Notifications

Event-driven notifications via email, in-app messages, and webhooks.

Anonymous · 09/09/2026

Notifications

The Notifications module delivers notifications to users and organizations through multiple channels — email, in-app messages, and outbound webhooks. It is entirely event-driven: apps define which events generate notifications, and users control how they receive them.

Architecture overview

```
App event (Symfony EventDispatcher)
    └─► NotificationDispatcher
            ├─ Resolves recipients
            ├─ Checks user preferences
            └─ Routes to channels:
                    ├─► EmailChannel       → Symfony Mailer
                    ├─► InAppChannel       → InAppNotification entity
                    └─► WebhookChannel     → HTTP POST (async via Messenger)
```

Defining notification events

Implement `EventDefinitionProviderInterface` to register the events your module can generate:

```
use
Derafu\PlatformBundle\Notifications\Contract\Service\EventDefinitionProviderInterface;
use Derafu\PlatformBundle\Notifications\ValueObject\EventDefinition;

final class ProjectEventDefinitions implements EventDefinitionProviderInterface
{
    public function getEvents(): array
    {
        return [
            new EventDefinition(
                code: 'project.created',
                name: 'Project created',
                description: 'A new project was created in your organization.',
                channels: ['email', 'inapp', 'webhook'],
            ),
```

```
            new EventDefinition(
                code: 'project.updated',
                name: 'Project updated',
                description: 'A project you are a member of was updated.',
                channels: ['inapp', 'webhook'],
            ),
        ];
    }
}
```

The service is auto-discovered via the `EventDefinitionProviderInterface` `AutoconfigureTag`. Events automatically appear in the user preferences UI and as available webhook triggers.

Dispatching a notification

When something happens in your app, dispatch a notification via the `NotificationDispatcherInterface`:

```
use
Derafu\PlatformBundle\Notifications\Contract\Service\NotificationDispatcherInterface;
use Derafu\PlatformBundle\Notifications\ValueObject\NotificationEvent;

final class ProjectService
{
    public function __construct(
        private readonly NotificationDispatcherInterface $dispatcher,
    ) {}

    public function create(Project $project, UserInterface $creator): void
    {
        // ... create project logic ...

        $this->dispatcher->dispatch(new NotificationEvent(
            code: 'project.created',
            subject: $project,
            actor: $creator,
            data: [
                'project_name' => $project->getName(),
                'project_url'  => '/projects/' . $project->getId(),
            ],
        ));
    }
}
```

The dispatcher resolves recipients, checks each user's preferences, and routes to the appropriate channels.

Recipient resolvers

The dispatcher does not know who should receive a notification — that is your domain knowledge. Implement `NotificationRecipientResolverInterface` to define the recipient logic for each event:

```
use
Derafu\PlatformBundle\Notifications\Contract\Service\NotificationRecipientResolverInte
rface;
use Derafu\PlatformBundle\Notifications\ValueObject\NotificationEvent;

final class ProjectNotificationResolver implements
NotificationRecipientResolverInterface
{
    public function supports(NotificationEvent $event): bool
    {
        return str_starts_with($event->code, 'project.');
    }

    public function resolve(NotificationEvent $event): array
    {
        $project = $event->subject; // the Project entity

        // Return all org members who have access to this project
        return $project->getOrganization()->getMembers()->map(fn($m) =>
$m->getUser())->toArray();
    }
}
```

The service is auto-discovered via the `NotificationRecipientResolverInterface` `AutoconfigureTag`. Multiple resolvers can be registered — the dispatcher collects all unique recipients across resolvers that support the event.

Delivery channels

Email

Notifications are rendered using Twig templates and sent via Symfony Mailer. The bundle looks for templates at:

```
templates/notifications/email/{event_code}.html.twig
```

Create a template for each event that should generate an email:

```
{# templates/notifications/email/project.created.html.twig #}
{% extends 'emails/base.html.twig' %}
```

```
{% block subject %}New project: {{ data.project_name }}{% endblock %}

{% block body %}
<p>A new project <strong>{{ data.project_name }}</strong> was created.</p>
<a href="{{ data.project_url }}">View project</a>
{% endblock %}
```

In-app

In-app notifications are persisted as `InAppNotification` entities and displayed in your UI. The bundle provides a Twig helper to fetch and render them. Mark as read via:

```
$inAppNotificationRepository->markAsRead($notification);
$inAppNotificationRepository->markAllAsReadForUser($user);
```

Webhooks

Users and organizations configure webhook endpoints in the settings UI. When a notification is dispatched, the bundle:

1. Finds active webhook endpoints subscribed to the event code.
2. Renders the payload as JSON.
3. Dispatches a Messenger message for async delivery.
4. Signs the payload with `X-Webhook-Signature: sha256=<hmac>` using the endpoint's secret.
5. Records the delivery attempt in `WebhookDelivery`.

Signature verification (on the receiving end)

```
$signature = hash_hmac('sha256', $rawBody, $secret);
$expected = 'sha256=' . $signature;
hash_equals($expected, $request->headers->get('X-Webhook-Signature'));
```

User preferences

Users control which channels they receive for each event from the settings UI. The preferences are stored in `UserNotificationPreference` entities.

The dispatcher automatically respects preferences — if a user has disabled email notifications for `project.created`, the email channel is skipped for that user even if the email resolver resolves them as a recipient.

Async delivery via Messenger

Webhook delivery happens asynchronously via Symfony Messenger to avoid blocking the request. Configure the routing:

```
# config/packages/messenger.yaml
framework:
    messenger:
        routing:
            'Derafu\PlatformBundle\Notifications\Messenger\DeliverWebhookMessage':
async
```

Webhook delivery retention

Delivery logs are cleaned up automatically. Configure the retention policy:

```
derafu_platform:
    notifications:
        webhook_delivery_retention:
            max_age_days:     30   # delete logs older than 30 days
            max_per_endpoint: 500  # keep at most 500 logs per endpoint
```

Run the cleanup command periodically (daily cron recommended):

```
bin/console notifications:webhook-deliveries:cleanup
```

Configuration

```
derafu_platform:
    notifications:
        from_email: '%env(MAILER_FROM_EMAIL)%'
        from_name:  '%env(MAILER_FROM_NAME)%'

        inapp_notification_class:              App\Entity\Notif\InAppNotification
        user_webhook_endpoint_class:           App\Entity\Notif\UserWebhookEndpoint
        organization_webhook_endpoint_class:
App\Entity\Notif\OrganizationWebhookEndpoint
        user_webhook_delivery_class:           App\Entity\Notif\UserWebhookDelivery
        organization_webhook_delivery_class:
App\Entity\Notif\OrganizationWebhookDelivery
        user_notification_preference_class:
App\Entity\Notif\UserNotificationPreference
        user_digest_preference_class:          App\Entity\Notif\UserDigestPreference
```

Required entities

Create concrete entities extending the bundle's base classes:

```
// src/Entity/Notif/InAppNotification.php
namespace App\Entity\Notif;

use Derafu\PlatformBundle\Notifications\Entity\BaseInAppNotification;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity]
#[ORM\Table(name: 'notif_inapp')]
class InAppNotification extends BaseInAppNotification {}
```

Repeat for: `UserWebhookEndpoint`, `OrganizationWebhookEndpoint`, `UserWebhookDelivery`, `OrganizationWebhookDelivery`, `UserNotificationPreference`, `UserDigestPreference`.

See Installation for the full entity table.

Last updated on 09/09/2026