Features

Everything the bundle provides, grouped by module.

Anonymous · 09/09/2026

Features

A reference of every capability the bundle provides, grouped by module. Each item links to its dedicated page where applicable.

Core

The foundation shared by all other modules.

- **Base resource abstraction** — `BaseResource` MappedSuperclass with `id`, `uuid` (v7), `slug`, owner, optional organization, and timestamps. App entities extend it to inherit the standard shape.
- **Seeder pattern** — `SeederInterface` + `derafu:platform:seed` command. Seeders are auto-discovered via service tags; run them at deploy time or via cron.
- **Resource registry** — `ResourceRegistryInterface` collects every app resource that implements `ResourceDescriptorInterface`, used by the Apps and Notifications modules to know which entities support integrations or webhooks.
- **Settings section registry** — `SettingsSectionProviderInterface` lets any module contribute sections to the settings sidebar; IdentityUi, Apps, and Notifications each register theirs automatically.

Identity

User management, authentication, organizations, and authorization. See User lifecycle, Organizations, and Authorization for full details.

Users and authentication

- **User entity** — UUID v7, `email` as login identifier, hashed password, `active` flag, `emailVerifiedAt`, and a free-form `config` JSON column for per-app data. Implements Symfony's `UserInterface`.
- **Registration with email verification** — `Registrar` creates the user, issues a single-use `email_verify` token, and dispatches `UserRegistered` with the plaintext token. The bundle stays transport-agnostic; the app's listener sends the email.
- **Password reset** — `PasswordResetter` handles request and confirm phases. Silent no-op when

the email is not registered (anti-enumeration). Dispatches `PasswordResetRequested` and `UserPasswordChanged`.
- **Email change** — `EmailChanger` binds the new address to the token payload; the current email is not touched until the token is consumed. Dispatches `EmailChangeRequested` and `UserEmailChanged` (with both old and new addresses).
- **Magic links (passwordless login)** — `MagicLinkManager` issues a short-lived token; `MagicLinkAuthenticator` consumes `_magic_link_token` from the query string and authenticates without a password.
- **Secure one-use tokens** — All lifecycle tokens use 256 bits of entropy, SHA-256 hashed at rest, `hash_equals` for timing-safe validation, configurable TTL per type.
- **JWT tokens** — `JwtTokenManager` wraps Lexik JWT behind a stable bundle boundary; issued tokens are persisted for audit and revocation.
- **Locale and timezone** — `Locale` and `Timezone` entities store supported options; `LocaleResolver` resolves the user's preferred locale from the configured list.

Organizations and teams

- **GitHub-style organizations** — `Organization` entity with UUID, no hardcoded owner column. The owner is the membership with `role.code = 'org.owner'`, which lets ownership transfer be a role swap.
- **Organization memberships** — `OrganizationMembership` pivot (user + organization + role). Unique constraint `(user, organization)`. `OrganizationManager` handles create, addMember, removeMember, and transferOwnership atomically.
- **Organization invitations** — `InvitationManager` issues SHA-256 hashed, email-bound invitations. Accept validates email match (case-insensitive) and creates the membership transactionally. Invitations are retained as audit trail.
- **Teams** — Named groups of org members with a URL-friendly `slug`. Assign a team to a resource with a role and all members inherit access. `TeamManager` handles create/addMember/removeMember with org-membership validation.
- **Ownership transfer** — Atomic swap of the owner role to a new member with caller-controlled demotion of the previous owner.

Authorization (RBAC)

- **Four-level cascading permission checker** — `PermissionChecker` resolves: resource-access → team-access → organization-membership → global. A super-admin passes every check without explicit grants. See Authorization.
- **Symfony voters (auto-registered)** — `PermissionVoter` (global), `OrganizationPermissionVoter` (org subject), `ResourcePermissionVoter` (resource subject). All add human-readable deny reasons to Symfony's Vote for Web Profiler debugging.
- **API Platform integration** — Voters fire via `is_granted('read', object)` with no extra configuration. The Doctrine tenant filter applies to listing operations automatically.

Security

- **Sudo mode** — Session-based re-authentication window (default 15 min). Auto-granted on every login; no DB column. `SudoManager` checks and refreshes the window.
- **API keys with scopes** — Long-lived `dik_` tokens; SHA-256 hashed at rest. `ApiKeyAuthenticator` fires on `Authorization: Bearer dik_...`. `#[RequiresScope]` enforces per-endpoint scope checks. Session users bypass scopes entirely.
- **Rate limiting per API key** — Fixed-window algorithm with GitHub-style `X-RateLimit-*` headers. Disabled by default. PSR-6 default implementation; a Redis atomic-`INCR` variant is also provided.
- **Account lockout** — Automatic lock after N failed login attempts (default 5, 30 min lock). `AccountLockListener` hooks into Symfony's authentication events; zero wiring needed.
- **Login history** — Every successful login is recorded (IP, User-Agent, authenticator). `LoginHistoryManager` provides queries for a security-activity page; `identity:login-history:purge` prunes old records.
- **Impersonation audit** — `ImpersonationAuditListener` records every `switch_user` session (admin, target, IP, start/end time). Fully automatic.
- **Two-factor authentication** — TOTP-based 2FA via Scheb 2FA Bundle, with QR code enrollment. `TwoFactorEnrollmentManager` manages enable/disable flows.

Identity UI

Ready-made controllers, routes, and Twig templates. All templates extend your `parent_layout` and can be overridden individually. See Identity UI.

- **Auth pages** — Login, register, password reset, magic link request, email verification wall, sudo confirmation, 2FA code entry.
- **Account settings** — Name, avatar, locale, timezone, password change, email change, active sessions, login history, API keys, JWT tokens.
- **Organization settings** — Create orgs, manage members, invitations, teams.
- **Locale negotiation** — `LocaleNegotiationListener` negotiates the user's locale from the configured list on every request.
- **Email verification enforcement** — `require_email_verification: true` redirects unverified users to the verification wall automatically.
- **Built-in transactional emails** — `IdentityMailerListener` sends verification, password reset, invitation, and other identity emails out of the box.

Apps

Plugin/integration ecosystem. See Apps.

- **App definitions** — Apps are services implementing `AppDefinitionInterface`, auto-discovered via service tags. Each declares code, name, category, icon, and supported scopes.

- **Three installation scopes** — Per-user (`BaseUserAppInstallation`), per-organization (`BaseOrganizationAppInstallation`), per-resource (`BaseResourceAppInstallation`).
- **Encrypted configuration** — `AppConfigEncryptor` transparently encrypts `sensitiveConfigKeys` at rest in the installation's JSON column.
- **Capability system** — Apps implement capability interfaces; `CapabilityRegistryInterface` answers "which installed apps for this org support capability X?".
- **App registry** — `AppRegistryInterface` collects all apps and applies per-deployment catalog overrides (scopes, flags, tags, supported resources) from YAML config.
- **Settings UI** — Users and org admins browse, install, configure, and remove apps from a dedicated settings section; no custom controllers needed.

Notifications

Event-driven delivery via email, in-app, and webhooks. See Notifications.

- **Event definitions** — Implement `EventDefinitionProviderInterface` to register notification events; they appear in preferences UI and webhook triggers automatically.
- **Email channel** — Renders Twig templates and delivers via Symfony Mailer.
- **In-app channel** — Persists `InAppNotification` entities; displayed in the UI with read/unread state.
- **Webhook channel** — HTTP POST to user/org-configured endpoints, HMAC-signed (`X-Webhook-Signature`), delivered async via Symfony Messenger. Delivery history is retained and cleaned up via cron.
- **Recipient resolvers** — `NotificationRecipientResolverInterface` decouples recipient logic from dispatch logic; multiple resolvers can run per event.
- **Per-user preferences** — Users control which channels they receive per event type; the dispatcher respects preferences before routing.
- **Async delivery** — Messenger integration keeps webhook delivery out of the main request path.
- **Retention policy** — Configurable `max_age_days` and `max_per_endpoint` limits; `notifications:webhook-deliveries:cleanup` command for periodic pruning.

API

Helpers for API Platform integration.

- **Scope enforcement** — `#[RequiresScope]` on API Platform state processors; a compiler pass builds a static processor → scopes map at compile time.
- **Tabulator.js adapter** — `TabulatorQueryStringSubscriber` translates Tabulator.js query parameters to API Platform filter parameters.
- **Serializer helpers** — Custom serializers for bundle value objects and entities.

CLI commands

Command	What it does
`derafu:platform:seed`	Run registered seeders
`identity:tokens:purge`	Delete expired verification tokens
`identity:login-history:purge --days=N`	Delete login records older than N days
`notifications:webhook-deliveries:cleanup`	Apply webhook delivery retention policy

Last updated on 09/09/2026