OSCRAT

Open-Source Cyber Resilience Act Tools

D3.8 - Full Product Documentation

08.09.2026

Document History			
Version	**Date**	**Modified by**	**Comments**
0	24.07.2026	OVES Enterprise	First version
0.1	31.07.2026	OVES Enterprise	Internal review UNICIS and Łukasiewicz-AI
1.0	08.09.2026	OVES Enterprise	Final version after modifications

Abstract

This document presents the technical architecture and implementation of the OSCRAT platform, an open-core solution supporting compliance with the EU Cyber Resilience Act (CRA). It describes the platform's architecture, security model, API design, data management, and core functionalities, including SBOM generation, vulnerability management, risk assessment, compliance assessment, incident management, and audit logging. The document also provides guidance for deployment, operation, and future platform extension.

Keywords

Cyber Resilience Act (CRA); OSCRAT; cybersecurity compliance; software supply chain security; Software Bill of Materials (SBOM); vulnerability management; risk assessment; compliance assessment; incident management; audit logging; secure software development; trust management platform.

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

DISCLAIMER

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

Table of contents

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

Figures

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

1. Introduction

This document is the technical documentation of the OSCRAT platform, an open-core (Apache-2.0) trust management platform implementing the workflows of the EU Cyber Resilience Act (CRA): product registration and applicability checking, role-based compliance assessment, SBOM generation and vulnerability scanning, incident reporting, risk assessment, configuration management, security awareness training, technical documentation up to the Declaration of Conformity, and a full audit trail.

It is written for developers and operators working on the platform. Sections 2 through 6 establish the foundations shared by all features: the architecture, the authentication and permission system, the API conventions, the code organization and security practices, and the file and attachment infrastructure. Section 7 describes each feature at the implementation level: data model, API surface, background jobs, and key mechanisms.

2. Architecture Overview

2.1 System Components

The monorepo (`pnpm-workspace.yaml`, globs `apps/**` and `packages/**`) contains two deployable applications and two shared packages:

- `apps/frontend` (`@oscrat/frontend`): Next.js 16 application using the pages router. It serves both the web UI and the entire REST API (the route files under `pages/api`).
- `apps/jobrunner` (`@oscrat/jobrunner`): a standalone Node worker that polls the database job queue and executes long-running work: SBOM generation, vulnerability scans, configuration-scan processing. It shells out to Syft, Grype, and the OpenSCAP tooling.
- `packages/model` (`@oscrat/model`): the shared data layer used by both apps. Contains the Prisma schema and migrations, the database-operation modules under `operations/` (one per domain area: team, product, version, vulnerability, incident, reports, worker jobs, audit, dashboard, encryption, and others), Yup validation schemas (`schemas/`, including the job payload schemas), shared TypeScript types (`types/`), and the audit logger (`audit/`).
- `packages/config`: a thin shared configuration package.

Root scripts orchestrate the workspace: every build first runs `pnpm --filter @oscrat/model generate` (Prisma client generation), and a `only-allow pnpm` preinstall hook enforces the package manager.

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

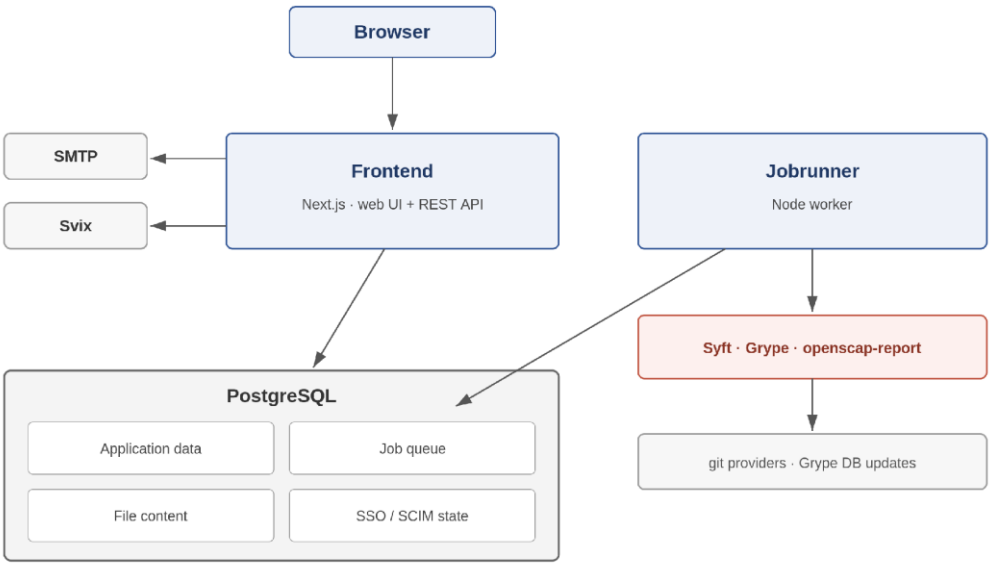

Figure 1. System components and their connections.

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

2.2 Repository Layout

High-level project tree, annotated with the directories a new developer will touch most:

oscrat-oves/	
apps/	
frontend/	Next.js app: web UI + the REST API
pages/	file-based routing
api/	all REST endpoints (auth, oauth, scim, teams/...)
auth/	login, signup, reset, SSO pages
organization/[slug]/	the app: dashboard, products, tasks, settings
form/	public CRA applicability check
components/	React components, grouped by feature
hooks/	feature-facing hooks (+ hooks/oscrat/)
lib/	server and client infrastructure
api/	axios client, endpoint modules, react-query hooks, query keys
middleware/	withTeamAuth / withUserAuth / withApiHandler guards
validation/	Yup schemas, one file per entity + shared primitives
compliance/	questionnaire templates, translations, PDF labels
email/	mail service + React Email templates
*.ts	permissions, errors, SSO, webhooks, db client, rate limiting
locales/	i18n resources (English at release)
models/	server-side helpers used by API routes
utils/, constants/	shared helpers, single-source definitions
jobrunner/	
src/index.ts	JobRunner class: polling, concurrency, timers
src/jobs/	one module per background job type
src/utils/	git, sbom, scanner, archive helpers
nixpacks.json	provisions syft / grype / openscap-report
packages/	
model/	shared data layer used by both apps
prisma/	schema.prisma + migrations
operations/	database operations, one module per area
audit/	transactional audit logger
schemas/	shared Yup schemas (incl. job payloads)
types/, constants/	shared domain types and constants
config/	thin shared configuration
docker-compose.yml	local PostgreSQL for development
railway_*.json	Railway deployment configs
pnpm-workspace.yaml	workspace definition

2.3 Data Layer

All state lives in PostgreSQL, accessed exclusively through Prisma. There is no object storage service; file content is stored as bytes in Postgres. Storage is split

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

across two tables so listings never pull blobs: `File` holds `fileData Bytes`, `fileSize`, and `mimeType`, while `Attachment` holds display metadata (`name`, `fileSize`, `mimeType`, `description`) and the polymorphic links to owning entities (see section 6). Scan inputs are gzip-compressed before storage and generated reports are zipped (`adm-zip`).

2.4 Job Queue and Background Processing

Background work uses a database-backed queue, the `WorkerJob` table, with no external broker. A job row records its type, status (pending, in progress, then completed, failed, or cancelled), the triggering user, the organization context (plus optional product and version), a JSON payload and result, error details, and process start/end timestamps. Reports produced by jobs hold a one-to-one back-reference to the job that created them.

The jobrunner (`apps/jobrunner/src/index.ts`) is a single `JobRunner` class. `popWorkerJob` claims the oldest pending job and marks it in-progress inside a single database transaction, so the pop is atomic across multiple runner instances. Concurrency is bounded by `MAX_CONCURRENT_JOBS` and the polling interval by `JOB_POLL_INTERVAL_MS` (defaults in Appendix C), with immediate re-dispatch when a slot frees. Job payloads are validated with Yup before execution. Failures are caught per job and persisted via `finishWorkerJob`. There is no automatic retry; a failed job stays `FAILED` until re-triggered. `SIGINT`/`SIGTERM` drain running jobs before exit. One recurring 24-hour timer runs alongside the queue: the Grype vulnerability database refresh (deferred until no jobs are running).

Figure 2. Worker job lifecycle.

2.5 External Tooling and Process Execution

The jobrunner invokes three external tools: **Syft** (SBOM generation and format conversion), **Grype** v0.116.0 (vulnerability scanning against SBOMs and repositories), and **openscap-report** 1.0.0 (rendering OpenSCAP ARF/XCCDF results to HTML), alongside `git` and `npm` for repository work. At startup the runner verifies that `syft`, `grype`, and `oscap-report` are on `PATH`, creates and write-tests the workspace root (`JOBRUNNER_WORKSPACE_ROOT`, default `/tmp/jobrunner-workspace`), and runs `grype db update`.

2.5.1 How processes are executed

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

All child processes run through `zx` template literals with default settings. zx quotes every interpolated value, so paths, refs, and filenames from user input cannot inject shell syntax. Working directories are set per invocation with `$({cwd})`, never by changing the process directory, and environment overrides are scoped the same way: scans run with `$({env: GRYPE_SCAN_ENV})`, which sets `GRYPE_DB_AUTO_UPDATE=false` and `GRYPE_DB_VALIDATE_AGE=true`, so individual scans never update the vulnerability database (that is centralized in the 24-hour refresh) but refuse to run against a stale one. Tools write their results to files rather than stdout (`--file`, `-o … =path`); only `oscap-report` and small git queries are captured from stdout.

Each job runs inside its own temporary directory, created by `withTempDirectory` (`src/utils/filesystem.ts`) with a job-id prefix and removed automatically when the job settles, on success or failure. Uploaded filenames are basename-sanitized before being joined into the directory. The lockfile step in SBOM generation (`npm install --package-lock-only`) is best-effort: its failure is logged as a warning and generation continues.

2.5.2 Git execution

Clones are HTTPS-only (no SSH path): a full clone followed by `git fetch --all --tags`, then checkout by ref priority (commit, tag, branch), with the resolved HEAD recorded in the job result. For private repositories the access token is decrypted at clone time and embedded in the clone URL using the provider's scheme (`x-access-token:` for GitHub, `oauth2:` for GitLab, `x-token-auth:` for Bitbucket).

2.5.3 Failure handling

Process failures surface as `JobError` values with codes from a shared catalog (`packages/model/constants/errorCodes.ts`: `JOB_*`, `SBOM_*`, `VULN_*`, `CONFIG_*`, `REPO_*`, `UNKNOWN`). `translateError` logs the full process output (exit code, stdout, stderr, signal) for diagnosis but returns a fixed, code-only error to be persisted, so raw tool output never reaches the user. Jobs also check for their expected output file after each tool run, catching the case where a tool exits zero without producing a result. The error code and message are written to the job row (`errCode`/`errMessage`) with status `FAILED`.

2.5.4 Operational characteristics

Child processes run without timeouts, so a hung clone or scan occupies one of the `MAX_CONCURRENT_JOBS` slots until the runner is restarted; watch job durations when operating. Tool outputs are read fully into memory before compression and storage, so runner memory scales with the largest artifact processed. Because clone URLs embed decrypted tokens and process errors log the failing command line, jobrunner logs should be treated as sensitive (section 5).

2.6 Deployment Model

The platform is platform-agnostic: any target that can run two Node services next to a PostgreSQL database can host it. Production deployment is not part of

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

the delivered project scope. What ships is the deployment model below, plus a working reference deployment on Railway that the project team uses for development.

2.6.1 How to think about deployment

A deployment consists of three parts. **PostgreSQL** is the only stateful component; all data, including file content, lives here, so backing up the database backs up everything. The **frontend** is a standard Next.js server process, effectively stateless (sessions are JWTs validated against the database), and can be scaled horizontally behind any load balancer; TLS termination is expected in front of it (secure cookies activate when `APP_URL` is https). The **jobrunner** is one or more worker processes that need no inbound network at all: only database access, outbound network for git clones and Grype database updates, local scratch disk, and the three external binaries (Syft, Grype, openscap-report) installed on the host or image. Multiple jobrunner instances can run in parallel (section 2.4).

A target environment must additionally provide: schema migrations applied before or with each frontend rollout (`prisma migrate deploy`); the environment variables from `.env.example`, with `DATABASE_URL` shared by both services and the embedded SSO layer; SMTP egress for the mail service; and a liveness probe wired to `GET /api/health`. Nothing assumes a specific vendor. The same model maps to a pair of containers, a Kubernetes deployment plus a worker, or any PaaS.

2.6.2 Reference deployment (Railway, used for development)

The repository carries a concrete implementation of this model for Railway with nixpacks builds, as JSON configs at the repository root. They also document what any other target needs to replicate:

- `railway_frontend.json`: build `pnpm frontend:build`, pre-deploy `pnpm db:migrate:deploy` (automated Prisma migrations on every deploy), start `pnpm frontend:start`, health check `GET /api/health` (300 s timeout, restart `on_failure`, max 10 retries). The health endpoint runs `SELECT 1` against the database and returns `{status, version, database, timestamp}`, responding 503 on database failure.
- `railway_jobrunner.json`: uses `apps/jobrunner/nixpacks.json`, which provisions `nodejs`, `syft`, `python3`, `pip`, and `curl` from nixpkgs, installs Grype v0.116.0 via the Anchore install script into `/usr/local/bin`, and installs `openscap-report==1.0.0` into a Python venv symlinked to `/usr/bin/oscap-report`.

Local development uses `docker-compose.yml` (a single `postgres:15` service on port 5432 with a `pg_isready` health check). `pnpm dev:setup` performs install, database up, `prisma migrate dev`, and seed; the dev server runs on port 4002.

2.7 Technology Stack Reference

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

Component	Technology / Version
Runtime	Node 22.14.0 (`.nvmrc`), TypeScript 5.4.5
Web framework	Next.js 16.1.2 (pages router), React 18.3.1
ORM / database	Prisma 6.7, PostgreSQL 15
Styling / UI	Tailwind CSS 3.4.3, daisyUI 4.11, Atlaskit component packages, Emotion 11
Data fetching	@tanstack/react-query 5.69 (axios client; SWR unused)
Forms / validation	Formik 2.4.6 + Yup 1.4 (API and jobrunner also validate with Yup)
Auth	NextAuth 4.24.13 (JWT strategy) + @boxyhq/saml-jackson 1.26.7 (SSO/SCIM)
Webhooks	Svix 1.24.0
Charts / PDF	Chart.js 4.4 + react-chartjs-2 5.2; @react-pdf/renderer 4; jsPDF 3
i18n	next-i18next 15.3 (English shipped at release)
Email	nodemailer 6.9 + React Email templates
Editor	@mdxeditor/editor 3.52 (markdown documentation)
Jobrunner libs	zx 8.5.5, adm-zip 0.5, fast-xml-parser 5.7, p-queue 8.1, tempy 3.1
External binaries	Syft (nixpkgs), Grype v0.116.0, openscap-report 1.0.0
CI	GitHub Actions: `pnpm check-all` (lint, typecheck, build)

3. Authentication and Access Control

3.1 Providers and Sign-in

Authentication is NextAuth v4 with the Prisma adapter (`pages/api/auth/[...nextauth].ts`). Providers are registered only when enabled through the comma-separated `AUTH_PROVIDERS` environment variable (code default `github,credentials`; `.env.example` ships `email,credentials`). Supported providers: `credentials`, `github`, `google`, `saml` (BoxyHQ SAML), and `email` (magic link; signs in existing users only, never creates accounts, and is off unless explicitly listed).

The credentials `authorize` flow runs, in order: a rate limiter (`lib/rate-limit.ts`, 5 attempts per minute per IP), reCAPTCHA validation (no-op when keys are unset), user lookup by lower-cased email, an email-verification gate when `CONFIRM_EMAIL` is enabled, and bcrypt password comparison (hash cost 12). Failures surface as

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

string error codes mapped to user-facing text in `lib/errorHandler.ts`. Rate limiting is per IP; there is no per-account lockout.

Signup is `POST /api/auth/join`: Yup `userSignupSchema` validation, optional reCAPTCHA, optional invitation token (expiry-checked; joining by invitation pre-verifies the email), duplicate-email check, password hash, and, when `CONFIRM_EMAIL` is set, a 24-hour verification token delivered by email. Password reset (`forgot-password` / `reset-password`) is rate-limited 5/min/IP, uses a one-hour token stored in `prisma.passwordReset`, and enforces the password policy before re-hashing. Signups from non-business email domains can be blocked with `DISABLE_NON_BUSINESS_EMAIL_SIGNUP`.

3.2 Sessions

Sessions use the JWT strategy with a hybrid revocable model: each token is backed by a server-side session row created at login (`createServerSession`) and checked on every request (`refreshServerSession`, which also slides the expiry). Logout deletes the row, so the token is invalidated server-side; a missing or expired row rejects the request. The maximum age is set by `JWT_SESSION_MAX_AGE` (7 days). The session callback also re-reads the user from the database on each request, so deleted users are rejected immediately. In production the session and CSRF cookies carry hardened attributes: HTTP-only, secure, and same-site restricted.

3.3 SSO and SCIM

SSO and SCIM are provided by SAML Jackson running embedded in the frontend (`lib/jackson.ts`), storing its state in the main database; no separate service is deployed. On SAML sign-in the team is resolved from the SAML tenant and IdP groups map to roles: groups carrying the `GROUP_PREFIX` grant `ADMIN`, anything else receives the team's default role. SCIM 2.0 is served at `pages/api/scim/v2.0/[...directory]`, authenticated by bearer secret. Every provisioning path (SAML just-in-time login, SCIM user creation, group changes) issues the awareness-training task (section 7.10).

Account linking is strict: an external identity attaches to an existing user only if that exact provider identity was previously linked to the same account. Matching by email alone is refused, and Google sign-ups with unverified email addresses are rejected.

3.4 Re-authentication for Sensitive Changes

Changing the account password (`PUT /api/password`) and changing the account email (`PUT /api/email`) both require the current password, verified with bcrypt, before the change is applied. Email changes additionally go through a 24-hour confirmation token when `CONFIRM_EMAIL` is enabled, confirmed at `/auth/confirm-email-change`.

3.5 Roles and Permissions

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

Membership roles are `OWNER`, `ADMIN`, `MEMBER`, `AUDITOR` (enum `Role`, stored on `TeamMember`). The permission matrix (`lib/permissions.ts`) maps role × resource × action (`create/update/read/delete/leave`) across ten resources: `team`, `team_member`, `team_invitation`, `team_sso`, `team_dsync`, `team_audit_log`, `team_webhook`, `team_api_key`, `task`, `documentation`.

Role	Permissions
OWNER / ADMIN	All actions (*) on all ten resources (the two matrices are identical).
MEMBER	`team`: read, leave · `team_member`: read · `task`: all · `documentation`: all
AUDITOR	Read-only on `team`, `team_member`, `task`, `documentation`

Enforcement is server-side via `throwIfNotAllowed(teamMember, resource, action)` inside the `withTeamAuth` middleware (section 4.2). CRA domain resources (products, versions, vulnerabilities, incidents, assessments) have no dedicated resource entry; their routes call `withTeamAuth()` without a permission tuple, so any team member may access them, with team membership itself as the boundary. The client mirrors the matrix: `usePermissions` fetches the caller's permissions from `/api/teams/[slug]/permissions` and `useCanAccess` re-implements the */includes check for conditional UI.

4. API Architecture

4.1 Route Organization

All API routes are Next.js pages-router API routes under `pages/api`. Top-level groups: `auth/` (NextAuth, join, password reset, SSO verify/ACS), `oauth/` (authorize, token, saml, userinfo), `scim/v2.0/[...directory]`, `invitations/[token]`, `teams/`, and singletons `health`, `users`, `password`, `email`, `idp`, `csp-report`, and `well-known/saml.cer`.

Everything team-scoped lives under `/api/teams/[slug]/...`: members, invitations, permissions, api-keys, webhooks, saml, directory-sync, audit-logs, data/[dataKey], tasks, documentation, assessments, csc, compliance data, dashboard summary, and attachment downloads. Nesting mirrors the domain hierarchy; the deepest branch sits on the product version:

`/api/teams/[slug]/products/[productId]/versions/[versionId]/`

with the sub-resources `incidents`, `repositories`, `vulnerabilities`, `sbom-reports`, `vulnerability-scan-reports`, `configuration-scan-reports`, `car`, `doc`, and `attachments`.

Public, unauthenticated routes exist only for published public documentation (`/api/teams/[slug]/public/...`) and return a whitelisted payload. Enterprise route files are gated early by feature flags (`FEATURE_TEAM_SSO/DSYNC/AUDIT_LOG/WEBHOOK/API_KEY`), returning 404 when disabled.

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

4.2 Middleware and Guard Layering

The guard system is built on one factory, `createMiddleware` (`lib/middleware/auth.ts`), which handles request logging and converts any thrown error into the uniform envelope `{ error: { message, code, values } }`, with the status taken from `ApiError.status` (401 for unauthorized, 500 fallback). Three wrappers derive from it:

- `withTeamAuth([resource, action]?)`: resolves the session, loads the caller's `TeamMember` for `req.query.slug`, optionally enforces the permission tuple via `throwIfNotAllowed`, then attaches `req.teamContext = {user, teamMember}` and `req.auditInfo = {user, team, productId?, versionId?}` (ids pulled from the route query) for downstream audit logging.
- `withUserAuth()`: session only; attaches `req.userContext`.
- `withApiHandler`: no authentication; logging and error envelope only. Used by self-authenticating or public routes (join, password reset, health, oauth, invitations).

The dispatch convention inside a handler is `switch (req.method)`, with the guard applied per method: each case returns `withTeamAuth([resource, action])(handleGET)(req, res)`, so different methods on one route can require different permissions. The default branch sets the `Allow` header and throws `ApiError(405)`.

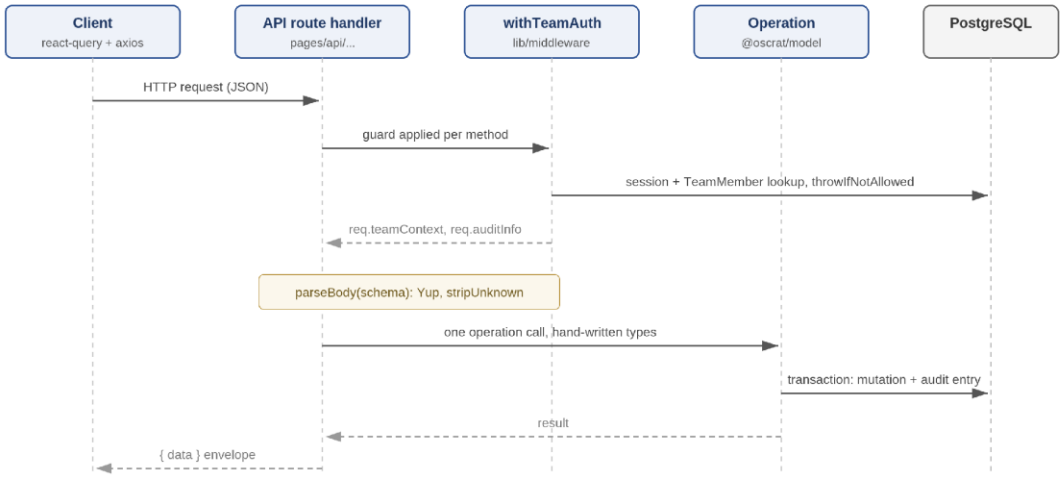

Figure 3. Request path from client to database, including the guard and validation steps.

4.3 Validation Architecture

Validation is Yup at every boundary. The same schema objects guard the forms, the API, and the background jobs, and this system is also where the platform enforces its input-security posture.

4.3.1 Where schemas live

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

Input-validation schemas live in `apps/frontend/lib/validation/`, one file per entity (task, product, version, vulnerability, incident, repository, team, documentation, assessment, signup, risk, and so on) plus two primitive collections. `inputs.ts` holds shared field primitives: name, email, password, phone, tax id, title, description, acronym, and the safe-text regexes, each carrying its length limits (title 100, description 500, password 8 to 128), so limits are defined once. `date.ts` adds shared date rules such as not-in-the-future. The second location is `packages/model/schemas/jobPayloads.ts`: one schema per background-job payload, shared between the frontend (which enqueues) and the jobrunner (which validates before executing).

Composition follows a consistent style. Primitives are defined unrequired, and each entity schema applies `.required()` at the composition site with an i18n key as the message; validation messages are translation keys, translated at render time, never English strings. Create and update variants are written as separate schemas. Where validation needs runtime context, schemas become factories: `createTaskCreateSchema()` and the product-name uniqueness schema close over data such as the list of existing product names.

4.3.2 From schemas to types

TypeScript types and Yup schemas relate in two directions, split by layer. Request and form shapes are derived from schemas: nearly every validation file ends with `Yup.InferType` exports (`TaskCreateData`, `ProductCreateData`, `VersionCreateData`, `RepositoryCreateInput`, and the rest), so the wire shape has exactly one definition. Domain types are hand-written in `packages/model/types/` and the schema is annotated against the type, which makes the compiler verify that schema and domain type stay in sync without generating the type. API handlers are the adapter between the two: `parseBody` returns the inferred type, the handler applies defaults and conversions, and calls a model operation whose signature uses the hand-written types. Job payloads are the one fully derived chain: payload types are inferred from the schemas and composed into the `WorkerJobPayload` union consumed by the jobrunner.

4.3.3 One schema, three boundaries

- **Forms.** Formik components import the same exported schema objects as their `validationSchema`. The task, product, version, vulnerability, risk, and signup forms all validate client-side with the identical object the server uses, so the two cannot drift. Field errors surface as i18n keys and are translated in the component.
- **API.** Handlers call `parseBody(schema, req)` (`lib/validation/validateRequest.ts`), which validates with `stripUnknown: true` and converts any `Yup.ValidationError` into `ApiError(400)`. Undeclared body keys are silently dropped: the server-side allow-list that prevents mass-assignment of fields like team or owner.
- **Jobrunner.** Every job executor validates its payload first, before any database or filesystem access (strip-unknown, all errors collected); failures become typed job errors. The shared `safeFilename` rule (no path separators) guards

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

exactly the payloads whose filename is joined into a temp directory, as the path-traversal defense.

4.3.4 Recurring patterns

Validation doubles as sanitization: the description-like primitives run DOMPurify with all tags stripped inside a Yup `.transform`, so XSS stripping happens in the same pass as validation; trimming is near-universal and emails are lower-cased. Conditional rules use `.when`: tax id required only for certain legal forms, incident detail fields required only when their flag is set, repository token required only for private authentication. Custom `.test` rules handle the hard cases: tax-id validation across EU formats via `stdnum`, phone numbers via `libphonenumber-js` with the country code read from the sibling field, and case-insensitive product-name uniqueness. Enum fields are validated against the Prisma enum's values, keeping the enum the single source of allowed values.

4.3.5 Error handling

`ApiError` (`lib/errors.ts`) carries `{status, code?, values?}`; Prisma error guards handle unique-constraint violations. Success responses use the `{ data: ... }` envelope; failures use `{ error: { message, code?, values? } }`, the message being a translation key the client renders in the user's language.

4.4 Frontend Data Layering

Client-side data access is layered in four tiers, built exclusively on react-query (the legacy `lib/fetcher.ts` fetcher is unused):

- **1. HTTP client.** `lib/api/client.ts` exports an axios-based `ApiClient` singleton (`baseURL: /api`) whose typed verbs unwrap the `{data}` envelope. A response interceptor redirects to sign-in on any 401, and failures are normalized to a uniform error shape.
- **2. Endpoint modules.** `lib/api/endpoints/*.ts` are thin URL builders per resource (`teamsEndpoints`, and `endpoints/oscrat/` for projects, versions, vulnerabilities, incidents, assessments, repositories, jobs, dashboard). `endpoints/oscrat/urls.ts` composes the nested team → product → version paths in one place.
- **3. Query/mutation hooks.** `lib/api/hooks/*.ts` wrap react-query. A shared query client applies conservative defaults: no automatic retries, no refetch on window focus, and short-lived caching. Query keys are centralized in `lib/api/queryKeys.ts`. Mutations invalidate their detail and list keys on success; cross-cutting count refreshes go through a shared invalidation helper (`lib/api/hooks/oscrat/invalidations.ts`), called from every incident, vulnerability, and task mutation.
- **4. Feature hooks.** `hooks/*.ts` and `hooks/oscrat/*.ts` compose the tier-3 hooks into flat, component-facing state (e.g. `useTasks`, `useOscratProject`, `useComplianceState`). Toasts are raised at the component and form level, not in the data layer.

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

5. Code Organization and Conventions

5.1 The model package is the data layer

Everything that touches the database lives in `packages/model`, and both apps consume it. The Prisma schema and migrations sit in `prisma/`. All domain logic sits in `operations/`, one module per domain area: product, version, task, vulnerability, incident, repository, the three report types, team, documentation, assessment, attachment, file, worker jobs, audit, dashboard, ownership, encryption, and the enterprise integrations. An operation module owns its transactions: it opens the database transaction, performs the mutation, and writes the audit entry within it. Ownership assertions (does this version belong to this team?) are also operations (`operations/ownership.ts`), so tenant checks happen in the data layer, not in handlers.

5.2 Where types live

Types come from three sources. Entity types, enums, and relation-shaped query results are generated or derived by Prisma from `schema.prisma` and re-exported through `@oscrat/model`; a schema change propagates to both apps at compile time. Hand-written types cover only what Prisma cannot type: the contents of `Json` columns (task properties, scan summaries, job payloads, CRA form data), kept in `packages/model/types/`. Request and form shapes are inferred from the Yup schemas (section 4.3).

5.3 Handlers delegate to operations

API handlers stay small. A typical route does four things: dispatch on the HTTP method, apply the permission guard per method, validate the body with `parseBody`, and call one operation from `@oscrat/model/operations`. Business rules, transactions, audit, and tenant scoping all live in the operation. When extending the platform, keep to this split: logic accumulating in a handler belongs in an operation module. Note: `apps/frontend/models/` is an older wrapper layer inherited from the starter kit that some routes still call; newer routes import operations directly. Both paths end in `packages/model/operations`, which is the layer to extend.

5.4 One task model for everything

Risk assessment, awareness training, configuration remediation, and compliance follow-up are all built on the single `Task` model rather than four separate systems: a taskType column and typed keys in the task properties. They share one lifecycle, one audit trail, and one UI (sections 7.7 to 7.10). New task-like workflows should follow the same pattern.

5.5 Single-source definitions

Values that would otherwise drift across the codebase are defined once and imported everywhere: the listing page size (`constants/pagination.ts`), chart colors (defined in `tailwind.config.js`, re-exported to charting code via

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

`lib/chartTokens.ts`), field length limits (in the validation primitives, section 4.3), enum-to-label maps, and cache invalidation helpers (section 4.4). Consistency in paging, colors, and limits therefore does not depend on manual upkeep.

5.6 Security practices

The platform is multi-tenant, and its security model rests on a small set of practices enforced consistently across the codebase. The platform was externally penetration-tested before release; the practices below reflect its final state and are the rules to uphold when extending it. The two open items from that test are deployment-configuration matters and sit with the operating team.

5.6.1 Tenant isolation

Team membership is the primary boundary, enforced in the data layer rather than in handlers. Every team-scoped query filters by the caller's team; ownership assertions (`operations/ownership.ts`) verify that nested resources belong to the team before any mutation; attachment access resolves through `attachmentTeamScope` across all owning relations (section 6). Unscoped lookups must not be used in routes, and misses return 404, not 403, so resource existence is not leaked. When adding a relation to a shared table, extend the scope query in the same change.

5.6.2 Input handling

Every request body passes a Yup schema with `stripUnknown: true`; undeclared fields are dropped, which prevents mass-assignment of fields like team or owner. Free-text fields are sanitized with DOMPurify inside the validation transforms (section 4.3). User-supplied filenames are never trusted: uploads reduce them to a basename, job payloads enforce the `safeFilename` rule (no path separators), and file bytes live in the database, so no filesystem path is ever derived from user input. Uploads pass size and type allow-lists per tier (section 6).

5.6.3 Authentication and sessions

Passwords are bcrypt with cost 12. Login and password-reset endpoints are rate-limited per IP. Sessions are revocable server-side and revoked on logout (section 3.2). Password and email changes require the current password (section 3.4). SSO and OAuth identities link to existing accounts only by the exact provider identity pair, never by email match (section 3.3).

5.6.4 Secrets

Repository access tokens are encrypted at rest (AES-256-GCM via `cryptr`, keyed by `TOKEN_ENCRYPTION_KEY`) and masked on API reads; only the jobrunner decrypts them, at clone time. API keys are stored hashed. Audit logging uses per-entity tracked-field whitelists, so secrets and file content never enter the log. Treat jobrunner logs as sensitive and restrict access to them: clone URLs embed decrypted tokens, and process failures log the failing command line (section 2.5).

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

5.6.5 Transport and headers

Security headers are set centrally in `next.config.js`: a Content-Security-Policy with `default-src 'self'`, `object-src 'none'`, `frame-ancestors 'none'`, and `script-src 'self' 'wasm-unsafe-eval'` (required by the PDF renderer's WASM), reporting to `/api/csp-report`; HSTS with a two-year `max-age`, includeSubDomains, and preload; `X-Frame-Options: DENY`; `nosniff`; a strict referrer policy; and a restrictive Permissions-Policy. Session cookies use the secure prefixes over HTTPS. The CSP is deployed in report-only mode; moving it to enforcing is an operational decision for the new team, informed by the reports collected at `/api/csp-report`.

5.6.6 Auditability

Audit entries are written on the same transaction as the mutation they record, so the trail cannot diverge from the committed state (section 7.12). File and report downloads are logged as read events. New entities that carry user data should be added to the audit coverage as part of their implementation, not afterwards.

6. Files and Attachments

A single attachment system handles binary content platform-wide: task attachments, version files, incident forensic evidence, vulnerability remediation plans, assessment evidence, the CAR and DoC on versions, and the SBOM and scan artifacts produced by the pipeline. There is no object storage service. Everything lives in Postgres, so one backup covers all data and no filesystem path is ever derived from user input.

6.1 Storage model

Storage is a two-table split. `File` holds the raw bytes (`fileData Bytes`) with size and MIME type; `Attachment` holds the display metadata (name, size, MIME type, description) and the polymorphic links to owning entities: task, version, incident, vulnerability, assessment, documentation, the three report types, and the named CAR/DoC relations on versions. The foreign key points from `Attachment` to `File` (`fileId`, unique). Size and MIME type are duplicated onto the attachment row so list views render entirely from `Attachment` without touching the blobs. Writes are transactional: file row, attachment row, and audit entry commit together.

6.2 Upload paths and validation

All uploads are multipart form-data parsed with formidable (Next.js body parsing disabled per route). There are three validation tiers:

- **Generic attachments** (tasks, version files, incident evidence, vulnerability files, assessments): 10 MB cap, zero-byte rejection, and an extension-to-MIME allow-list covering documents, structured data, archives, and images

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

(`lib/utils/fileUpload.ts`). The browser-sent MIME type must match what the extension implies.

- **CAR and DoC**: dedicated validation (`docFileValidation.ts`). The CAR accepts PDF/XLS/XLSX; the DoC accepts PDF only, and the server forces the stored MIME type to `application/pdf`. The same module exports a browser-side variant so client and server share one rule set.
- **Pipeline imports** (SBOM, configuration scans): a separate 100 MB, XML-only path, since scan artifacts legitimately exceed the generic cap.

Filenames are handled defensively: the user-supplied name is normalized and reduced to its basename, stripping any path components, before being stored as the display name. Because bytes live in the database, the name is only used for display and for the download header, never to address storage.

6.3 Compression

Scan inputs are gzip-compressed before storage, and only the file id travels in the job payload; the jobrunner gunzips on read and deletes the input file when the job finishes, on success or failure. Generated outputs (raw Grype results, rendered configuration reports) are zipped before being stored as attachments.

6.4 Download path and tenant isolation

There is exactly one download route, `GET /api/teams/[slug]/attachments/[attachmentId]/download`, and it always resolves the attachment through `attachmentTeamScope(teamId)`: an OR across all nine owning relations, including the nested paths such as vulnerability → version → product → team. An attachment is reachable only if some link resolves to the caller's organization; anything else returns a 404. This data-layer scoping is the tenant-isolation boundary for file content, and the unscoped lookup helper carries an explicit warning against use in routes. Every download writes an audit entry (action `attachment.download`, with the resolved linked entity in the metadata). There is no unauthenticated attachment download; public documentation pages serve no attachments.

6.5 Lifecycle

An attachment is created together with its file content and audit entry in one transaction, linked to exactly one owning entity. Over its life it is read through the scoped download route, and, for the CAR and DoC, replaced in place: the attachment row is stable and only the file bytes are swapped, so the version's pointers never dangle. Deletion follows the owning entity: attachments cascade away with their task, version, assessment, documentation, or report, while incident and vulnerability links set-null on delete, so evidence survives and remains reachable through its version link; CAR and DoC removal deletes attachment and file together. One special case: the generated SBOM attachment is linked to both its report and its version, and the vulnerability-scan

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

flow reads the SBOM bytes back through it, making it the working input of the rescan pipeline rather than just a downloadable artifact.

6.6 Client side

Uploads are plain `FormData` posts through the shared API client. A small hook family wraps the flows: a shared download hook (fetches a blob, saves via file-saver), extended per context by the task, version, and assessment attachment hooks, exposing uploading/deleting/downloading flags for the UI. Errors surface as toasts. Client-side pre-checks mirror the server rules (size caps and accepted extensions on the pickers), with the server remaining authoritative.

7. Features

The sections below describe each feature at the implementation level: data model, API surface, background jobs, and key mechanisms. They build on the foundations of sections 2 through 6.

7.1 Product and Version Management

Products and versions anchor the domain model; most other entities reference a product version.

7.1.1 Data model

A product (`OscratProduct`) captures identity (name, acronym, owning organization) and CRA posture: a product type (IoT devices, embedded software, remote data processing, and so on), the CRA classification (default, important class I or II, or critical), the applicable conformity procedure (from self-assessment up to EUCC certification), a compliance status progressing from not-assessed through in-progress to a final outcome, a risk level, an active/inactive flag, and links to the reporting organizations assigned to it.

A version (`OscratProductVersion`) records the version identifier, release and support-end dates, and its lifecycle state, from draft through active and supported to deprecated, archived, or withdrawn (full state lists in Appendix B). Each version holds two dedicated attachment slots: the Conformity Assessment Report (CAR) and the Declaration of Conformity (DoC).

7.1.2 API and UI

Routes: `/api/teams/[slug]/products` (list/create), `/products/search`, `/products/[productId]`, `/products/[productId]/versions`, `/versions/[versionId]`, plus the version sub-resources listed in section 4.1. The version details page is the daily working surface: `TABS_CONFIG` (`components/oscrat/versions/versionDetails/tabs/tabs.tsx`) renders the tabs for compliance, task, files, documentation, sbom, scans, vulnerabilities, configuration, incidents, repository, and audit-log.

7.1.3 CRA applicability check (public)

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

The applicability check runs without an account at `/form` (`pages/form/index.tsx` → `components/craForm/`). The question bank is static JSON (`components/craForm/data.json`): a sequence of applicability questions, each with remark, hint, CRA references, and answer options that can be eliminatory or skip ahead, plus a risk question whose options carry a risk level. Logic in `utils/craForm.ts` and `hooks/useCraForm.ts` handles eliminatory answers and skip chains, and the highest-risk answer given determines the resulting `OscratProductCategory`.

State persists in browser `localStorage` (key `craFormState`), so an anonymous visitor can complete the check, sign up, and continue: `pages/organization/[slug]/products/add-product/cache` re-hydrates the stored answers, creates the product with the derived category plus an initial version, saves the answers as an `OscratAssessment` of type `CRA`, and clears the cache. `CraViewModal` replays stored answers against the question bank for later review.

7.2 Dashboard

The organization dashboard (`pages/organization/[slug]/dashboard.tsx`) is the landing view after sign-in. It composes three blocks: `CompletedAppCheck` (state of the CRA applicability check, read from `localStorage`, linking into cached product creation), `RecentActivities` (the most recent product/version-scoped audit-log entries via `useRecentActivities`: date, product, version, user, action), and the compliance snapshot built from `useComplianceData` and `useLatestAssessment`, with PDF export. The server-side summary, `getTeamDashboardSummary` (`packages/model/operations/dashboard.ts`), returns counts of total products, products with in-progress compliance, active products, products with a withdrawn version, open vulnerabilities, open incidents, and SBOM reports. Charts are Chart.js 4 via react-chartjs-2, with colors drawn from the shared chart tokens.

7.3 Repositories and SBOM Generation

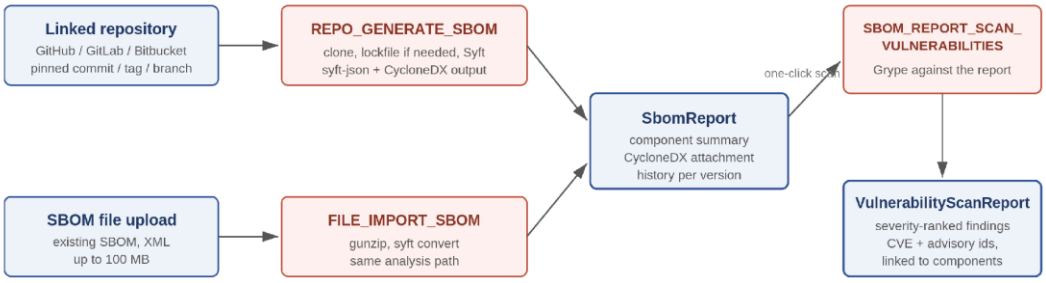

Figure 4. SBOM and vulnerability pipeline (sections 7.3 and 7.4): generation, import, and one-click scanning.

7.3.1 Repository linking

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

Each product version can link exactly one repository (`OscratRepository`, `versionId` unique): a provider (`GITHUB | GITLAB | BITBUCKET`), the repository URL, and a pin to `targetBranch`, `targetTag`, or `targetCommit` (checkout priority: commit, then tag, then branch). Public repositories need no credentials (`authType: PUBLIC`); private ones use a personal access token (`authType: PERSONAL_ACCESS_TOKEN`).

Tokens are encrypted at rest with `cryptr` (AES-256-GCM with a PBKDF2-derived key seeded from `TOKEN_ENCRYPTION_KEY`) in `packages/model/operations/encryption.ts`. API reads return a masked placeholder, never the value. The jobrunner decrypts only at clone time (`apps/jobrunner/src/utils/git.ts`), constructing provider-specific auth URLs (`x-access-token:` for GitHub, `oauth2:` for GitLab, `x-token-auth:` for Bitbucket).

7.3.2 SBOM generation job (`REPO_GENERATE_SBOM`)

`jobs/sbom.ts` runs inside a temp directory. It clones the repository at the pinned ref, then runs `generateLockFileIfNeeded`: if a `package.json` exists with no lockfile, it executes `npm install --package-lock-only` so exact dependency versions are captured. Syft then runs with dual output (`syft-json` and `cyclonedx-xml`). `analyzeSBOM` builds the stored summary (total components, package-type breakdown, package list), which lands in `SbomReport.sbomData`; the CycloneDX XML is stored as an attachment.

7.3.3 SBOM import (`FILE_IMPORT_SBOM`)

Existing SBOM files are uploaded through `.../sbom-reports/import` (formidable, body parser off, 100 MB cap, XML only), gzip-stored, and processed by `jobs/sbomImport.ts`: gunzip, `syft convert` to syft-json, then the same analysis path. Every SBOM report is kept per version with history, and any report can be sent to vulnerability scanning with one click (`.../vulnerability-scan-reports/scan-sbom`).

7.4 Vulnerability Management

7.4.1 Automated scanning (Grype)

Two job types scan for vulnerabilities: `REPO_SCAN_VULNERABILITIES` (clone, lockfile generation, then `grype dir:` over the working tree) and `SBOM_REPORT_SCAN_VULNERABILITIES` (`grype sbom:` against a stored SBOM report, generated or imported). Grype runs with `GRYPE_DB_AUTO_UPDATE=false` and `GRYPE_DB_VALIDATE_AGE=true`; the runner refreshes the Grype database at startup and every 24 hours, deferring the refresh until no jobs are running.

`analyzeGrypeResults` produces the stored summary: total count, per-severity counts (critical/high/medium/low/negligible), Grype version, and the finding list. Each finding carries the advisory id, the CVE (resolved from related vulnerabilities when the primary id is a GHSA or similar), severity, the package and version that introduced it, and the fixed-in version; every finding is therefore traceable to the component that introduced it. Raw results are zipped and

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

stored with the report (`VulnerabilityScanReport`), which back-links its source SBOM report.

7.4.2 Manual tracking

Vulnerabilities can also be tracked manually per version (`OscratProductVulnerability`): name and description, severity, CVE reference, advisory id, affected vendor, external references, date of discovery, affected member states, and attachments for remediation plans and evidence. Each record moves through the CRA handling workflow: pending, preparation, receipt, verification, remediation development, release, and post-release (Appendix B). Dedicated add/edit pages and a details page drive the transitions, and advisories are kept distinct from vulnerabilities throughout.

7.5 Incident Management

Incidents are recorded per product version (`OscratProductIncident`) with the fields CRA reporting expects: a classification (general, confidentiality, integrity, availability, access control, vulnerabilities, technical failure, or theft/loss), an attack type (denial of service, unauthorised access, malware, abuse, other), severity, the affected asset, detection and handling dates, description and scope, root cause, corrective and preventive actions, a suspected-unlawful-act flag with description, and a cross-border-impact flag with details. The table is indexed on status, severity, and detection date for the overview queries.

Each incident moves through its own lifecycle from pending through declared, active, and resolved to completed (Appendix B). New incidents start as pending; records are locked against edits once completed. Forensic evidence attaches directly to the incident through the unified attachment system. Reporting organizations (CSIRTs and authorities, with name, acronym, and reporting email) are modeled separately (`OscratReportingOrganization`) and assigned per product. Open incidents surface on the product and version overview pages.

7.6 Compliance Assessment and Reporting

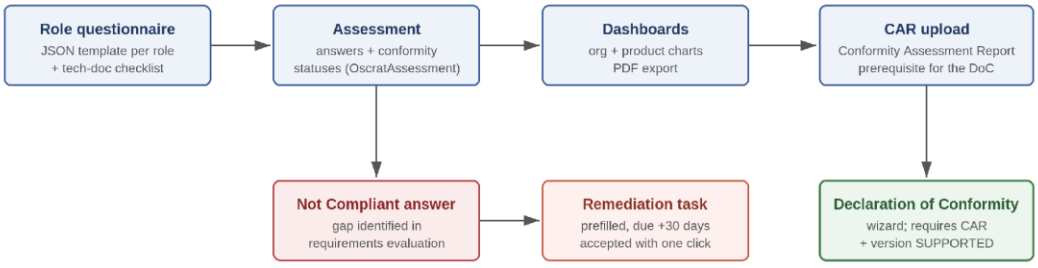

Figure 5. Compliance chain: questionnaire to remediation tasks, and CAR to Declaration of Conformity.

7.6.1 Assessment model and templates

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

Assessments are stored in `OscratAssessment`: `type` (`CRA` for applicability answers, `ORG` for organization-level process assessments, `COMPLIANCE` for product-version assessments), `schemaVersion` (currently 1.0.0), the full answer set as `rawData Json`, and optional `productId`/`versionId` (null for organization-level).

Questionnaire content lives in JSON templates separate from application logic, under `lib/compliance/assessments/team/` and `.../version/`, one file per role: manufacturer, SME manufacturer, importer, distributor, and authorized representative (role enum `OscratOrganizationRole`; `DATA_STEWARD` is the SME-manufacturer role in the UI). A template is an array of areas, each containing requirements with `reqId`, requirement text, `craReference` (e.g. "Article 19 (1)", "Annex VII"), a `genericTask` used to prefill remediation work, hints, and typed questions (boolean/text, optionally with required file-upload evidence). `complianceDataManager.getComplianceData(role)` loads the role's template server-side.

7.6.2 Technical Documentation Checklist

The checklist template (`tech-doc-checklist.json`) is appended as an additional area for manufacturers, importers, and distributors, and as an optional area, excluded from the result, for authorized representatives. Unlike the questionnaires it is a checklist: each entry pairs a documentation requirement with its regulatory reference and a compliant/non-compliant status, and the UI deep-links the CRA annexes on EUR-Lex. Conformity statuses across all assessments: Fully compliant, Partially compliant, Not Compliant, Not Applicable, In Evaluation, Not Evaluated.

7.6.3 Translations

Template values are i18n keys with English defaults shipped in `locales/en/compliance-*.json`. The export pipeline produces translation templates per namespace (`GET /api/teams/[slug]/compliance-translation-template?namespace=...`); uploaded translations are stored as `TeamData` rows keyed `compliance:translation:{type}-{role}:{lang}`, with a custom-override fallback chain in `lib/compliance/translations.ts`. The UI lists the EU languages; English ships at release.

7.6.4 From gaps to work, and to the DoC

Requirements evaluation turns gaps into tasks: when an answer is rated Not Compliant, `useComplianceTaskGeneration` proposes a prefilled task (title from the requirement, description from `genericTask`, due in 30 days, `originType: AUTOMATIC`), which the user accepts with one click. Dashboards at organization and product level chart the results (Chart.js). The documentation chain ends on the version: a Conformity Assessment Report as the prerequisite (generated or uploaded, see below), then the Declaration of Conformity (`.../doc`; PDF only). The DoC wizard (`useDocWizard`, with Simple and Full templates) requires the CAR to be present and the version to be in `SUPPORTED` status.

7.6.5 How the CAR is built

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

The Conformity Assessment Report has two sources that converge on the same endpoint. When a version assessment reaches completion, `ComplianceForm` calls `generateAndUploadCAR`: it renders the assessment to PDF via `exportComplianceToPDF` (`CompliancePDFExport.tsx`, @react-pdf/renderer), names the file from the product, and POSTs it to `.../versions/[versionId]/car`, where it becomes the version's CAR attachment. Alternatively, an externally produced report (PDF/XLS/XLSX) can be uploaded manually through the same endpoint, which upserts the attachment in place. The DoC gate reads whichever CAR is set.

The generated PDF contains an overview page (product, organization, generation date, overall progress, evaluation-status and conformity charts, summary counts), a requirements table (id, requirement, area, status, conformity), and one detail page per assessed questionnaire requirement with its CRA reference, hint, each question's answer, additional information, and an evidence-attached flag. The same export is available as a manual download from the compliance tab and the dashboard. PDF labels come from `lib/compliance/pdfTranslations.ts`; requirement and question text is resolved through the compliance translator.

7.6.6 Changing and expanding the questions

The templates hold i18n keys, not text, so the three kinds of change touch different files:

- **Editing question text.** Change the value in the English locale file (`locales/en/compliance-*.json`); the template JSON stays untouched. Team-uploaded translations override these values per language at render time.
- **Adding a question or requirement.** Append to the area in the template JSON under `lib/compliance/assessments/{team,version}/`, following the id conventions (`reqId` prefixed per area, `questionId` as `<reqId>-Q<n>`), and add the matching keys (`area-N.<reqId>.requirement`, `.hint`, `.questions.Q<n>.text`) to the English locale file. The translation-export endpoint serves the English locale file verbatim, so new keys become translatable automatically.
- **Adding an area.** New object with a unique numeric `id` (100 is reserved for the tech-doc checklist) in the team and/or version template, plus its `area-N.*` locale keys. The `optional` and `areaType` fields control how the UI and the compliance result treat it.
- **Adding a role.** Requires the enum value in `OscratOrganizationRole` (Prisma), template files in both directories, and registration in every role-to-file map: `complianceDataManager.ts`, `lib/compliance/translations.ts`, `constants/complianceTemplates.ts`, and `getRoleForTeam` in `lib/compliance/utils.ts`, plus the locale namespace.

Two things to know before editing. First, always consult the role-to-file mapping in `complianceDataManager.ts` rather than trusting file names: the file a role loads is defined by that map, and the names do not correspond one-to-one to the roles. Second, there is no template migration: `schemaVersion` is recorded on every save

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

but not compared on read, and stored answers are keyed by `requirementId/questionId`. Adding a question lowers the affected requirement's completion percentage on existing assessments until it is re-answered; renaming or removing ids leaves stored answers orphaned. Existing assessments are handled by updating them in place or by resetting (`resetAssessment`, which deletes and restarts).

7.7 Task Management

Tasks are the platform's unit of work. Each `Task` gets a per-organization task number (allocated from a counter on the team), a title and description (with optional localization ids so system-generated tasks render in the user's language), a to-do / planned / in-progress / done lifecycle, a due date, an author and assignee, optional links to a product and version, an origin marker distinguishing manual from automatically created tasks, a task type classifying the work (training, vulnerability, incident, SBOM, documentation, risk, configuration management, requirements, or other) plus comments, attachments, and two-way links to documentation.

The `properties Json` column makes the single task model extensible. Typed keys (packages/model/types/task.ts) cover configuration-scan references (configuration_report_id, configuration_rule_id, configuration_cce, configuration_severity, configuration_rule_title, configuration_rule_description), attached cybersecurity controls (csc_controls, with an embedded csc_audit_logs trail), and the risk-assessment payload (section 7.8)

ISO-mapped controls: catalogs ship as JSON (`ISO-CSC-controls-2013.json`, `ISO-CSC-controls-2022.json`, `CSF2_1.json`, `MVPS-controls.json`); the organization picks its ISO baseline via `/api/teams/[slug]/csc/iso`, and controls attach to a task through `PUT /tasks/[taskNumber]/csc` with add/remove/change operations. Automatic task creation comes from three sources: compliance requirements evaluation (section 7.6), configuration-scan findings (section 7.9), and awareness training (section 7.10). Routes: `/api/teams/[slug]/tasks` and `/tasks/[taskNumber]/{comments, attachments, csc, documentation}`.

7.8 Risk Assessment

Risk assessments are a task type, stored entirely in the task properties behind an `enableRiskAssessment` flag; there is no separate model. Stage one records the threat, the affected security categories (confidentiality, integrity, availability), likelihood, impact, and an owner; exposure is computed from a 3×3 risk matrix (low/medium/high on each axis) defined in `utils/riskCalculation.ts`. Stage two is locked until stage one is saved and records the treatment decision (accept, reduce, avoid, or transfer), mitigation measures, residual exposure, and the responsible party.

Both stages are validated by Yup schemas (lib/validation/risk.ts) and rendered as two Formik forms in RiskAssessmentSection.tsx; asset context comes from the linked product. A risk task cannot be set to DONE until both stages are

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

stored; the API rejects the update otherwise. Because a risk assessment is a task, it follows the normal task lifecycle, appears in the shared task list, and inherits comments, attachments, and audit logging; exposure scores drive prioritization.

7.9 Configuration Management

The platform processes OpenSCAP results rather than scanning machines itself. Users upload scan artifacts in ARF, XCCDF, or OVAL format to `.../configuration-scan-reports/import` (formidable, 100 MB, XML only). The format is detected from the file header (`detectConfigurationScanFormat`); accepted versions are ARF 1.1, XCCDF 1.2, and OVAL 5.x. The XML is gzipped, stored as a `File`, and a single transaction creates the `ConfigurationScanReport`, the `PROCESS_CONFIGURATION_SCAN` job (the payload carries only the file id), and the audit entry.

The job (`jobs/configurationScan.ts`) gunzips the artifact and renders a read-only HTML report: through the `oscap-report` CLI for ARF/XCCDF, or an in-house OVAL renderer (`ovalHtmlReport`), since oscap-report cannot render OVAL. Parsing uses `fast-xml-parser`; the HTML is zipped and attached, and the raw input file is deleted afterwards. The stored `ConfigurationScanSummary` holds pass/fail/error counts, profile and benchmark metadata, target hostname, scan date, and per-rule results (`ruleId`, title, CCE id, severity, result, description, references), sorted by result priority then severity.

When the job finishes it creates one task of type CONFIGURATION_MANAGEMENT for each failed rule, due in 30 days, carrying the rule reference and severity in the task properties and created with originType: AUTOMATIC. Rules that already have an open task are skipped, so re-processing a scan does not duplicate work. A task can also be created by hand from the report. A JSON-path query over properties.configuration_rule_id back-links existing tasks to rules, so the report shows "view task" instead of offering a duplicate. Past reports are retained per version for tracking configuration posture over time.

7.10 Security Awareness Training

Training has no dedicated model; it is built on Task (**type TRAINING**) and TeamMember.lastAwarenessTrainingCompletion.
ensureAwarenessTrainingTask is idempotent: it first looks for an active training task for the user, and otherwise creates a TODO task assigned to the member, due in 30 days, with localized title/description and originType: AUTOMATIC. It fires on every membership path (team creation, invitation acceptance, SSO just-in-time login, and SCIM provisioning), so the program runs regardless of how a user was added.

Completion and renewal: when a training task is set to DONE, the update handler stamps lastAwarenessTrainingCompletion on the membership and creates the next training task for the same member, due 365 days later.

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

7.11 Documentation Module

The documentation module stores versioned markdown documents (`Documentation` model): `slug` (unique per team), `title`, markdown `content` (edited with MDXEditor), `visibility` (`PRIVATE` | `PUBLIC`), `status` (`DRAFT` | `PUBLISHED` | `ARCHIVED`), an integer `version`, and scoping to the organization, a product, or a version via optional `productId`/`versionId`. All operations run in transactions with audit context. Edits are rejected on archived documents, and the version counter increments only on substantive change (content or title), not on visibility or status flips. Title uniqueness is enforced per scope, and ownership assertions verify that the referenced product/version belongs to the team.

Public serving: unauthenticated routes under `/api/teams/[slug]/public/...` return only `PUBLISHED` + `PUBLIC` documents with a whitelisted response payload, rendered on public pages with a dedicated layout and shareable URL. Documents and tasks link both ways through the `DocumentationTask` junction, so a document can point at the work it supports and vice versa.

7.12 Audit Logging

Audit logging is a custom system built around the `AuditLog` model (the earlier Retraced integration is not used). Each entry stores the actor (`userId`, `userType` of `USER` | `API_KEY` | `SYSTEM`, with denormalized `userName`/`userEmail` so identities resolve after account changes), the `action` string (`entity.verb`), a `crud` flag, the target (`targetType`/`targetId`/`targetName`), the team, denormalized product/version ids and names, `metadata Json`, `ipAddress`, `userAgent`, and timestamp. Indexes support the common query patterns, led by team plus creation date.

Transactional writes: `createAuditContextWithTx` writes the log row on the caller's Prisma transaction client, so the audit entry commits or rolls back atomically with the mutation it records; the trail cannot diverge from actual state. Before/after storage: creates snapshot the tracked fields; updates compute an RFC 6902 JSON Patch (`rfc6902`) over the tracked-field subset and enrich each operation with the previous value, so the UI renders edits as old and new side by side; no-op updates are skipped. Coverage spans every audited entity type (products, versions, vulnerabilities, incidents, repositories, tasks, SBOM and scan reports, files, teams, members, invitations, SSO connections, directory sync, webhooks, API keys, assessments, attachments, and documentation), each with an explicit tracked-fields whitelist so secrets and blobs never enter the log. Report and file downloads are logged as read events.

Context flows in through the middleware: `withTeamAuth` attaches `req.auditInfo` (user, team, product/version from the route), which handlers pass to model operations. Querying: `POST /api/teams/[slug]/audit-logs` (gated by the `team_audit_log` resource, i.e. owners and admins) with filters for page/pageSize (capped at 100), action, target type/id, user, product, version, crud flag, and date range; `/audit-logs/filter-options` supplies the distinct users and target types for the filter UI.

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

7.13 Organizations and Access

The organization is the `Team` model (team and organization were unified in one migration). Beyond identity (`name`, unique `slug`, `domain`, `defaultRole`, per-team `taskIndex`), it carries the CRA-relevant data: the organization's roles under the act (manufacturer, importer, distributor, authorized representative, SME manufacturer), its legal form (from natural person to public body, including open-source foundation and steward), size classification (micro, small, medium enterprise), tax id, and postal and contact details. Product acronyms and version support-end dates live on the respective models for reporting. Sign-up captures these organizational details up front.

Membership and invitations: `TeamMember` links users with a role (section 3.5) and the training completion stamp. Members join by invitation: unique token with expiry, one invitation per team and email, delivered by the dedicated mail service (nodemailer + React Email templates) as `${APP_URL}/invitations/{token}`. The same mail service covers welcome, verification, password-reset, and email-change flows. Magic-link login is off by default in favor of the standard sign-in flow.

Enterprise features, individually gated by `FEATURE_TEAM_*` flags: SAML SSO and SCIM directory sync (section 3.3); outbound webhooks via Svix (per-team application, lazily created, no-op when `SVIX_API_KEY` is unset) for invitation/member/team events; and API keys (`ApiKey` model storing only a hashed key, with `expiresAt` and `lastUsedAt`) managed under team settings. Login attempts are rate limited (section 3.1).

8. Operations Runbook

8.1 Local development

- Prerequisites: Node 22.14 (`.nvmrc`), pnpm, Docker.
- `pnpm dev:setup`: install, start Postgres (docker-compose), run `prisma migrate dev`, seed.
- Dev server on port 4002; database `postgres:15` on 5432.
- Jobrunner development additionally requires `syft`, `grype`, and `oscap-report` on `PATH`; the runner refuses to start without them.

8.2 CI and deployment

- CI is a single GitHub Actions workflow running `pnpm check-all` (lint, typecheck, build). There is no automated test suite in CI.
- The development deployment runs on Railway (section 2.6); production hosting is the operating team's choice, following the deployment model in section 2.6. In the Railway setup, database migrations run automatically as the

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

frontend's pre-deploy command; replicate that ordering (`prisma migrate deploy` before rollout) on any other target.

- Health monitoring: `GET /api/health` verifies database connectivity; Railway restarts on failure (max 10).

8.3 Job operations

- Failed jobs remain `FAILED` with `errMessage`/`errCode` persisted; there is no automatic retry. Re-trigger from the UI (a new job) after fixing the cause.
- Tune throughput with `MAX_CONCURRENT_JOBS` (default 3) and `JOB_POLL_INTERVAL_MS` (default 5000).
- The Grype DB updates daily inside the runner; no external action needed.

8.4 Database

- Schema changes follow the standard Prisma flow: edit `schema.prisma` in `packages/model`, run `prisma migrate dev` locally, commit the migration; production applies it on the next deploy.
- All file content lives in Postgres; size backups accordingly. Scan artifacts are the main growth driver (compressed, but retained per version for history).

9. Third-Party Services

Accounts and services the operating team inherits, with their configuration:

Service	Purpose	Configuration
SMTP provider	Transactional email (invitations, verification, resets)	`SMTP_HOST/PORT/USER/PASSWORD/FROM`
Svix	Outbound webhooks (optional; no-op when unset)	`SVIX_API_KEY`
Google reCAPTCHA	Bot protection on login/signup/reset (optional)	reCAPTCHA site/secret keys
Matomo	Product analytics	Matomo URL/site id
GitHub / Google OAuth	Social login (only if enabled in `AUTH_PROVIDERS`)	OAuth app client id/secret
BoxyHQ SAML Jackson	SSO/SCIM, embedded in-process; no external account	Uses main `DATABASE_URL`

The frontend also references dicebear (avatar images, allowed in CSP `img-src`) and includes the Mixpanel and OpenAI client libraries as dependencies; verify which are active in your deployment before rotating credentials.

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

10. Extension Points

Common extensions and where to implement them:

- **New background job type.** Add the value to the `WorkerJobType` enum (Prisma), define a Yup payload schema in `packages/model/schemas/jobPayloads.ts`, implement the handler in `apps/jobrunner/src/jobs/`, register it in the runner dispatch, and create jobs through a model operation that opens the transaction (job, report, and audit together, following `createConfigurationScanReportWithJob`).
- **New questionnaire role or content.** Add or edit the JSON template under `lib/compliance/assessments/{team,version}/`, add the English locale namespace under `locales/en/`, and register the role-to-file mapping in `complianceDataManager.ts`, `lib/compliance/translations.ts`, and `constants/complianceTemplates.ts`. Bump `schemaVersion` if the answer shape changes.
- **New translation.** Export the namespace template via the compliance-translation endpoint, translate, and upload. Translations are stored per team in `TeamData`; no deploy needed.
- **New attachment target.** Add the foreign key to `Attachment`, extend `attachmentTeamScope` with the new relation path (critical: this is the tenant-isolation boundary), and extend `getAttachmentLinkedEntity` for audit metadata.
- **New audited entity.** Add the `EntityType` and its `TRACKED_FIELDS` whitelist in `packages/model/audit/`, and perform mutations through `createAuditContextWithTx` on the operation's transaction.
- **New API resource.** Follow the per-method guard pattern (section 4.2): a Yup schema in `lib/validation/`, an endpoint module in `lib/api/endpoints/`, query keys in `queryKeys.ts`, react-query hooks, and a feature hook in `hooks/`.
- **New task-based workflow**. Follow the risk/training pattern: a taskType column and typed keys in the task properties (typed in packages/model/types/task.ts) rather than a new model, inheriting lifecycle, comments, attachments, and audit.

11. Appendix

A. Background job types

Job type	Trigger	What it does
`REPO_GENERATE_SBOM`	SBOM tab → generate	Clone pinned ref, generate lockfile if needed, Syft → syft-json + CycloneDX
`FILE_IMPORT_SBOM`	SBOM import upload	Gunzip upload, `syft convert`, analyze and store

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

Job type	Trigger	What it does
`REPO_SCAN_VULNERABILITIES`	Scan repository	Clone + `grype dir:` scan, severity-classified findings
`SBOM_REPORT_SCAN_VULNERABILITIES`	One-click scan of an SBOM report	`grype sbom:` against the stored report
`PROCESS_CONFIGURATION_SCAN`	OpenSCAP artifact upload	Render ARF/XCCDF/OVAL to HTML report, per-rule results, remediation tasks

One additional 24-hour timer runs inside the runner (not a queue job): the Grype database refresh.

B. Key lifecycle enums

Entity	Lifecycle
Product version	`DRAFT → ACTIVE → SUPPORTED → DEPRECATED → ARCHIVED / WITHDRAWN`
Product compliance	`NOT_ASSESSED → IN_PROGRESS → COMPLIANT / NON_COMPLIANT / PENDING_CERTIFICATION / CERTIFIED`
Vulnerability handling	`PENDING → PREPARATION → RECEIPT → VERIFICATION → REMEDIATION_DEVELOPMENT → RELEASE → POST_RELEASE`
Incident	`PENDING → START → DECLARED → STABLE → ACTIVE → RESOLVED → COMPLETED`
Task	`TODO → PLANNED → IN_PROGRESS → DONE`
Worker job	`PENDING → IN_PROGRESS → COMPLETED / FAILED / CANCELLED`
Documentation	`DRAFT → PUBLISHED → ARCHIVED`

C. Principal environment variables

Variable	Purpose
`DATABASE_URL`	PostgreSQL connection (shared by frontend, jobrunner, and embedded SAML Jackson)
`APP_URL`	Canonical application URL (cookies, email links, SAML issuer)
`AUTH_PROVIDERS`	Comma-separated enabled providers (`credentials`, `github`, `google`, `saml`, `email`)
`TOKEN_ENCRYPTION_KEY`	Key for repository access-token encryption (cryptr / AES-256-GCM)

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

Variable	Purpose
`CONFIRM_EMAIL`	Require email verification before credentials sign-in
`DISABLE_NON_BUSINESS_EMAIL_SIGNUP`	Block signups from consumer email domains
`SMTP_HOST / PORT / USER / PASSWORD / FROM`	Mail service
`SVIX_API_KEY`	Webhook delivery (optional; feature no-ops when unset)
`FEATURE_TEAM_SSO / DSYNC / AUDIT_LOG / WEBHOOK / API_KEY`	Enterprise feature flags (404-gate their routes)
`GROUP_PREFIX`	IdP group prefix granting ADMIN on SSO login (default `boxyhq-`)
`MAX_CONCURRENT_JOBS`	Jobrunner concurrency bound (default 3)
`JOB_POLL_INTERVAL_MS`	Queue polling interval (default 5000)
`JOBRUNNER_WORKSPACE_ROOT`	Scratch directory for clones and scans (default `/tmp/jobrunner-workspace`)
`CONFIGURATION_SCAN_AUTO_TASKS`	Create a configuration-management task for each failed rule after a scan (default true)

The full list with sample values is in `.env.example` at the repository root.

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the granting authority can be held responsible for them. The project funded under Grant Agreement No. 101190180 is supported by the European Cybersecurity Competence Centre.

OSCRAT

Open-Source Cyber Resilience Act Tools

D 3.8 - Full Product Documentation

www.oscrat.eu https://www.linkedin.com/c