PERFORMANCE MANAGEMENT SYSTEM · V1.0.0

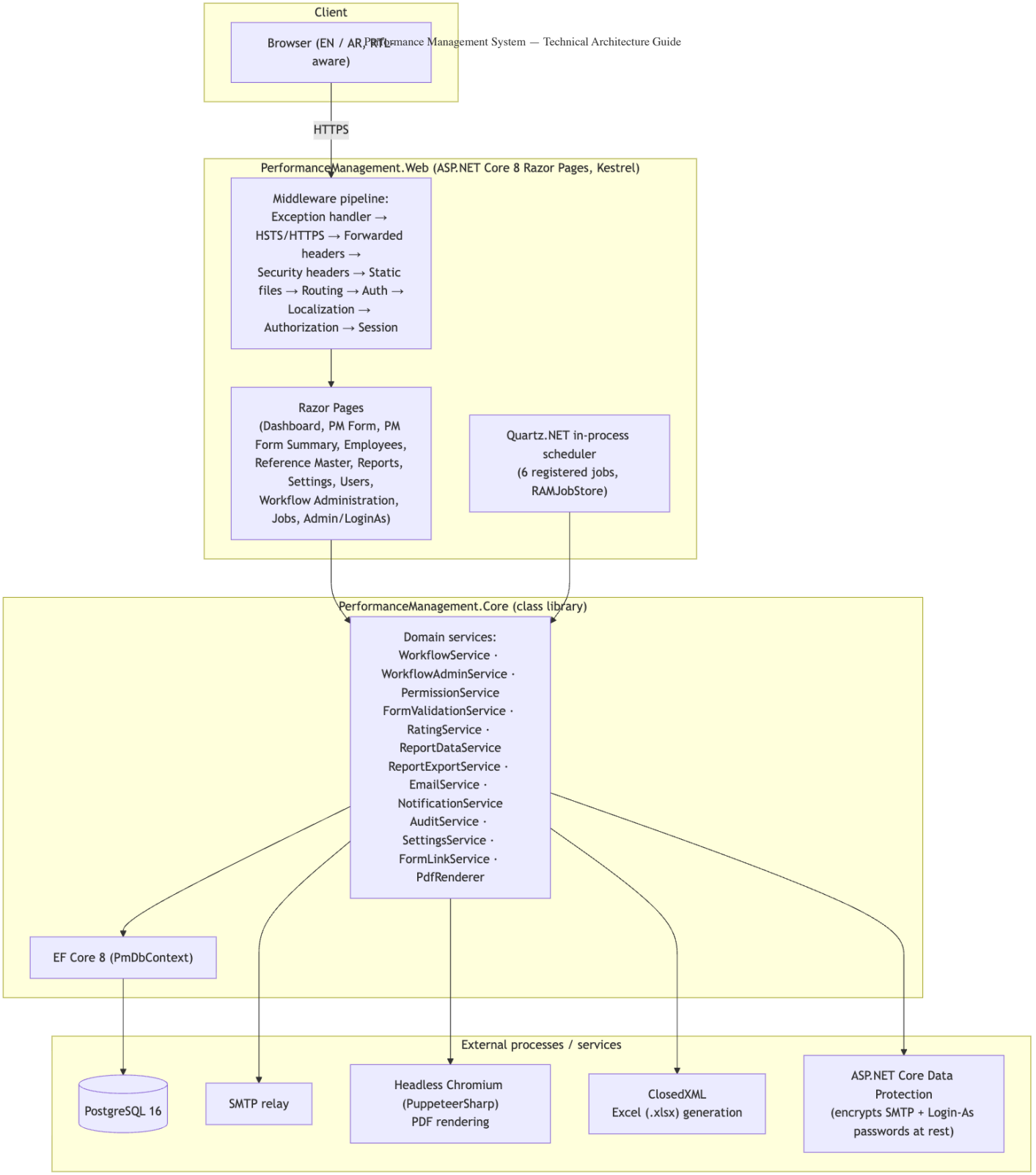
Technical Architecture Guide

Solution architecture, layer responsibilities, the workflow/reporting/email engines, security, performance, and known trade-offs — for engineers and architects.

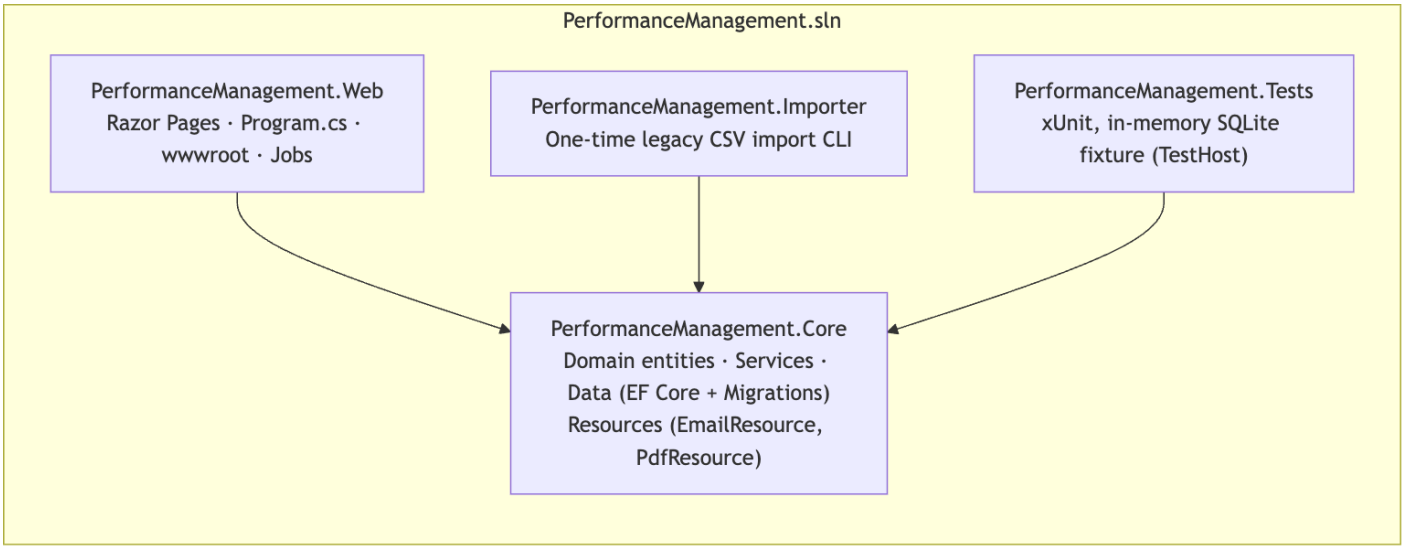
Contents

1. Solution architecture

A conventional three-tier Razor Pages application: server-rendered UI, a class-library business layer, and PostgreSQL. There is no separate API layer or SPA front end — Razor Pages posts back to the server, and the small amount of client-side JavaScript (a searchable-combobox widget) is progressive enhancement over plain HTML forms, not a client-side application framework.


Client
Browser (EN / AR, RTL aware)
HTTPS
PerformanceManagement.Web (ASP.NET Core 8 Razor Pages, Kestrel)
Middleware pipeline: Exception handler → HSTS/HTTPS → Forwarded headers → Security headers → Static files → Routing → Auth → Localization → Authorization → Session
Razor Pages (Dashboard, PM Form, PM Form Summary, Employees, Reference Master, Reports, Settings, Users, Workflow Administration, Jobs, Admin/LoginAs)
Quartz.NET in-process scheduler (6 registered jobs, RAMJobStore)
PerformanceManagement.Core (class library)
Domain services: WorkflowService · WorkflowAdminService · PermissionService FormValidationService · RatingService · ReportDataService ReportExportService · EmailService · NotificationService AuditService · SettingsService · FormLinkService · PdfRenderer
EF Core 8 (PmDbContext)
External processes / services
PostgreSQL 16
SMTP relay
Headless Chromium (PuppeteerSharp) PDF rendering
ClosedXML Excel (.xlsx) generation
ASP.NET Core Data Protection (encrypts SMTP + Login-As passwords at rest)

2. Project structure & dependency flow



PROJECT	CONTAINS
`PerformanceManagement.Web`	Razor Pages, `Program.cs` (startup/middleware pipeline), Quartz job registrations, static assets, per-page localization resources.
`PerformanceManagement.Core`	Domain entities, `PmDbContext` + EF Core migrations, and every business service (see below).
`PerformanceManagement.Importer`	One-time CLI for the legacy CSV data migration — not part of normal runtime operation.
`PerformanceManagement.Tests`	xUnit suite against an in-memory SQLite fixture, exercising `Core` directly.

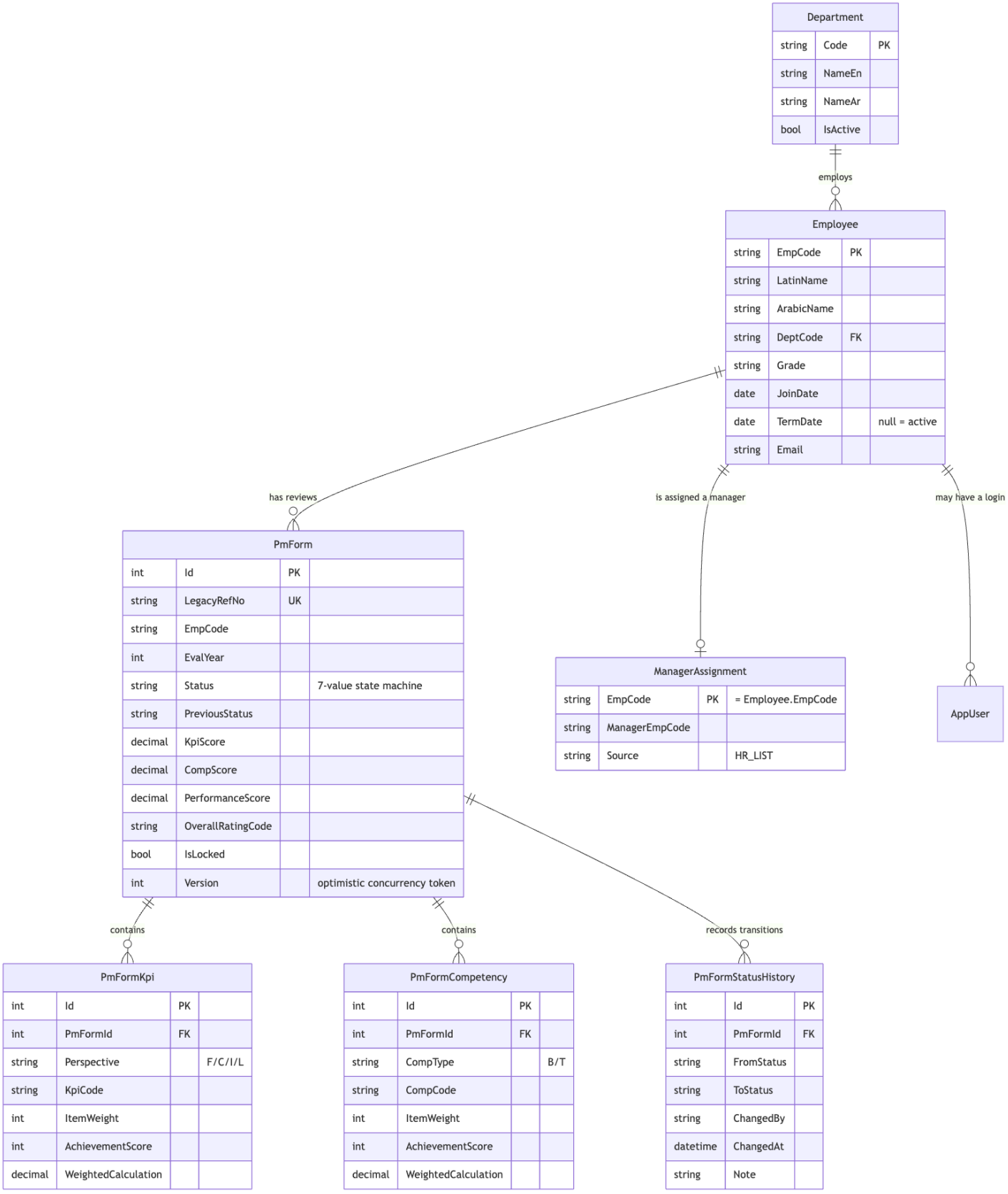
Dependencies are strictly one-directional: `Web` and `Importer` depend on `Core`; `Core` depends on neither. This keeps every business rule (workflow transitions, validation, scoring, permissions) testable without booting a web host, and reusable by both the interactive app and the one-time import tool.

Core services

SERVICE	RESPONSIBILITY
`WorkflowService`	The seven-status state machine — every normal transition, each wrapped in its own database transaction with optimistic-concurrency handling.
`WorkflowAdminService`	Search/read-model assembly for Workflow Administration, plus the six recovery actions (delegates the actual mutation to `WorkflowService`, then writes an audit entry).
`PermissionService`	Resolves "can this user act on this specific employee's form" — self, direct manager, HR admin, branch viewer, self-managed exceptions.
`FormValidationService`	Weight-completeness and achievement-completeness rules, shared by the normal submit path and the Administrative Completion override.
`RatingService` / `ScoringService`	Weighted-score calculation and rating-band resolution.
`ReportDataService` / `ReportExportService`	Report data assembly and PDF/Excel rendering.
`EmailService`	Deduplicated, idempotent, optionally-redirected SMTP dispatch with a full delivery log.
`NotificationService`	In-app notification-centre records.
`AuditService`	Append-only admin action trail, gated by a settings toggle (with one unconditional bypass reserved for logging the toggle itself).
`SettingsService`	The singleton `SystemSettings` row — encrypts SMTP/verification passwords via Data Protection.
`FormLinkService`	Signs and verifies the tokenized deep links used in every workflow email.
`PdfRenderer`	Headless-Chromium HTML-to-PDF rendering (PuppeteerSharp).

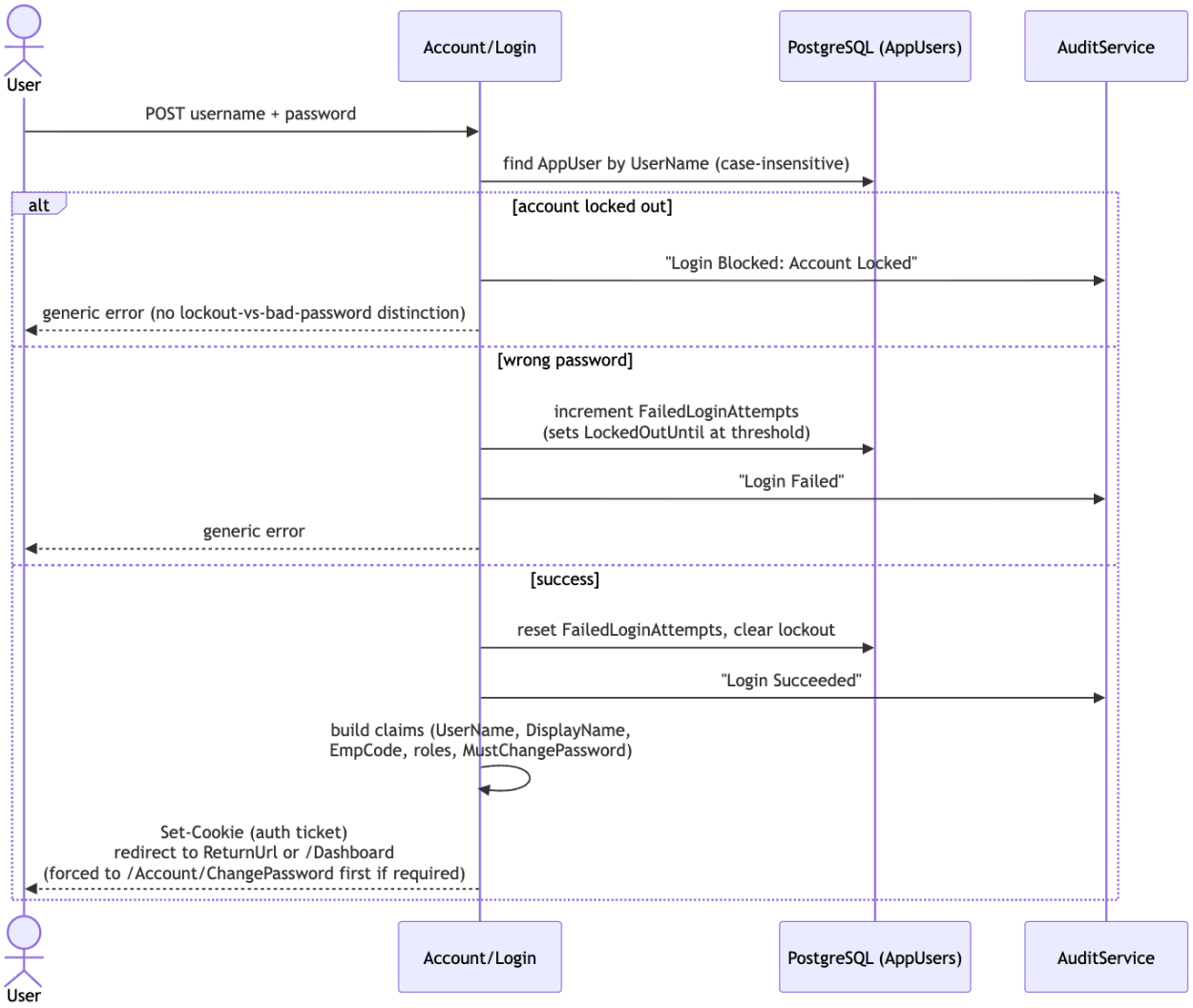
3. Database design

PostgreSQL 16 via EF Core 8, 22 tables, 11 migrations. Full ER diagrams and column-level detail are in the **Database Guide**; summary here:

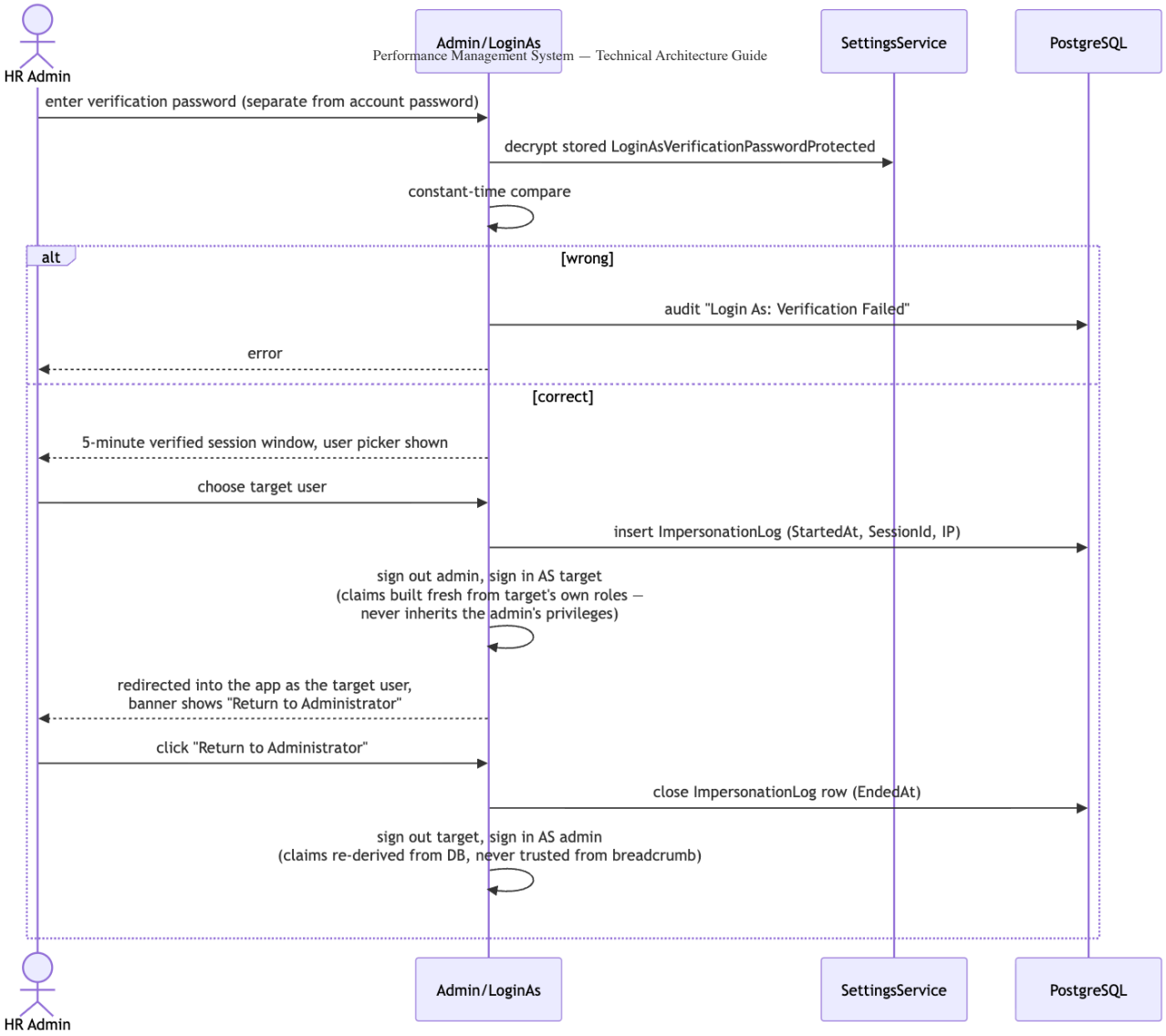


Optimistic concurrency is enforced via a `Version` column configured as an EF Core concurrency token on `PmForm` — two simultaneous writers to the same review will not silently overwrite each other; the loser receives a "changed by another user, please retry" result.

4. Authentication & authorization flow

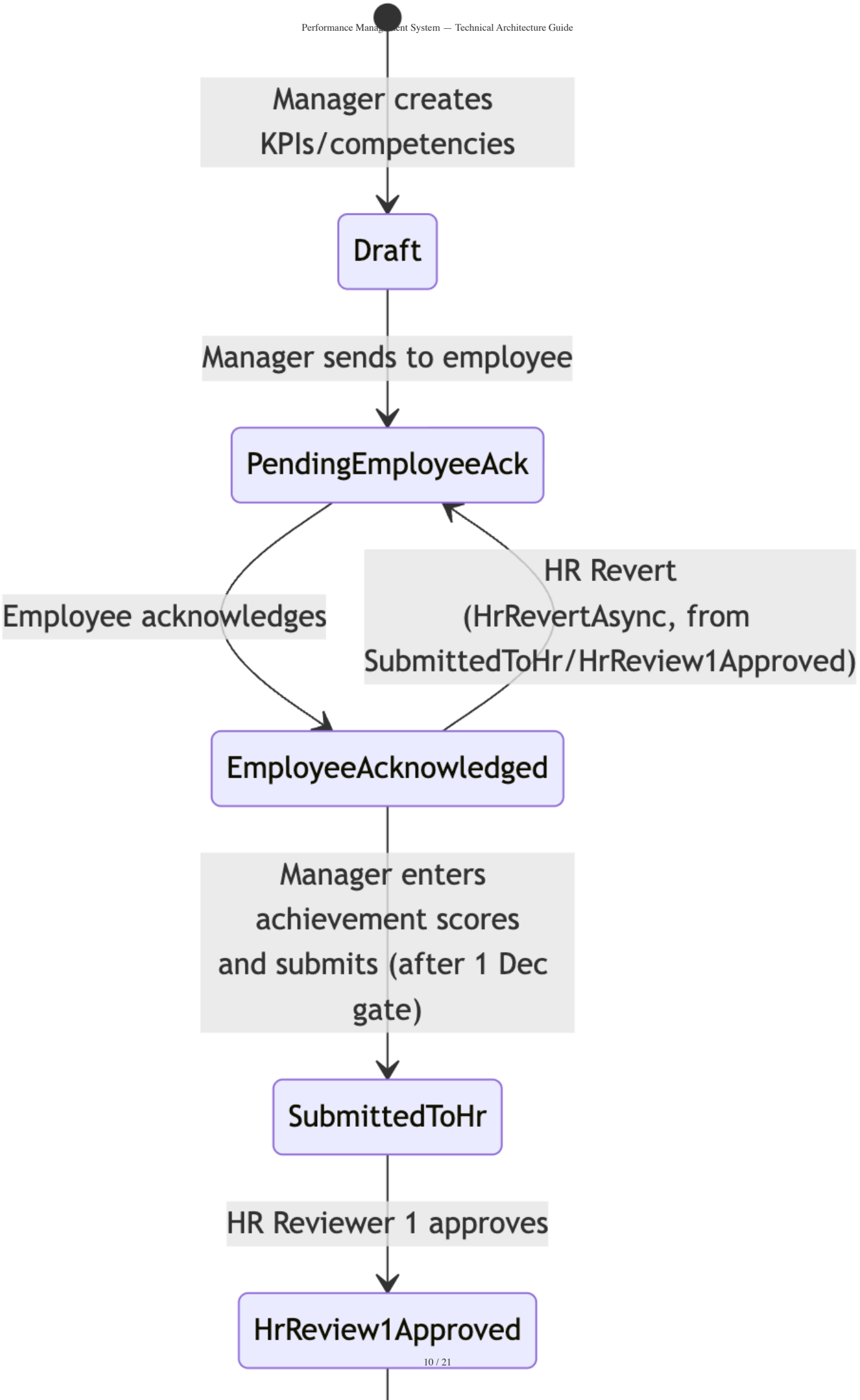


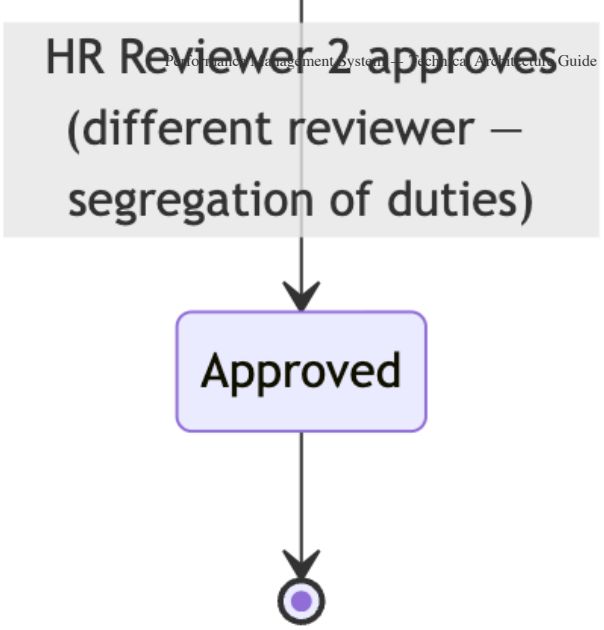
Cookie authentication with a global fallback policy (`RequireAuthenticatedUser`) applied to every page not explicitly marked `[AllowAnonymous]` . Two roles exist in the claims model — `HR_ADMIN` and `VIEWER` — layered with a second, data-driven authorization check on the PM Form itself (`PermissionService`), since "can I act on this specific record" depends on the manager-assignment table, not a static role.



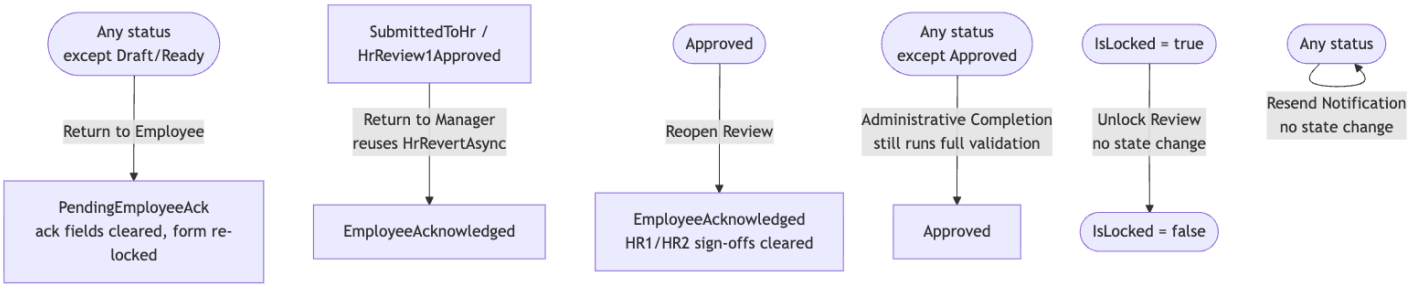
The "Login As" impersonation feature builds the impersonated session's claims fresh from the target user's own roles — it never inherits or extends the administrator's privileges — and "Return to Administrator" re-derives the admin's claims from the database rather than trusting anything carried in the impersonated session's own claims.

5. Workflow engine


Manager creates KPIs/competencies
Draft
Manager sends to employee
PendingEmployeeAck
Employee acknowledges
HR Revert (HrRevertAsync, from SubmittedToHr/HrReview1Approved)
EmployeeAcknowledged
Manager enters achievement scores and submits (after 1 Dec gate)
SubmittedToHr
HR Reviewer 1 approves
HrReview1Approved

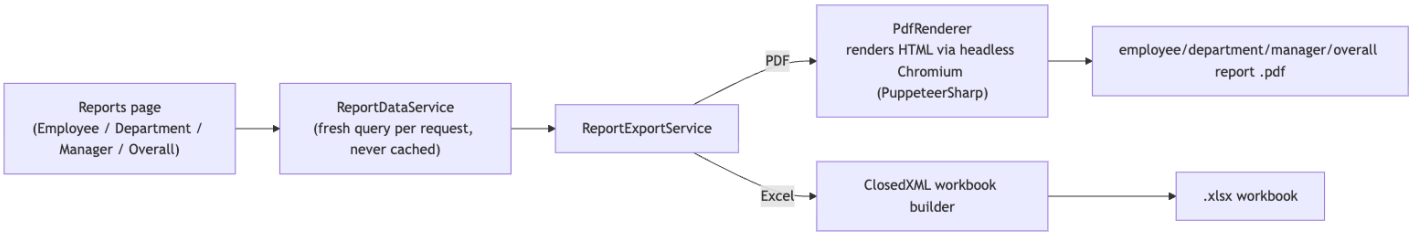


Every transition method in `WorkflowService` follows the same shape: open a transaction, re-read and validate the current status inside that transaction (defends against a stale read from before the transaction started), apply the mutation, commit, then dispatch exactly one email *after* the commit — so a slow or failed SMTP send can never roll back or block a state change that has already legitimately happened. Failures are logged to the `EmailLog` table rather than thrown.



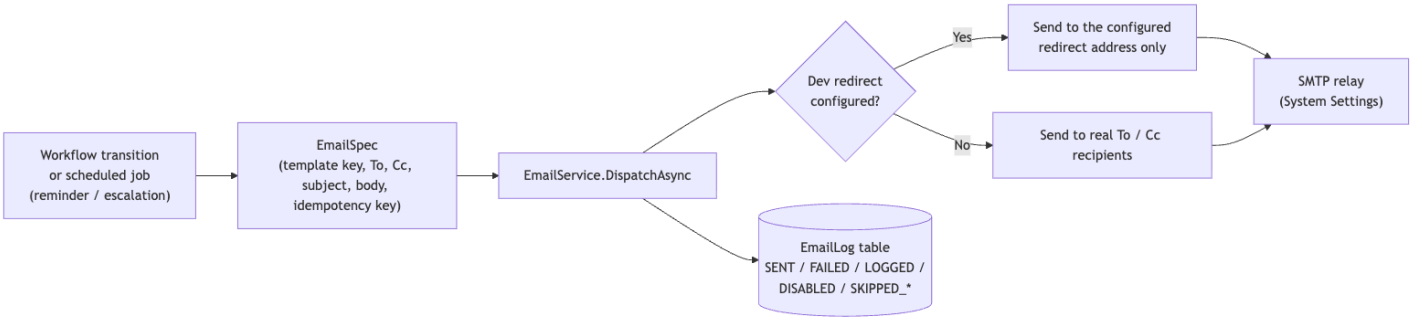
The six Workflow Administration overrides reuse this same transactional machinery (one even delegates directly to the existing `HrRevertAsync` transition rather than duplicating its logic), adding only a post-commit audit-log write tagged with an "Workflow Administration: " action-source prefix.

6. Reporting engine



`ReportDataService` queries are deliberately uncached — every export re-reads the database at request time, trading a small amount of latency for a strict guarantee that a report never shows stale data. PDF rendering uses the same headless-Chromium engine that renders any website, which is what makes correct Arabic/RTL PDF output straightforward — the HTML report template is simply given `dir="rtl"` and the browser's own bidi engine does the rest, rather than a from-scratch PDF-layout library having to reimplement bidi text shaping.

7. Email pipeline



Recipient deduplication (To wins over Cc), an idempotency key (`template:formRef:version`) that prevents a duplicate send for the same transition being retried, and an *opt-in* development-redirect address for rehearsing against real data without emailing real employees. With no redirect configured — the normal production state — mail is delivered to the real To/Cc recipients.

8. Localization architecture

ASP.NET Core's built-in localization middleware, with one `.resx` / `.ar.resx` pair per Razor Page (view-level strings via `IViewLocalizer`) and, where a page's code-behind needs its own strings, a matching `{PageModel}.resx` pair (via `IStringLocalizer<T>`). A small set of cross-page strings live in one shared `SharedResource.resx`. Culture is resolved per-request by a custom provider that falls back, in order, to: the signed-in user's saved preference, a culture cookie, then the organization default — so language follows the account across devices, not just the browser.

9. Audit system

An append-only `AuditLog` table, written through one service method, gated by a `SystemSettings.EnableAuditLogging` flag — with a single, deliberate exception: the write that logs the toggle being changed uses an unconditional bypass, so disabling auditing is itself always recorded and can never erase evidence of its own occurrence. A separate, never-gated `ImpersonationLog` table independently tracks every "Login As" session.

10. Scheduler

Quartz.NET, in-process, with the default in-memory `RAMJobStore` — a deliberate choice for a single-node deployment (see §12). Six jobs are registered by name in a static job registry so the Job Management page can trigger/inspect them without depending on the scheduler's own runtime state surviving a restart; actual run history is persisted in a dedicated `ScheduledJobRun` table instead.

11. Security posture

- Account lockout applies uniformly, including to administrator accounts.
- Generic authentication error messages prevent username enumeration and lockout-state leakage.
- HTTPS redirection, HSTS, and `Secure` / `SameSite` cookies in Production (relaxed only in Development, where the local dev server is plain HTTP).
- Baseline security response headers (`X-Content-Type-Options` , `X-Frame-Options: DENY` , `Referrer-Policy` , a Content-Security-Policy).
- SMTP and Login-As verification passwords encrypted at rest via ASP.NET Core Data Protection.
- A startup guard refuses to boot in Production with any shipped default credential unchanged.

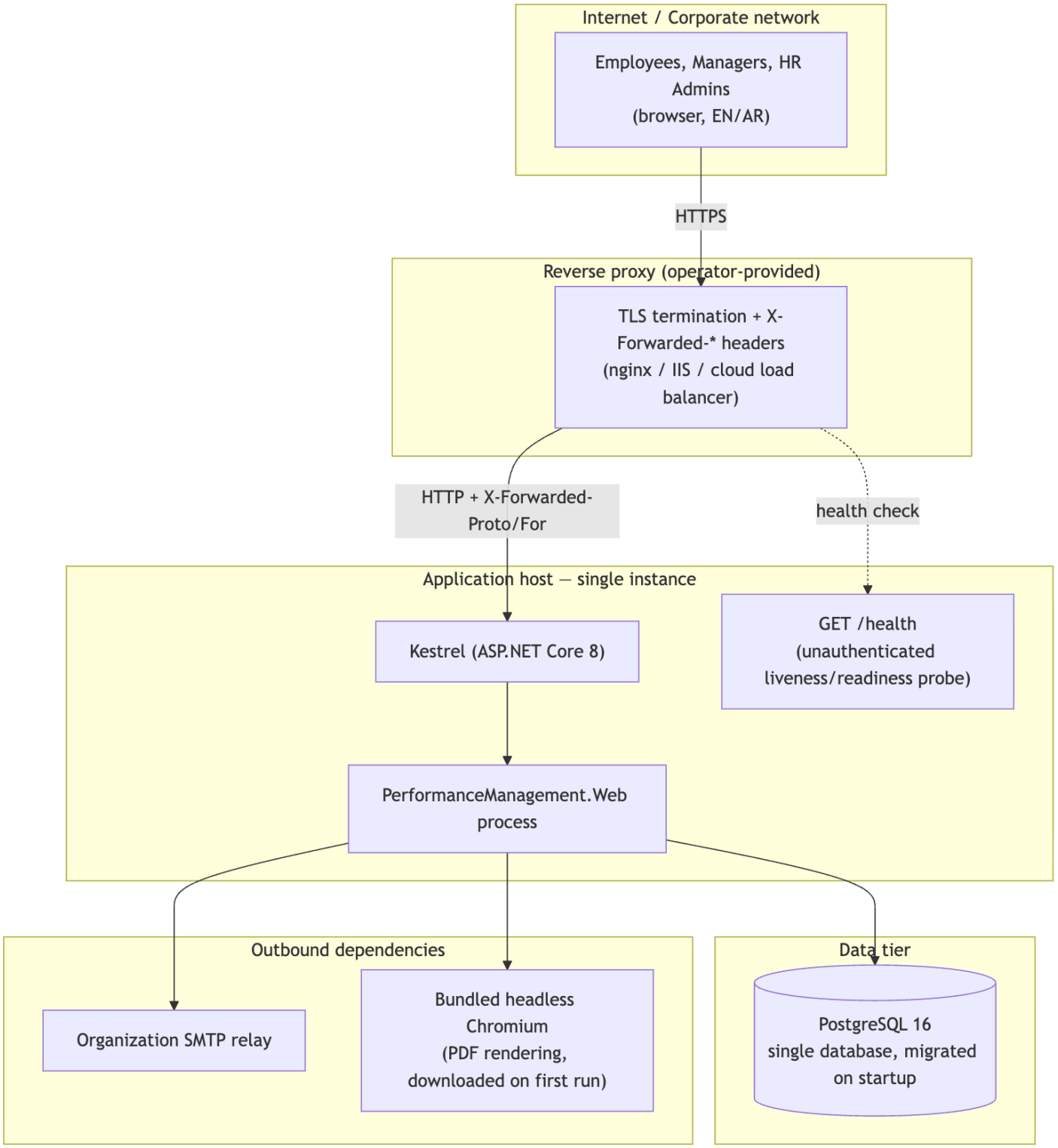
12. Performance & scalability

Read/search pages use `AsNoTracking()` queries and server-side pagination (50–200 rows per page); indexes exist on every column actually filtered by a search page (`PmForm(EvalYear, Status)` , `AuditLog(EntityType, EntityId)` , `ManagerAssignment.ManagerEmpCode` , among others). A 60-second command timeout guards against a hung connection blocking a request indefinitely.

KNOWN LIMIT

Automatic EF Core connection-retry-on-failure is intentionally not enabled: several `WorkflowService` methods manage their own explicit database transactions, and EF's retrying execution strategy rejects user-initiated transactions unless every such call site is first rewritten to run inside `CreateExecutionStrategy().ExecuteAsync(...)` . Tracked as a roadmap item.

13. Deployment



See the Deployment Guide for the full checklist. In short: single application instance, PostgreSQL as the only stateful dependency, migrations applied automatically on startup, an unauthenticated `/health` endpoint for load-balancer probes.

14. Design decisions & known trade-offs

DECISION	WHY
Snapshot fields on `PmForm` (name, department, grade at time of review)	A review must always read back the way it looked when reviewed, even if the employee is later renamed or transfers — intentional, not accidental denormalization.
Single wide `SystemSettings` row rather than one table per category	Every setting is read together on nearly every request; explicitly documented as a deliberate trade-off in code, not an oversight.
Server-rendered Razor Pages, no SPA framework	The workflow is fundamentally a sequence of forms and approvals — a full client-side framework would add complexity without a corresponding UX benefit here.
In-process Quartz scheduler	Matches the single-node deployment target; explicitly documented as not safe to scale horizontally without further work.
Headless-Chromium PDF rendering instead of a native .NET PDF library	Correct Arabic/RTL bidi text layout comes for free from the browser engine, at the cost of bundling a Chromium binary.

15. Future extensibility

The service-layer separation (all business logic in `Core`, referenced but never duplicated by `Web`) means a future API layer, a mobile client, or additional report types could be added by calling the same services rather than re-implementing workflow/permission/scoring logic. The audit and notification infrastructure is already generic enough (entity type/id, free-text details) to extend to new event types without a schema change.