Multi-tenancy & RLS

Row-level security with automatic Doctrine filtering by tenant.

Anonymous · 09/09/2026

Multi-tenancy & row-level security

The bundle provides automatic row-level filtering for entities that belong to a tenant — an Organization, a User, or (with a small amount of app code) any custom resource.

How it works

Three pieces work together:

1. **Tenant contexts** — services that hold "who/what is the current tenant for this request". The bundle provides `OrganizationContext` and `UserContext`; the app can create its own.
2. **Integration interfaces** — `OrganizationOwnedInterface` and `UserOwnedInterface` mark which entities get filtered.
3. **`IdentityTenantFilter`** — a Doctrine SQLFilter that appends `WHERE organization_id = :id` and/or `WHERE owner_id = :id` to every query on entities that implement those interfaces.

A `TenantFilterListener` enables the filter automatically on each request when at least one context has a tenant set.

Built-in: Organization as tenant

```
// App middleware (kernel.request listener, priority >= 0):
$orgId = $request->headers->get('X-Organization-Id');
$org = $orgRepository->find($orgId);
$this->organizationContext->setCurrent($org);

// All queries on OrganizationOwnedInterface entities now auto-filter:
$contribuyentes = $contribuyenteRepository->findAll();
// SQL: SELECT ... FROM contribuyentes WHERE organization_id = 42
```

No changes to repositories, no manual WHERE clauses.

Built-in: User as tenant

```
// App middleware:
$this->userContext->setCurrent($this->getUser());

// All queries on UserOwnedInterface entities now auto-filter:
$notes = $noteRepository->findAll();
// SQL: SELECT ... FROM personal_notes WHERE owner_id = 7
```

Both at once

An entity can implement both interfaces:

```
class Contribuyente implements OrganizationOwnedInterface, UserOwnedInterface
{
    use OrganizationOwnedTrait;
    use UserOwnedTrait;
}
```

When both contexts are active, both conditions apply (AND):

```
WHERE organization_id = 42 AND owner_id = 7
```

Timing

Set the context in a `kernel.request` listener at priority **0 or higher**. The `TenantFilterListener` runs at priority **-10** — after your context is set but before most controller logic.

Setting the context in a controller is too late: queries during authentication or in other listeners would not be filtered.

Disabling the filter

The filter only activates when at least one context has a value. For admin panels, public endpoints, or CLI commands where you want unfiltered access, simply don’t set any context.

To explicitly disable mid-request:

```
$this->entityManager->getFilters()->disable('identity_tenant');
```

Custom tenant: resource-level (e.g. Contribuyente)

The bundle covers Organization and User as tenants. For resource-level tenancy (e.g. all queries scoped to a specific Contribuyente), follow the same pattern:

1. Create a context interface + implementation

```
// src/Service/ContribuyenteContextInterface.php
interface ContribuyenteContextInterface extends TenantContextInterface
{
    public function getCurrent(): ?Contribuyente;
    public function setCurrent(?Contribuyente $contribuyente): void;
}

// src/Service/InMemoryContribuyenteContext.php
final class InMemoryContribuyenteContext implements ContribuyenteContextInterface
{
    private ?Contribuyente $current = null;

    public function getCurrent(): ?Contribuyente { return $this->current; }
    public function setCurrent(?Contribuyente $c): void { $this->current = $c; }
    public function getTenantId(): ?int { return $this->current?->getId(); }
    public function getTenantEntity(): ?object { return $this->current; }
}
```

2. Create a marker interface for filterable entities

```
interface ContribuyenteOwnedInterface
{
    public function getContribuyente(): ?Contribuyente;
}
```

3. Create a Doctrine SQLFilter

```
use Doctrine\ORM\Query\Filter\SQLFilter;

class ContribuyenteTenantFilter extends SQLFilter
{
    public function addFilterConstraint(ClassMetadata $entity, string $alias): string
    {
        if
($entity->getReflectionClass()?->implementsInterface(ContribuyenteOwnedInterface::clas
s)) {
            try {
                return sprintf('%s.contribuyente_id = %s', $alias,
$this->getParameter('contribuyente_id'));
            } catch (\InvalidArgumentException) {}
```

```
        }
        return '';
    }
}
```

4. Register and activate

```
# config/packages/doctrine.yaml
doctrine:
    orm:
        filters:
            contribuyente_tenant:
                class: App\Doctrine\Filter\ContribuyenteTenantFilter
                enabled: false
```

```
// In your middleware (kernel.request listener):
$contribuyente = $contribuyenteRepository->find($request->get('contribuyente_id'));
$this->contribuyenteContext->setCurrent($contribuyente);

$filter = $this->entityManager->getFilters()->enable('contribuyente_tenant');
$filter->setParameter('contribuyente_id', (string) $contribuyente->getId(),
'integer');
```

That's it — the exact same pattern the bundle uses internally.

API Platform

The `IdentityTenantFilter` works transparently with API Platform. Since API Platform uses Doctrine under the hood, the filter applies to all listing operations automatically:

```
GET /api/contribuyentes
→ SQL: SELECT ... FROM contribuyentes WHERE organization_id = 42
```

No custom state provider needed. Just set the tenant context in your middleware and API Platform listings are filtered.

For **per-object authorization** (can this user access THIS specific resource?), use API Platform's `security` attribute with the bundle's voters — see Authorization → API Platform.

Summary

Tenant type	Provided by	Interface to implement	Context service
Organization	Bundle	`OrganizationOwnedInterface`	`OrganizationContextInterface`

Tenant type	Provided by	Interface to implement	Context service
User	Bundle	`UserOwnedInterface`	`UserContextInterface`
Custom resource	App	App-defined interface	App-defined context

Last updated on 09/09/2026