index');
}
```

Funciones de Twig

El componente Workflow define varias funciones de Twig para facilitar la programación de las plantillas:

- **workflow_can(objeto, transicion)**
  Devuelve *true* si el objeto puede realizar la transición.

- **workflow_transitions(objeto)**
  Devuelve un array con todas las transiciones posibles de un objeto según su estado actual.

- **workflow_marked_places(objeto)**
  Devuelve un array con los nombres de los estados actuales de un objeto.

- **workflow_has_marked_place(objeto, estado)**
  Devuelve *true* si el objeto tiene el estado.

En el siguiente bloque de código podemos ver algunos ejemplos de uso:

```
<h3>Actions</h3>
{% if workflow_can(post, 'publish') %}
    <a href="...">Publish article</a>
{% endif %}
{% if workflow_can(post, 'to_review') %}
    <a href="...">Submit to review</a>
```

```
{% endif %}
{% if workflow_can(post, 'reject') %}
    <a href="...">Reject article</a>
{% endif %}

{# Recorrer las transiciones posibles de un objeto #}
{% for transition in workflow_transitions(post) %}
    <a href="...">{{ transition.name }}</a>
{% else %}
    No actions available.
{% endfor %}

{# Comprobar si un objeto está en un determinado estado #}
{% if workflow_has_marked_place(post, 'to_review') %}
    <p>This post is ready for review.</p>
{% endif %}

{# Obtener todos los estados de un objeto #}
{% if 'waiting_some_approval' in workflow_marked_places(po
st) %}
    <span class="label">PENDING</span>
{% endif %}
```

Eventos

Para hacer nuestros workflows más flexibles y potentes, disponen de muchos eventos sobre los que actuar.

Cada paso en la transición de un estado a otro lanza 3 eventos:

- Un evento genérico para todos los workflows;
- Un evento para el workflow concreto;
- Un evento para el workflow concreto y la transición o estado concernientes.

Los eventos que se generan cada vez que se inicia una transición de estado son los siguientes, en este orden:

- **workflow.guard**
  Valida si una transición es válida.

Los 3 eventos notificados son:

- workflow.guard
- workflow.[workflow name].guard
- workflow.[workflow name].guard.[transition name]

- **workflow.leave**
  El *subject* (objeto) está a punto de salir de un *place*.

Los 3 eventos notificados son:

workflow.leave
workflow.[workflow name].leave
workflow.[workflow name].leave.[place name]

- **workflow.transition**

El objeto va a realizar una transición.

Los 3 eventos notificados son:

- workflow.transition
- workflow.[workflow name].transition
- workflow.[workflow name].transition.[transition name]

- **workflow.enter**

El objeto está a punto de entrar en un *place*. El place del objeto todavía no está actualizado.

Los 3 eventos notificados son:

- workflow.enter
- workflow.[workflow name].enter
- workflow.[workflow name].enter.[place name]

- **workflow.entered**

El objeto ha entrado en uno o más *places*. El objeto ya está actualizado. (Aquí podría ser un buen sitio para hacer flush de Doctrine).

Los 3 eventos notificados son:

- workflow.entered
- workflow.[workflow name].entered
- workflow.[workflow name].entered.[place name]
- **workflow.completed**

El objeto ha completado la transición.

Los 3 eventos notificados son:

- workflow.completed
- workflow.[workflow name].completed
- workflow.[workflow name].completed.[transition name]
- **workflow.announce**

Ino por cada transición que sea ahora accesible para el objeto.

Los 3 eventos notificados son:

- workflow.announce
- workflow.[workflow name].announce
- workflow.[workflow name].announce.[transition name]

NOTA

Los eventos se notifican al Dispatcher aunque la transición no haga cambiar el *place*.

Ejemplo

Este ejemplo registra en el log los cambios de *place* de un objeto.

```
use Psr\Log\LoggerInterface;
use Symfony\Component\EventDispatcher\EventSubscriberInter
face;
use Symfony\Component\Workflow\Event\Event;

class WorkflowLogger implements EventSubscriberInterface
{
    public function __construct(LoggerInterface $logger)
    {
        $this->logger = $logger;
    }

    public function onLeave(Event $event)
    {
        $this->logger->alert(sprintf(
            'Blog post (id: "%s") performed transaction "%
s" from "%s" to "%s"',
            $event->getSubject()->getId(),
            $event->getTransition()->getName(),
            implode(', ', array_keys($event->getMarking()-
>getPlaces()))),
```

```
            implode(', ', $event->getTransition()->getTos(
))
        ));
    }

    public static function getSubscribedEvents()
    {
        return array(
            'workflow.blog_publishing.leave' => 'onLeave',
        );
    }
}
```

Guard Events

Los eventos **workflow.guard** se notifican cada vez que se llama a cualquiera de los métodos **Workflow::can**, **Workflow::apply** o **Workflow::getEnabledTransitions**.

Con estos eventos se puede añadir programación personalizada para decidir qué transiciones son válidas o no.

En el siguiente ejemplo, se añade un evento de tipo Guard para evitar que los posts que no tengan título puedan cambiar al estado 'to_review'.

```
use Symfony\Component\Workflow\Event\GuardEvent;
use Symfony\Component\EventDispatcher\EventSubscriberInter
```

```
face;

class BlogPostReviewListener implements EventSubscriberInt
erface
{
    public function guardReview(GuardEvent $event)
    {
        /** @var \App\Entity\BlogPost $post */
        $post = $event->getSubject();
        $title = $post->title;

        if (empty($title)) {
            // Posts with no title should not be allowed
            $event->setBlocked(true);
        }
    }

    public static function getSubscribedEvents()
    {
        return array(
            'workflow.blogpost.guard.to_review' => array('
guardReview'),
        );
    }
}
```

Métodos del objeto Event

Cada evento del Workflow es una instancia de **Symfony\Component\Workflow\Event\Event**. Esto significa que cada evento tiene acceso a la siguiente información:

- getMarking()

  Devuelve el **marking** del workflow.

- getSubject()

  Devuelve el objeto/entidad sobre la que se está trabajando.

- getTransition()

  Devuelve la transición que se está realizando.

- getWorkflowName()

  Devuelve un string con el nombre del workflow.

Métodos del objeto GuardEvent

Esta clase extiende de la anterior y tiene dos métodos más.

- isBlocked()

  Devuelve true si la transición está bloqueada.

- setBlocked()

  Bloquea la transición.

Métodos del objeto TransitionEvent

Esta clase extiende de la primera y tiene dos métodos más.

- setNextState($state)

  Establece el siguiente estado del objeto.

- getNextState()

  Obtiene el nombre del siguiente estado.

Cómo depurar un Workflow

El componente de Workflow viene con un comando que genera una representación de los workflows en formato dot

> *bin/console workflow:dump name | dot -Tpng -o graph.png*

El comando dot es forma parte de Graphviz y convierte ficheros en formato dot en ficheros png.

Se puede descargar de Graphviz.org.