Actividad

Symfony. Sesiones

1. Sesiones

El componente Symfony **HttpFoundation** tiene **un subsistema de sesión muy potente y flexible** que está diseñado para proporcionar una gestión de sesiones que puede utilizar para almacenar información sobre el usuario entre solicitudes a través de una interfaz clara orientada a objetos utilizando una variedad de controladores de almacenamiento de sesión.

Las sesiones de Symfony están diseñadas para reemplazar el uso de las funciones PHP súper globales y nativas de $_SESSION relacionadas con la manipulación de la sesión, como session_start(), session_regenerate_id(), session_id(), session_name() y session_destroy().

IMPORTANTE: Las sesiones solo se inician si lee o escribe desde ellas.

2. Instalación

Debe instalar el componente HttpFoundation para controlar las sesiones:

```
$ composer requiere symfony/http-foundation
```

1. Uso básico de la sesión

La sesión está disponible a través del objeto Request y el servicio RequestStack. Symfony inyecta el servicio request_stack **en servicios y controladores**

Fíjate en el siguiente ejemplo como se obtiene el objeto sesión en un controlador

```
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

public function index(Request $request): Response
{
    $session = $request->getSession();

    // ...
}
```

2. Atributos de sesión

La gestión de sesiones de PHP requiere el uso de la super-global de $_SESSION. Sin embargo. Para ayudar a superar esto, Symfony utiliza *bolsas de sesión* vinculadas a la sesión para encapsular un conjunto de datos específico de **atributos**.

Este enfoque mitiga el poder corromper el espacio de nombres dentro de superglobal de $_SESSION porque cada bolsa almacene todos sus datos en un espacio de nombres único. Esto permite a Symfony coexistir pacíficamente con otras aplicaciones o bibliotecas que podrían usar el superglobal de $_SESSION y todos los datos siendo completamente compatibles con la gestión de sesiones de Symfony.

Un contenedor de sesión es un objeto PHP que actúa como una matriz. Fíjate como se realizan las operaciones comunes con una sesión en el siguiente código

```
// stores an attribute for reuse during a later user request
$session->set('attribute-name', 'attribute-value');

// gets an attribute by name
$foo = $session->get('foo');

// the second argument is the value returned when the attribute doesn't exist
$filters = $session->get('filters', []);
```

Los atributos almacenados permanecen en la sesión durante el resto de la sesión de ese usuario. De forma predeterminada, los atributos de sesión son pares clave-valor administrados con la `clase AttributeBag`.

IMPORTANTE :Las sesiones se inician automáticamente cada vez que lee, escribe o incluso compruebas la existencia de datos en la sesión. Esto puede perjudicar el rendimiento de la aplicación, ya que todos los usuarios recibirán una cookie de sesión. Para evitar el inicio de sesiones para usuarios anónimos, debes evitar el acceso a la sesión en el código.

3. Mensajes Flashs en la sesion

Puede almacenar mensajes especiales, denominados mensajes "flash", en la sesión del usuario. Por diseño, **los mensajes flash están pensados para ser utilizados exactamente una vez: desaparecen de la sesión automáticamente tan pronto como los recuperas.** Esta característica hace que los mensajes "flash" sean particularmente buenos para almacenar notificaciones de usuario.

Por ejemplo, imagina que estás procesando el envío de un formulario:

```
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
// ...

public function update(Request $request): Response
{
    // ...

    if ($form->isSubmitted() && $form->isValid()) {
```

```
        // do some sort of processing

        $this->addFlash(
            'notice',
            'Your changes were saved!'
        );
        // $this->addFlash() is equivalent to $request->getSession()->getFlashBag()->add()

        return $this->redirectToRoute(/* ... */);
    }

    return $this->render(/* ... */);
}
```

Después de procesar la solicitud, el controlador establece un mensaje flash en la sesión y, a continuación, hace la redirección. La clave de mensaje (fíjate en este ejemplo) puede ser cualquier cosa: usarás esta clave para recuperar el mensaje.

En la plantilla de la página siguiente (o mejor aún, en la plantilla de diseño base), lee los mensajes flash de la sesión utilizando el método flashes() proporcionado por la variable global de la aplicación Twig:

```
{# templates/base.html.twig #}

{# read and display just one flash message type #}
{% for message in app.flashes('notice') %}
    <div class="flash-notice">
        {{ message }}
    </div>
{% endfor %}

{# read and display several types of flash messages #}
{% for label, messages in app.flashes(['success', 'warning']) %}
    {% for message in messages %}
        <div class="flash-{{ label }}">
            {{ message }}
        </div>
    {% endfor %}
{% endfor %}

{# read and display all flash messages #}
{% for label, messages in app.flashes %}
    {% for message in messages %}
        <div class="flash-{{ label }}">
            {{ message }}
        </div>
    {% endfor %}
{% endfor %}
```

Es común usar un aviso, advertencia o error como claves de los diferentes tipos de mensajes, pero puedes usar cualquier clave que se ajuste a sus necesidades.

4. Configuración de la sesión

En el framework Symfony, las sesiones están habilitadas de forma predeterminada. El almacenamiento de sesiones y otras configuraciones se pueden controlar en la configuración framework.session en config/packages/framework.yaml:

```
# config/packages/framework.yaml
framework:
  # Enables session support. Note that the session will ONLY be started if you read or write from it.
  # Remove or comment this section to explicitly disable session support.
  session:
    # ID of the service used for session storage
    # NULL means that Symfony uses PHP default session mechanism
    handler_id: null
    # improves the security of the cookies used for sessions
    cookie_secure: auto
    cookie_samesite: lax
    storage_factory_id: session.storage.factory.native
```

Establecer la opción de configuración handler_id en null significa que Symfony utilizará el mecanismo de sesión nativo de PHP. Los archivos de metadatos de sesión se almacenarán fuera de la aplicación Symfony, en un directorio controlado por PHP. Aunque esto suele simplificar las cosas, es posible que algunas opciones relacionadas con la expiración de la sesión no funcionen según lo esperado si otras aplicaciones que escriben en el mismo directorio tienen una configuración de duración máxima más corta.

IMPORTANTE: Las sesiones de Symfony son incompatibles con la directiva php.ini session.auto_start = 1. Esta directiva debe estar desactivada en php.ini, en las directivas del servidor web o en .htaccess.

5. Tiempo de inactividad/actividad

A menudo hay circunstancias en las que es posible que desee proteger o minimizar el uso no autorizado de una sesión cuando un usuario se aleja de su terminal mientras está conectado destruyendo la sesión después de un cierto período de tiempo de inactividad. Por ejemplo, es común que las aplicaciones

bancarias cierren la sesión del usuario después de solo 5 a 10 minutos de inactividad. Establecer la duración de la cookie aquí no es apropiado porque puede ser manipulado por el cliente, por lo que debemos hacer la caducidad en el lado del servidor. La forma más fácil es implementar esto a través de la recolección de elementos no utilizados de sesión, que se ejecuta con una frecuencia razonable. El cookie_lifetime se establecería en un valor relativamente alto y el gc_maxlifetime de recolección de elementos no utilizados se establecería para destruir sesiones en el período de inactividad deseado.

La otra opción es comprobar específicamente si una sesión ha caducado después de iniciarla. La sesión se puede destruir según sea necesario. Este método de procesamiento puede permitir que la caducidad de las sesiones se integre en la experiencia del usuario, por ejemplo, mediante la visualización de un mensaje.

Symfony registra algunos metadatos sobre cada sesión para darte un control preciso sobre la configuración de seguridad:

```
$session->getMetadataBag()->getCreated();
$session->getMetadataBag()->getLastUsed();
```

Ambos métodos devuelven una marca de tiempo Unix (relativa al servidor).

Estos metadatos se pueden usar para caducar explícitamente una sesión en el acceso:

```
$session->start();
if (time() - $session->getMetadataBag()->getLastUsed() > $maxIdleTime) {
    $session->invalidate();
    throw new SessionExpired(); // redirect to expired session page
}
```

También es posible saber en el valor de cookie_lifetime para una cookie en particular leyendo el método getLifetime():

```
$session->getMetadataBag()->getLifetime();
```

El tiempo de caducidad de la cookie se puede determinar agregando la marca de tiempo creada y la vida útil.

6. Almacenar sesiones en una base de datos

Symfony almacena las sesiones en archivos de forma predeterminada. **Si la aplicación es atendida por varios servidores, debes usar una base de datos para que las sesiones funcionen en diferentes servidores.**

Symfony puede almacenar sesiones en todo tipo de bases de datos (relacionales, NoSQL y clave-valor), pero recomienda bases de datos clave-valor como Redis para obtener el mejor rendimiento.

IMPORTANTE: Nosotros usaremos el almacenamiento en archivos locales del servidor