LCL Environnement — Documentation technique

Systèmes informatiques (Scoro, ODK Central, rapports, chatbot, LabTracker, ScoroPost)

LCL Environnement

None

Table of contents

1. Vue d'ensemble

Télécharger toute la documentation en PDF

Tout l'écosystème informatique de LCL Environnement roule sur **un seul serveur** (droplet DigitalOcean), géré par **Coolify** et exposé sous le nom de domaine `lclchantiers.com`. Cinq projets s'y déploient, chacun avec son propre dépôt GitHub. Voir Infrastructure pour le détail du serveur, du domaine et de Coolify.

Le fil conducteur des données relie ces projets entre eux : **Scoro** (gestion de projets) sait quels chantiers existent ; **ODK Central** récolte ce qui se passe sur le terrain ; les autres applications transforment ces données en documents et tableaux de bord utilisables.

```
flowchart LR
    Scoro([Scoro<br/>gestion de projets])
    Bridge[ScoroOdkBridge]
    ODK([ODK Central<br/>collecte terrain])
    Rapports[lcl-rapports]
    LabTracker[LabTracker]
    ScoroPost[ScoroPost]

    Scoro -->|projets récents<br/>CM-/CMC-, 20 min| Bridge
    Bridge -->|crée des entities| ODK
    ODK -->|soumissions terrain| Rapports
    ODK -->|échantillons + devis| LabTracker
    LabTracker -->|dépôt des rapports labo| Rapports

    Scoro -.->|projets fermés<br/>ENV/GEO| ScoroPost
    ODK -.->|formulaires de<br/>satisfaction| ScoroPost
```

Les traits pleins suivent le chemin principal des données (Scoro → pont → ODK Central → applications de suivi/rapports). Les traits pointillés montrent **ScoroPost**, qui interroge Scoro et ODK Central directement pour son propre usage (évaluations de fermeture de projet), sans passer par le pont.

Projet	Rôle en une phrase	Détail
Pont Scoro → ODK	Recopie automatiquement les nouveaux projets Scoro vers ODK Central	→
Rapports de chantier	Transforme les soumissions ODK en PDF classés par chantier, alerte sur les non-conformités	→
Suivi labo (LabTracker)	Suit chaque échantillon au laboratoire, calcule et vérifie la conformité des essais	→
Chatbot technique	Répond aux questions techniques des techniciens à partir des normes de génie civil	→
Évaluations de fermeture (ScoroPost)	Envoie un questionnaire de satisfaction à la fermeture d'un projet et compile les résultats	→

Pour le vocabulaire (ODK, Coolify, infrastructure), voir le Glossaire.

2. Infrastructure

Résumé des sections 1 à 4 du *Guide des systèmes informatiques : LCL Environnement* (C. Bolduc, été 2026). Aucune connaissance technique n'est nécessaire pour lire cette page.

2.1 Vue d'ensemble en une image

Tout roule sur **un seul ordinateur loué chez DigitalOcean**, allumé 24 h/24. Plusieurs logiciels y tournent en même temps, chacun dans sa « boîte » séparée. **Coolify** joue le rôle de gestionnaire d'immeuble : il installe les boîtes, les redémarre, les met à jour et dirige les visiteurs vers la bonne porte.

```
flowchart TB
    Internet --> Domaine["lclchantiers.com<br/>(nom de domaine)"]
    Domaine --> Droplet["Droplet DigitalOcean"]
    subgraph Droplet["Droplet DigitalOcean"]
        Coolify["Coolify (le gestionnaire)"]
        Coolify --> ODK["ODK Central"]
        Coolify --> Rapports["Application rapports"]
        Coolify --> LabTracker["LabTracker"]
        Coolify --> Autres["(autres projets)"]
    end
```

2.2 Le serveur : le « droplet » DigitalOcean

Un **droplet** est un ordinateur loué au mois chez DigitalOcean. Il n'a ni écran ni clavier : on s'y connecte à distance. Il roule Linux (Ubuntu). C'est ce qui permet à nos applications d'être accessibles en tout temps, sans dépendre d'un ordinateur au bureau.

- La facture est **mensuelle** ; si elle n'est pas payée, tout tombe.
- L'accès se fait via le compte DigitalOcean de l'entreprise : quelqu'un chez LCL doit en conserver les identifiants.
- DigitalOcean peut prendre des **snapshots** (photos complètes du serveur) : c'est le filet de sécurité pour revenir en arrière en cas de pépin.

> ⚠️ **Un seul point de panne**
>
> Tout est sur une seule machine. Si le droplet tombe, **tous les services tombent en même temps**. C'est acceptable pour notre taille, mais ça veut dire que les sauvegardes sont critiques.

2.3 Le nom de domaine : lclchantiers.com

Un nom de domaine est une adresse lisible par des humains : sans lui, il faudrait taper l'adresse IP du serveur. On utilise des **sous-domaines**, comme les numéros de porte d'un même immeuble :

Adresse	Ce qu'on y trouve
`odk.lclchantiers.com`	ODK Central : gestion des formulaires terrain
`rapports.lclchantiers.com`	Application de génération de rapports PDF
`echantillons.lclchantiers.com`	LabTracker : suivi des échantillons au labo
(voir les pages projets)	Les autres services

Chaque sous-domaine est enregistré dans le **DNS**, l'annuaire mondial qui dit « `odk.lclchantiers.com` , c'est cet ordinateur-là ». Cet annuaire se configure chez le fournisseur où `lclchantiers.com` a été acheté.

Pour ajouter un nouveau service accessible sur Internet, il faut :

1. Créer une entrée DNS (un enregistrement **A**) qui pointe le sous-domaine vers l'adresse du droplet.
2. Indiquer ce sous-domaine dans Coolify pour le service concerné.

2.4 Coolify : le gestionnaire des applications

Coolify installe, démarre, arrête et met à jour toutes nos applications, via une interface web, l'équivalent gratuit et auto-hébergé de services comme Heroku ou Vercel. Avant, chaque application était installée « à la main » directement sur le serveur : fragile, et seule la personne qui l'avait montée pouvait s'y retrouver. Avec Coolify, une mise à jour se résume à pousser le code sur GitHub, puis cliquer sur **Redeploy** (ou rien du tout si l'automatique est activé).

2.4.1 Les concepts de Coolify

Concept	Ce que c'est
Projet	Un regroupement logique (ex. tout ce qui touche LabTracker)
Application (ressource)	Un logiciel qui roule, relié à un dépôt GitHub
Base de données	Coolify peut aussi héberger les bases PostgreSQL des applications
Déploiement	Mettre en ligne la dernière version du code ; historique consultable, retour en arrière possible
Variables d'environnement	Les réglages et **mots de passe**, stockés dans Coolify et jamais dans le code, pour que le code reste public sur GitHub sans exposer de secret

2.4.2 Caddy : le portier

Coolify utilise **Caddy** comme réceptionniste : quand quelqu'un tape `rapports.lclchantiers.com`, c'est Caddy qui reçoit la visite et la redirige vers la bonne application. Caddy gère aussi automatiquement les **certificats HTTPS** (le cadenas du navigateur), renouvelés sans intervention.

2.4.3 Le quotidien avec Coolify

Je veux…	Comment faire
Voir si un service roule	Ouvrir Coolify, regarder le statut de l'application (vert = OK)
Comprendre pourquoi ça plante	Onglet **Logs** de l'application
Mettre à jour une application	Pousser le code sur GitHub, puis **Redeploy**
Annuler une mise à jour ratée	Historique des déploiements → redéployer la version précédente
Changer un mot de passe / un réglage	Onglet **Environment Variables**, puis redéployer

> ⚠ **Warning**
>
> Modifier une variable d'environnement ne prend effet **qu'après un redéploiement**.

3. Standards pour les développeurs

Conventions de travail communes aux 5 dépôts (`ScoroOdkBridge`, `lcl-rapports`, `lcl-chatbot`, `lcl-labtracker`, `ScoroPost`) et à ce dépôt de documentation. Le but : que n'importe qui reconnaisse à quoi sert une branche juste en lisant son nom, peu importe le dépôt.

3.1 Nommage des branches

Trois niveaux, du plus stable au plus court terme :

Branche	Créée à partir de	Rôle
`main`	–	Code en production. Ne reçoit que des fusions depuis `dev`, jamais de commit direct.
`dev`	`main`	Intégration. Les branches de travail se fusionnent ici en premier ; c'est ce qui est validé avant d'aller en production (voir Infrastructure et la règle *tester sur dev avant prod* du `CLAUDE.md` du dossier parent).
`<type>/KAN-XXX-description-courte`	`dev`	Une branche par ticket. Fusionnée dans `dev`, puis supprimée une fois fusionnée.

```
gitGraph
    commit id: "release"
    branch dev
    checkout dev
    commit id: "intégration"
    branch feature/KAN-123-nouveau-rapport
    checkout feature/KAN-123-nouveau-rapport
    commit id: "travail"
    checkout dev
    merge feature/KAN-123-nouveau-rapport
    checkout main
    merge dev
```

> ⚠️ **`dev` (branche Git) ≠ `dev` (environnement Coolify)**
>
> Ne pas confondre les deux : la branche Git `dev` décrite ici existe dans les 5 dépôts. L'environnement Coolify `lcl-chatbot-dev` (voir Infrastructure) est un déploiement séparé propre à `lcl-chatbot`, qui tourne habituellement à partir de cette branche `dev` mais reste un concept distinct (c'est un serveur qui roule le code, pas une branche du dépôt).

3.1.1 Format d'une branche de travail

```
<type>/KAN-XXX-description-courte
```

- **`KAN-XXX`** : le numéro du ticket (ex. `KAN-142`), pour retrouver facilement de quoi la branche parle.
- **`description-courte`** : quelques mots en minuscules séparés par des tirets, qui résument le ticket (pas obligé de recopier le titre exact).
- **`<type>`** : ce que fait la branche. Les plus utilisés :

Type	Utilisation
`feature/`	Nouvelle fonctionnalité
`fix/`	Correction de bug
`chore/`	Tâche sans impact fonctionnel (dépendances, config, ménage)
`docs/`	Documentation seulement (ex. ce dépôt)

Exemples :

```
feature/KAN-118-alerte-non-conformite
fix/KAN-142-proctor-correction
docs/KAN-97-ajout-standards-dev
```

3.2 Fusion (merge)

Deux fusions distinctes, chacune avec sa propre validation — ne pas sauter d'étape même si le ticket semble simple.

3.2.1 1. Branche de travail → `dev` (aller)

1. Créer la branche à partir de `dev` (pas de `main`).
2. Développer, committer.
3. Avant d'ouvrir la Pull Request, valider **localement** (voir le tableau ci-dessous selon le dépôt).
4. Ouvrir la PR **vers `dev`**, jamais directement vers `main`.
5. Une fois fusionnée, supprimer la branche de travail.

3.2.2 2. `dev` → `main` (chemin retour, mise en production)

C'est l'étape qui applique la règle *tester sur dev avant prod* : `dev` accumule plusieurs branches de travail fusionnées, donc revalider l'ensemble avant de le promouvoir en production, pas seulement le dernier changement.

1. S'assurer que `dev` est à jour et build/tourne sans erreur.
2. Revalider (voir tableau) — en particulier si plusieurs tickets se sont accumulés dans `dev` depuis la dernière mise en production.
3. Ouvrir la PR **`dev`** → **`main`**.
4. Après la fusion, redéployer dans Coolify (voir Infrastructure) et vérifier les **logs** de l'application pendant quelques minutes.
5. En cas de problème en production, revenir à la version précédente via l'historique des déploiements Coolify plutôt que de pousser un correctif dans l'urgence.

3.2.3 Quoi valider, selon le dépôt

Dépôt	Avant de fusionner (aller et retour)
`lcl-labtracker`	Les **78 tests automatisés** doivent passer — calculs certifiés (béton, granulats, Proctor), aucune erreur tolérée.
`lcl-rapports`	Si le changement touche la mise en page d'un rapport : comparer les PDF générés aux **113 PDF de référence**, image par image.
`lcl-chatbot`	Valider sur l'environnement Coolify `lcl-chatbot-dev` avant de fusionner vers `main` / `lcl-chatbot-prod`. Si des PDF de normes ont changé, relancer l'ingestion (rien ne se propage automatiquement) et surveiller les crédits OpenAI/Anthropic avant des tests en masse.
`ScoroOdkBridge`	Vérifier manuellement qu'aucune fiche ODK en double n'est créée (le fichier de suivi sur volume persistant fait foi) et que le compte de service ODK a toujours ses droits.
`ScoroPost`	S'assurer que `MAIL_REDIRECT_TO` et `ODK_FAKE_LINKS` sont dans l'état voulu avant tout envoi réel (retirés en prod réelle, actifs en test).
`lcl-docs` (ce dépôt)	`.venv\Scripts\python -m mkdocs build` sans erreur/lien cassé, et vérifier visuellement la page modifiée.

Note

Cette page documente une convention d'équipe, pas une contrainte imposée par un outil (aucune protection de branche automatisée n'est en place pour l'instant `[à confirmer]`). Elle vaut ce que vaut son respect par tout le monde.

4. Projets

4.1 ScoroOdkBridge

Service d'arrière-plan (.NET 10 Worker Service, sans UI ni endpoint HTTP) qui fait le pont entre **Scoro** (gestion de projets) et **ODK Central** (collecte de données terrain) pour LCL Environnement.

À intervalle régulier, le service interroge l'API Scoro pour les projets récemment modifiés, filtre ceux qui concernent le contrôle des matériaux, et crée une **entity** correspondante dans un dataset ODK Central, pour que les équipes de terrain puissent remplir des formulaires ODK rattachés à ces projets. Le pont ne fait que **créer** des entities ; il ne les modifie ni ne les supprime jamais.

Ce dépôt fait partie de l'infrastructure LCL Environnement décrite dans l'Infrastructure (un seul droplet, Coolify, Caddy, `lclchantiers.com`). Ce service tourne en arrière-plan uniquement : il n'a pas de sous-domaine.

4.1.1 Comment ça marche

À chaque cycle (`Worker.cs`) :

1. Charge le fichier de suivi local (`ProjectTracker`) des IDs Scoro déjà traités.
2. Interroge Scoro (`ScoroClient.GetProjectsModifiedAsync`) pour les projets modifiés depuis `LookbackMinutes` minutes.
3. Filtre les projets éligibles : nom de projet préfixé `CM-` ou `CMC-` (`ProjectMapper.IsEligible`). **`CML-` est volontairement exclu** pour l'instant.
4. Pour chaque projet éligible pas encore dans le tracker, construit le payload d'entity (`ProjectMapper.ToEntity`) et le crée dans ODK Central (`OdkClient.CreateEntityAsync`).
5. Marque le projet comme traité dans le tracker dès la création réussie.
6. Une erreur ODK sur un projet est loguée et n'interrompt ni le cycle ni le projet suivant (`OdkApiException` catché par projet). Une exception inattendue interrompt le cycle en cours mais pas la boucle du worker : elle est loguée et le prochain cycle démarre normalement au tick suivant.

Le premier cycle tourne immédiatement au démarrage, puis à chaque tick d'un `PeriodicTimer` (intervalle = `PollIntervalMinutes`, 20 min par défaut).

4.1.2 Architecture

Trois dossiers indépendants, assemblés dans `Program.cs` :

```
Scoro/    client de l'API Scoro (lecture seule)
ODK/      client de l'API entities d'ODK Central (création seule)
Bridge/   la colle : filtre d'éligibilité, mapping, fichier de suivi
```

`Scoro/`

- **`ScoroClient`** : POST vers `projects/list` (paginé, `filter.modified_date`) et `projects/view/{id}`. La clé d'API Scoro est envoyée comme **champ du corps de la requête** (`apiKey`), pas comme header (voir `ScoroClient.BuildBody`).
- **`ScoroServiceCollectionExtensions`** : pipeline de résilience (`Microsoft.Extensions.Http.Resilience` / Polly) : retry sur 429/503 et erreurs de transport, jusqu'à 5 tentatives, backoff exponentiel avec jitter. Respecte le header `x-ratelimit-reset` de Scoro (ou `Retry-After`) quand présent au lieu du backoff par défaut. Timeout de 20 s par tentative.
- **`ScoroProject`** : désérialise la forme `projects/list` / `projects/view`, y compris le tableau `custom_fields`, aplati dans une map insensible à la casse (`CustomFieldsMap`) exposée via des accesseurs typés : `SiteAdress` (`c_adresse_site`), `City` (`c_ville`), `PrescribedMandate` (`c_mandat_prescrit`), `Sharepoint` (`c_sharepoint`), `Referencer` (`c_referencer`). Un

`FlexibleStringConverter` coerce les champs personnalisés numériques/booléens Scoro en `string`, et les valeurs texte sont HTML-décodées (ex. `Ayer&#039;s Cliff` → `Ayer's Cliff`).

- **`ScoroResponse<T>`** : enveloppe `status` / `statusCode` / `messages` / `data` commune aux réponses Scoro ; `EnsureOk` lève une `ScoroApiException` si `status != OK`.

`ODK/`

- **`OdkClient`** : `CreateEntityAsync` (POST `v1/projects/{ProjectId}/datasets/{DatasetName}/entities`) et `GetExistingScoroIdsAsync` (lecture OData paginée `$top` / `@odata.nextLink` sur le dataset, utilisée pour reconstruire le tracker si besoin).
- **`OdkSessionHandler`** : `DelegatingHandler` qui gère l'authentification par jeton de session : connexion email/mot de passe au premier appel, jeton mis en cache, rafraîchi 30 min avant expiration, un seul retry automatique sur 401 (jeton révoqué côté serveur).
- **`OdkServiceCollectionExtensions`** : enregistre le `HttpClient` typé avec le handler de session. **En développement uniquement** (`IHostEnvironment.IsDevelopment()`), fait confiance au certificat auto-signé d'ODK local, jamais activé en production.

`Bridge/`

- **`ProjectMapper.IsEligible`** : seule source de vérité pour savoir quels projets Scoro sont pontés.
- **`ProjectMapper.ToEntity`** : construit le payload `(label, data)` de l'entity : `scoro_id`, `project_no`, `company_name`, `address`, `city`, `prescribed_mandate`, `start_date`, `end_date`. `label` = nom du projet Scoro (affiché dans le menu déroulant du formulaire ODK).
- **`ProjectTracker`** : ensemble d'IDs Scoro déjà pontés, persisté en JSON (`data/tracking.json` par défaut, chemin configurable). Les écritures passent par un fichier temporaire puis un `File.Move` atomique pour survivre à un crash en cours d'écriture. `SeedAsync` (actuellement non appelé automatiquement) permet de reconstruire le tracker à partir des entities existantes dans ODK via `OdkClient.GetExistingScoroIdsAsync`.

Le fichier de suivi est critique

`data/tracking.json` **doit** vivre sur un volume persistant en production. S'il est perdu ou vidé, le pont va tenter de recréer en entity ODK **tous** les projets éligibles historiques au prochain cycle, créant des doublons. Ne jamais le supprimer ou le réinitialiser sans d'abord le reconstruire via `GetExistingScoroIdsAsync` / `ProjectTracker.SeedAsync`.

4.1.3 Prérequis

- .NET SDK 10
- Accès réseau à l'API Scoro (`https://lcl.scoro.com/api/v2`) et à l'instance ODK Central cible

4.1.4 Configuration

Liée via le pattern Options (`ValidateDataAnnotations().ValidateOnStart()`) sur trois sections : `Scoro`, `Odk`, `Bridge`.

Section	Clé	Requis	Défaut	Description
`Scoro`	`BaseUrl`	non	`https://lcl.scoro.com/api/v2`	URL de base de l'API Scoro
`Scoro`	`ApiKey`	**oui**	–	Clé d'API Scoro
`Scoro`	`CompanyAccountId`	**oui**	–	Identifiant de compte Scoro
`Scoro`	`Lang`	non	`fre`	Langue des réponses Scoro
`Scoro`	`PageSize`	non	`25`	Taille de page pour `projects/list`
`Odk`	`BaseUrl`	**oui**	–	URL de base d'ODK Central
`Odk`	`Email`	**oui**	–	Email du compte de service ODK
`Odk`	`Password`	**oui**	–	Mot de passe du compte de service ODK
`Odk`	`ProjectId`	non	`1`	ID du projet ODK Central cible
`Odk`	`DatasetName`	non	`scoro_projects`	Nom du dataset ODK cible
`Bridge`	`TrackingFilePath`	**oui**	`data/tracking.json`	Chemin du fichier de suivi (volume persistant en prod)
`Bridge`	`LookbackMinutes`	non	`60`	Fenêtre de recherche des projets modifiés (chevauche l'intervalle de poll par sécurité)
`Bridge`	`PollIntervalMinutes`	non	`20`	Intervalle entre deux cycles

`appsettings.json` contient la structure avec des secrets vides. `appsettings.Development.json` contient des valeurs de dev locales (URL ODK `https://localhost`). En production, **les secrets viennent des variables d'environnement Coolify**, jamais codés en dur (règle du `CLAUDE.md` racine). Le mapping standard ASP.NET Core s'applique, ex. `Scoro__ApiKey`, `Odk__Password`.

Un `UserSecretsId` est déjà configuré dans le `.csproj` : pour du dev local sans toucher `appsettings.Development.json`, `dotnet user-secrets set "Odk:Password" "..."` fonctionne aussi.

4.1.5 Lancer le projet

```
dotnet build
dotnet run                      # démarre la boucle du worker (poll toutes les PollIntervalMinutes)
dotnet run -- --selftest        # test de parsing JSON hors-ligne, aucun appel réseau, puis quitte
dotnet run -- --scoro-test      # appelle la vraie API Scoro avec les creds configurées, affiche les projets correspondants, puis quitte
```

Il n'y a pas de projet de test dédié. `Scoro/Tests/ScoroParsingSelfTest.cs` est un self-test artisanal (assertions `Check(label, bool)` affichées en PASS/FAIL sur stdout) qui valide la désérialisation JSON de `ScoroProject` /

`ScoroResponse` contre un exemple de payload `projects/list`. **Lancer `--selftest` après toute modification sous `Scoro/`** qui touche au parsing (mapping des champs personnalisés, HTML-decoding, coercion numérique→string, etc.).

4.1.6 Docker

Build multi-étapes, correspond à la production :

```
docker build .
```

`mcr.microsoft.com/dotnet/sdk:10.0` (build) → `mcr.microsoft.com/dotnet/runtime:10.0` (exécution), point d'entrée `dotnet ScoroOdkBridge.dll`. En production (Coolify), monter un volume persistant sur le chemin de `Bridge:TrackingFilePath` et fournir toutes les variables d'environnement de secrets (voir la section Configuration).

4.1.7 Problème connu : secrets commités

`appsettings.Development.json` est **suivi par git** (seul `.dockerignore` l'exclut ; `.gitignore` ne le fait pas) et contient une vraie clé d'API Scoro et un vrai mot de passe du compte de service ODK Central, en clair. Ceci viole la règle « jamais de secret dans le code » du `CLAUDE.md` racine.

Ne pas aggraver en commitant d'autres vrais secrets dans des fichiers de config. Ceci doit être traité comme une remédiation à faire, pas comme un état normal :

1. Retirer `appsettings.Development.json` du suivi git et l'ajouter à `.gitignore`.
2. Purger le secret de l'historique git.
3. Faire tourner (rotate) la clé Scoro et le mot de passe du compte de service ODK compromis.

4.1.8 Notes de collaboration

- Ne jamais dupliquer la logique déjà couverte par le fichier de suivi : il empêche les doublons et vit sur un volume persistant à ne jamais effacer sans comprendre pourquoi.
- Le pont ne traite que les projets `CM-` / `CMC-` ; `CML-` est délibérément exclu (voir `ProjectMapper.IsEligible`).
- Le pont ne fait que créer des entities ODK, jamais modifier ni supprimer.
- Se réveille toutes les `PollIntervalMinutes` (20 min par défaut).

4.2 LCL Rapports

Portail des rapports de contrôle qualité de LCL Environnement.

Les techniciens saisissent leurs relevés sur le terrain dans **ODK Central** (formulaires F1 à F7). Ce portail transforme ces soumissions en **PDF**, les classe par chantier, les rend consultables dans un site web, et **alerte par courriel** dès qu'un rapport révèle une non-conformité. Il reçoit aussi les rapports de laboratoire produits par **LabTracker**, une application distincte, et les range aux côtés des rapports de terrain.

```
flowchart TB
    Terrain["Terrain<br/>(formulaires)"] --> ODK["ODK Central"]
    ODK -->|"soumissions + entités (lecture)"| Generateur["Générateur PDF<br/>odk_reports/ (un module par rapport)"]
    Generateur -->|"écrit rapports/&lt;no projet&gt;/*.pdf"| API["API Flask"]
    LabTracker["Laboratoire (LabTracker)"] -->|POST| API
    API -->|"courriel d'alerte (non-conformités)"| Alertes["Alertes"]
    API -->|"/api/..."| Site["Site React<br/>consultation, téléchargement, réglages"]
```

4.2.1 Ce qu'il faut comprendre en premier

Il n'y a pas de base de données. Un « projet » est un dossier nommé d'après son numéro ODK (`rapports/CMC-1274-9623/`), et un rapport est un fichier PDF dedans. Tout l'état vit dans des fichiers JSON cachés à la racine du dossier des rapports :

Fichier	Rôle
`.generated.json`	Suivi incrémental : quelles soumissions sont déjà générées
`.projects.json`	Numéros de projets connus d'ODK, publiés par le générateur
`.received.json`	Index des rapports déposés par LabTracker
`.alertes.json`	Destinataires des alertes, modifiables depuis le site
`.alertes-envoyees.json`	Non-conformités déjà signalées, pour ne pas les répéter

Rien ne tourne en arrière-plan. Aucun cron, aucun minuteur. Une génération part quand quelqu'un clique sur « Générer » dans le site (ou lance la commande sur le serveur). Les alertes sont donc aussi réactives que les générations, sauf celles des dépôts LabTracker, qui partent immédiatement.

ODK est la source de vérité. Les PDF générés sont reconstructibles à volonté. Les rapports **reçus** de LabTracker, eux, ne le sont pas : effacer le dossier des rapports les perdrait définitivement.

4.2.2 Organisation du dépôt

```
backend/
  api_server.py          API Flask : consultation, génération, réglages, proxy ODK
  inbound_reports.py     Réception des rapports LabTracker (POST /api/rapports)
  odk_report_generator.py Point d'entrée du générateur (enveloppe mince)
  odk_reports/           Le générateur — voir son propre README
  Dockerfile             Image de production
  docker-compose.yml     Déploiement (utilisé par Coolify)
frontend/
  src/App.js             L'application React (fichier unique)
tests/
  golden/                113 PDF de référence, un par cas couvert
  compare_pdfs.py        Compare les PDF régénérés aux références, au pixel
docs/                    Décisions et contrats d'interface
```

Le générateur a sa propre documentation, plus détaillée : `backend/odk_reports/README.md`, à lire avant d'ajouter ou de modifier un rapport.

4.2.3 Démarrer en local

Prérequis : Python 3.12, Node 20+, et un accès ODK si tu veux générer de vrais rapports.

Backend

```
cd backend && python -m venv venv && ./venv/Scripts/pip install -r requirements.txt
```

```
cd backend && RAPPORTS_DIR=../test-output ./venv/Scripts/python api_server.py
```

L'API écoute sur le port 5050. Sous Linux ou macOS, remplacer `./venv/Scripts/` par `./venv/bin/`.

Frontend

```
cd frontend && npm install && npm start
```

Le site démarre sur le port 3000 et relaie `/api` vers le port 5050 (`proxy` dans `package.json`). L'écran de connexion demande des identifiants ODK ; sans eux, tu peux quand même exercer l'API directement.

Générer des rapports

```
python backend/odk_report_generator.py --url https://odk.lclchantiers.com --email TON_EMAIL --password 'TON_MOT_DE_PASSE' --project 1 --output ./test-output
```

`--form` limite à un formulaire, `--no-projet` à un chantier, `--force` ignore le suivi incrémental et régénère tout.

4.2.4 Tester

La suite compare les PDF régénérés aux 113 références de `tests/golden/`, en les rendant à l'image et en mesurant l'écart au pixel :

```
python backend/odk_report_generator.py --url ... --output ./test-output --force
```

```
backend/venv/Scripts/python tests/compare_pdfs.py
```

Une différence signalée est soit une régression, soit un changement voulu. Dans le second cas, mettre à jour les références **dans un commit séparé** du changement de code, pour que l'historique reste lisible.

Cette suite exige aujourd'hui un accès ODK : elle ne peut pas tourner en intégration continue. Remplacer l'appel à ODK par des données enregistrées est le prochain chantier, et il débloquerait une CI.

4.2.5 Configuration

Tout passe par des variables d'environnement ; aucun secret n'est versionné.

Variable	Rôle
`RAPPORTS_DIR`	Où vivent les PDF (défaut `/opt/odk-rapports/rapports`)
`ODK_UPSTREAM`	Où l'API joint ODK Central
`ODK_HOST_HEADER`	En-tête `Host` envoyé à ODK derrière un proxy
`ODK_GENERATOR_URL`	URL d'ODK utilisée par le générateur
`PORTAL_PUBLIC_URL`	URL publique du portail, mise dans les liens sortants
`LABTRACKER_TOKEN`	Jeton de service attendu des dépôts LabTracker
`SMTP_HOST`, `SMTP_PORT`, `SMTP_SECURITY`, `SMTP_USER`, `SMTP_PASSWORD`, `SMTP_FROM`	Envoi des alertes

Sans `SMTP_HOST`, aucune alerte n'est tentée. Sans `LABTRACKER_TOKEN`, tout dépôt est refusé. C'est délibéré : un portail mal configuré doit fermer, pas ouvrir.

4.2.6 Déployer

La production tourne sur un Droplet DigitalOcean, derrière **Caddy géré par Coolify** (voir `docs/migration-2026-05-07.md`). Le backend est conteneurisé : `backend/docker-compose.yml`, avec le dossier des rapports monté depuis l'hôte pour survivre au conteneur.

Le frontend est encore servi en fichiers statiques par un mini-nginx et demande un `npm run build` séparé. `backend/install.sh` est l'**ancien** script d'installation (venv + systemd + nginx), conservé pour mémoire mais plus utilisé : ne pas s'en servir.

4.2.7 Conventions

- **Code et commentaires en anglais**, chaînes visibles par l'utilisateur en français (étiquettes des PDF, messages de l'interface, journaux). Les noms de formulaires et de domaine (« planche de référence ») restent en français même dans un commentaire anglais.
- Un commentaire explique **pourquoi**, pas quoi : le code dit déjà quoi.
- Commits atomiques, mise à jour des PDF de référence séparée du code.

4.2.8 À savoir avant de modifier quoi que ce soit

> ⚠ **Aucune authentification**
>
> L'API n'a aucune authentification hormis le jeton des dépôts LabTracker. Consultation, génération et réglages sont ouverts à qui atteint le portail.

- **Le mot de passe ODK transite en argument de ligne de commande** vers le sous-processus du générateur, donc visible dans `ps`.
- **Les PDF sont traités comme binaires** (`.gitattributes`) : sans ça, git réécrit les fins de ligne à l'intérieur et corrompt les références.
- **Un rapport ne doit jamais lire l'horloge.** Son rendu doit dépendre de ses seules données, sinon il change tous les jours et devient intestable.

4.3 LabTracker

Tableau de bord de suivi des échantillons pour le contrôle qualité des matériaux de construction de LCL Chantiers (béton, granulats, enrobés). Il synchronise les soumissions terrain depuis ODK Central, suit chaque éprouvette à travers le laboratoire, saisit et calcule les résultats d'essais, produit les rapports de certification en PDF, et les transmet au portail central de rapports de l'entreprise.

4.3.1 Pour commencer

Le problème qu'il résout

Un technicien de chantier prélève un échantillon sur le terrain et remplit un formulaire dans ODK Collect (Android). La soumission arrive dans ODK Central. Du côté du laboratoire, il faut savoir quels échantillons sont arrivés, lesquels sont en attente, lesquels sont en retard, qui en est responsable, quels sont les résultats d'essais, et s'ils respectent le devis, avec une trace de tout ce qui a changé.

Le concept à comprendre en premier

L'unité de suivi n'est pas l'échantillon, c'est l'éprouvette.

Un échantillon est le matériau physique prélevé sur le chantier. Il est divisé en une ou plusieurs éprouvettes, et c'est l'éprouvette qui porte un statut, une priorité, une échéance, un laboratoire assigné et ses propres essais.

- **Béton** : un échantillon → plusieurs cylindres, affichés `-01`, `-02`, `-03`.
- **Granulats / enrobé** : un échantillon → exactement une éprouvette, sans suffixe.

Presque tous les écrans et tableaux découlent de cette distinction. `OdkSample` conserve ce que le terrain a rapporté ; `SampleSpecimen` conserve ce que fait le laboratoire.

Lancer en local

```
cp .env.example .env          # DB password + ODK credentials

docker compose up -d db       # Postgres, mapped to 5433 to avoid conflicts

# Create src/LabTracker.Web/appsettings.Development.json — see Configuration below.
# It is git-ignored and holds real secrets.

cd src/LabTracker.Web
dotnet run                    # migrations are applied automatically on startup
```

Application sur http://localhost:5000. La connexion passe par Azure AD, un compte dans le tenant est donc nécessaire.

```
dotnet test                   # the calculation engine's unit tests
```

4.3.2 Architecture

Deux projets plus un projet de tests :

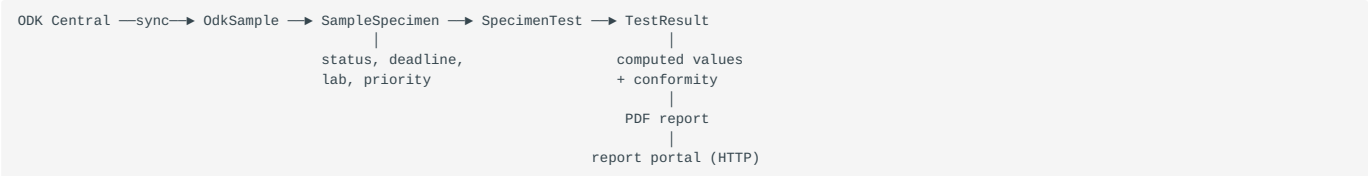
Projet	Contient
LabTracker.Core	Entités du domaine, EF Core, client ODK et worker de synchronisation, règles d'échéance, moteur de calcul des résultats d'essais, générateurs de rapports PDF, envoi au portail
LabTracker.Web	Interface Blazor Server **et** les workers en arrière-plan, dans un seul processus
LabTracker.Core.Tests	Tests xUnit du moteur de calcul et de ses règles

PostgreSQL 16 conserve les payloads bruts d'ODK dans des colonnes `jsonb`, à côté de colonnes typées qui en sont extraites, de sorte qu'un champ jamais promu en colonne reste tout de même récupérable.

Deux workers `BackgroundService` tournent dans le processus web :

- `SyncWorker` : récupère les jeux de données ODK à intervalle régulier.
- `ReportDeliveryWorker` : transmet les rapports PDF en file d'attente au portail de rapports, avec réessais.

Flux de données



4.3.3 Stack technique

- .NET 10 · Blazor Server (rendu interactif côté serveur)
- EF Core 10 + Npgsql · PostgreSQL 16
- **Microsoft.Identity.Web** : authentification Azure AD (OpenID Connect)
- **QuestPDF** : génération des rapports
- Docker Compose, déployé via Coolify

Tout exige un utilisateur authentifié : une politique d'autorisation par défaut couvre chaque endpoint. Le claim `name` de l'utilisateur connecté est inscrit sur chaque entrée d'historique.

4.3.4 Synchronisation ODK

`SyncWorker` effectue un premier passage 15 secondes après le démarrage, puis toutes les *N* minutes (10 par défaut). Le bouton du tableau de bord déclenche un cycle à la demande.

Jeu de données	Mapper	Préfixe de champ
projets	ProjectMapper	-
specs_beton	BetonSpecMapper	`sb_*`
specs_enrobe	EnrobeSpecMapper	-
specs_granulaires	GranulairesSpecMapper	`sg_*`
echantillons_beton	BetonSampleMapper	`eb_*`
echantillons_granulaires	GranulairesSampleMapper	`eg_*`
echantillons_enrobe	EnrobeSampleMapper	`ee_*`
essais_beton	BetonEssaiMapper	-

Chaque jeu de données est acheminé vers l'`IEntityMapper` enregistré pour son `EntityKind`.

À savoir avant de toucher à la synchronisation :

- **Curseur incrémental.** Seules les entités modifiées depuis `LastSyncedAt` sont récupérées. Le curseur est capturé *avant* le travail et avancé seulement après une récupération complètement réussie, de sorte qu'un échec en cours de cycle est réessayé plutôt qu'ignoré.
- **Date de coupure.** `OdkOptions.CutoffUtc` fait ignorer par la synchronisation tout échantillon créé avant cette date, ce qui écarte le bruit historique.
- **L'ordre des jeux de données n'est pas garanti.** `DatasetConfigs.Kind` est persisté comme une *string*, donc l'ordre du worker est alphabétique. Ne jamais présumer qu'un jeu de données est synchronisé avant un autre (voir la section sur le béton frais ci-dessous pour la façon dont c'est géré).
- **Les échecs sont contenus.** Une erreur sur un jeu de données est journalisée puis absorbée, pour que la boucle continue de tourner.

Essais de béton frais sur chantier

Le jeu de données `essais_beton` porte l'affaissement, la teneur en air et la température mesurés sur le chantier pendant que le béton est encore frais. Chaque entité est stockée dans sa propre table `BetonEssais` ; **après** chaque cycle de synchronisation, `FreshConcreteApplier` copie les valeurs sur l'`OdkSample` correspondant, apparié par numéro de projet + numéro d'échantillon.

Cette conception en deux étapes est délibérée : un essai peut être récupéré avant l'échantillon auquel il appartient, donc l'appliquer directement dans le mapper le perdrait à mesure que le curseur avance. Il reste plutôt en attente et est appliqué à un cycle ultérieur. Comme les valeurs vivent sur l'échantillon, chaque éprouvette de cet échantillon les reçoit d'un coup.

4.3.5 Le déroulement au laboratoire

Échéances

Une stratégie par matériau (`Core/Lab/`), distribuée par `DeadlineService` :

- **Béton** : premier cylindre à prélèvement + 7 jours, les autres à + 28 jours.
- **Granulaires / Enrobé** : délais provisoires ; les véritables règles de délai attendues du client restent à confirmer.

Calculée à la création de l'éprouvette, et recalculée seulement si la date de prélèvement change. Une modification manuelle active `DeadlineOverridden`, qui la protège définitivement de tout recalcul.

Essais et charge de travail

Chaque éprouvette porte un ou plusieurs `SpecimenTest`. `TestCatalog` conserve les essais prédéfinis et leurs durées standards :

Essai	Matériau	Heures standards
Bris de cylindre 7 j	Béton	0.25
Bris de cylindre 28 j	Béton	0.25
Granulométrie	Granulaires	1.25
Proctor	Granulaires	2.5
Autre (texte libre)	-	porte son propre nom et ses propres heures

`WorkloadService` additionne les heures en attente par laboratoire. Les laboratoires actifs sont Granby et Sherbrooke.

Résultats d'essais : le moteur de calcul

> ⚠ **À manipuler avec précaution**
>
> LabTracker recalcule les essais ; il ne se contente pas de stocker des valeurs finales. Le moteur vit dans `Core/Lab/Results/` sous forme de fonctions pures, sans effet de bord, ce qui le rend testable unitairement. C'est aussi la partie à traiter avec le plus de précaution, puisqu'un chiffre erroné ici devient silencieusement un résultat de certification.

Fichier	Responsabilité
`GranulometrieCalculator`	Masses des tamis → courbe de passants combinée, pertes granulo, D10/D30/D50/D60, Cu/Cc, module de finesse, fractions du sol
`ProctorCalculator`	Points de compactage → teneur en eau et densités, sommet de la courbe ajustée, sélection de la méthode A/B/C, correction pour gros granulats
`BetonResistanceCalculator`	kN → MPa avec une table de correction d'élancement
`GradingEnvelopeCatalog`	Les 14 fuseaux granulométriques MTQ/BNQ fixes (MG-20, CG-14, Sable, Nette…) et la conformité par tamis
`ConformityService`	Verdicts contre le devis : résistance du béton selon l'âge, courbe granulométrique contre son fuseau, seuil de perte granulo
`SpecimenStatusRules`	Dérive le statut de l'éprouvette à partir des résultats de ses essais

Le béton fait exception : la presse rapporte déjà des MPa, donc la résistance est saisie directement. `BetonResistanceCalculator` n'est donc **pas utilisé par l'interface** ; il est conservé, avec ses tests, au cas où un flux basé sur la charge reviendrait.

Règles de conformité qui comptent :

- **Béton** : un bris doit atteindre **70 %** de la résistance spécifiée à 7 jours, **100 %** à 28 jours.
- **Granulométrie** : la courbe doit rester à l'intérieur de son fuseau, jugée uniquement sur les tamis que le fuseau liste explicitement.
- **Perte granulo** : les pertes grossières et fines doivent rester sous **0,3 %**.

La saisie des résultats fait avancer l'éprouvette automatiquement : tout essai non conforme la marque *Non-conformité* ; un ensemble complet et conforme la marque *Terminé*. Un statut fixé à la main par le laboratoire n'est autrement pas touché. Chaque changement est écrit dans l'historique de l'éprouvette, champ par champ.

Moules Proctor

Le volume et la masse à vide des moules sont des propriétés fixes, revérifiées environ une fois par an, donc configurées une seule fois dans **Paramètres** (barre du haut) plutôt que ressaisies à chaque essai. Chaque résultat enregistré fige un instantané du volume et de la masse avec lesquels il a été calculé, de sorte que revérifier ou retirer un moule ne modifie jamais un résultat déjà existant.

Un Proctor lit aussi les valeurs de passants à 5 mm et 20 mm depuis la **granulométrie de la même éprouvette** ; elles orientent la suggestion de méthode/moule et la correction pour gros granulats. En pratique, la granulométrie est toujours réalisée en premier.

4.3.6 Configuration

Deux sources selon la façon dont l'application tourne :

- **Docker / Coolify** : variables d'environnement (ci-dessous). `docker-compose.yml` les lit depuis `.env`.
- **En local avec `dotnet run`** : `src/LabTracker.Web/appsettings.Development.json`, **exclu de git**, contenant les sections `ConnectionStrings:Default`, `Odk`, `AzureAd` et `Portail`.

Ne jamais commiter de secrets. Le fichier de développement est exclu de git volontairement.

Variable	Rôle
`DB_PASSWORD`	Mot de passe Postgres
`ODK_BASE_URL`, `ODK_EMAIL`, `ODK_PASSWORD`, `ODK_PROJECT_ID`, `ODK_SYNC_INTERVAL`	Compte de service ODK Central et cadence de synchronisation
`AzureAd__Instance`, `AzureAd__TenantId`, `AzureAd__ClientId`, `AzureAd__ClientSecret`	Inscription de l'application Azure AD
`PORTAIL_BASE_URL`, `PORTAIL_TOKEN`, `PORTAIL_ENABLED`	Point d'accès et jeton de service du portail de rapports

Le processus force la culture `fr-CA` pour que les dates et les nombres soient formatés de façon cohérente, peu importe la locale de l'hôte. Derrière un reverse proxy, `ForwardedHeaders` est configuré pour que les redirections OIDC se résolvent correctement.

Si un proxy d'entreprise effectuant une inspection TLS brise la connexion ODK avec `UntrustedRoot`, définir `"Odk:AllowUntrustedCertificate": true` dans le fichier de développement. **Jamais en production.**

4.3.7 Rapports

Deux rapports PDF, tous deux reproduisant les formulaires papier existants du laboratoire, générés avec QuestPDF depuis `Core/Reports/`.

Rapport	Portée	Point d'accès
Ruptures des cylindres de béton (CAN/CSA-A23.2-9C)	par projet	`GET /reports/ruptures/{projectNumber}`
Rapport d'analyse sur sols (LC 21-040 + Proctor)	par éprouvette	`GET /reports/granulo/{specimenId}`

Les deux peuvent aussi être transmis au portail depuis la fenêtre de l'éprouvette.

Points qu'une personne qui découvre le projet va rencontrer :

- Le rapport béton imprime **une ligne par échantillon**, pas par cylindre : les relevés à l'état frais, le bris à 7 jours, les deux bris à 28 jours, leur moyenne, et la moyenne cumulative de ces moyennes. La moyenne cumulative de la première ligne est toujours des tirets.
- Un projet peut référencer plusieurs devis de béton, alors que le formulaire ne prévoit qu'un seul bloc de spécification ; le rapport émet donc **une page par devis**.
- Les valeurs sous leur exigence sont marquées `**`, expliquées par la légende sous le tableau.
- Le logo d'en-tête est une **ressource intégrée** (`Reports/Assets/logo-entete.png`), pour que l'image de marque ne puisse pas manquer dans un conteneur.

Envoi au portail de rapports

Les rapports sont transmis au portail central de l'entreprise (`lcl-rapports`). L'interface est spécifiée dans `docs/contrat-envoi-rapports.md`, la source de vérité partagée, reflétée du côté du portail. À lire avant de modifier quoi que ce soit ici.

Le mécanisme est une **outbox** : mettre un rapport en file d'attente écrit une ligne `ReportDelivery` contenant le PDF, et `ReportDeliveryWorker` l'envoie en arrière-plan, un à la fois.

- Une panne du portail **retarde** un rapport, elle ne le perd jamais.
- Chaque rapport a un `sourceId` stable, donc renvoyer un rapport corrigé le **remplace** plutôt que de le dupliquer.
- Les erreurs sont classées en réessayables ou non : une requête mal formée n'est jamais réessayée indéfiniment, et un incident réseau ne fait jamais perdre un rapport. Un projet inconnu est réessayé selon un horaire étalé, car le portail peut simplement ne pas encore le connaître.
- Chaque envoi déclare explicitement ses **non-conformités**, liste vide incluse ; le portail se fie à cette déclaration plutôt que d'analyser le PDF.

L'envoi est **manuel** aujourd'hui (un bouton par rapport). L'envoi automatique à la complétion est conçu mais pas activé. **Paramètres** → **Rapports** remet en file d'attente tous les rapports déjà envoyés, pour repeupler le portail après que son dossier a été vidé.

4.3.8 Tableau de bord

- Liste d'éprouvettes filtrable et triable : cliquer sur l'en-tête d'une colonne pour trier, sur l'icône de filtre pour ouvrir un panneau.
- Les filtres se combinent : numéro d'échantillon, type, projet, plage de dates de soumission, statut, priorité, non-conformité, laboratoire et réception physique.
- Des cartes de mesures, plus un panneau de charge de travail par laboratoire, extensible jusqu'au détail par éprouvette.
- La vue par défaut trie par échéance et cache *Terminé*, pour que le travail terminé disparaisse de la liste.
- Cliquer sur une ligne ouvre la fenêtre de l'éprouvette : **Détails** (données ODK en lecture seule + boutons de rapport), **Historique** (audit complet), **Modifier** (statut, priorité, échéance, laboratoire, réception, notes, essais, numéro d'échantillon manuel), **Résultats** (saisie des essais, calcul en direct, conformité, graphiques).
- Un manuel d'utilisation se trouve à `/manuel` ; les réglages s'ouvrent depuis la barre du haut.

4.3.9 Organisation du dépôt

```
labtracker/
├── docker-compose.yml          Postgres + app, for Coolify
├── .env.example
├── docs/
│   └── contrat-envoi-rapports.md    Interface contract with the report portal
├── tests/
│   └── LabTracker.Core.Tests/      xUnit tests for the calculation engine
└── src/
    ├── LabTracker.Core/
    │   ├── Entities/           OdkProject, OdkSample, SampleSpecimen, SpecimenTest, TestResult,
    │   │                       BetonEssai, ProctorMould, ReportDelivery, AuditEntry,
    │   │                       OdkDatasetConfig, Beton/Granulaires/EnrobeSpec
    │   ├── Enums/              SampleType, EntityKind, LabTrackingStatus, Priority, Labo,
    │   │                       TestType, ResultConformity, DeliveryStatus
    │   ├── Data/               AppDbContext
    │   ├── Lab/                Deadline strategies, DeadlineService, TestCatalog, WorkloadService
    │   │   └── Results/        The calculation engine (see above)
    │   ├── Reports/            RuptureReportGenerator, GranuloReportGenerator, Assets/
    │   │   └── Portal/         Outbox delivery: client, service, worker, NC declaration
    │   ├── Odk/                OdkCentralClient, SyncWorker, FreshConcreteApplier, OdkOptions
    │   │   └── Mappers/        IEntityMapper + one mapper per entity kind + MapperHelpers
    │   └── Migrations/
    └── LabTracker.Web/
        ├── Program.cs          DI, auth, startup migration, dataset + mould seeding, endpoints
        ├── DesignTimeDbContextFactory.cs   Lets `dotnet ef` run without booting the host
        ├── Components/
        │   ├── Pages/          Dashboard, Manual, Error, NotFound
        │   └── Shared/         SampleDetailDialog, ResultsTab, SettingsDialog
        └── wwwroot/app.css     Custom dark theme
```

4.3.10 Travailler sur le projet

Migrations de base de données

Les migrations sont appliquées **automatiquement au démarrage**, donc il est rare de devoir lancer quelque chose à la main. Pour en ajouter une après un changement de modèle :

```
cd src/LabTracker.Web
dotnet ef migrations add MyChange --project ../LabTracker.Core --startup-project .
```

Deux points pratiques :

- Si l'application tourne, la compilation échoue sur une DLL verrouillée. Ajouter `--configuration Release` pour compiler vers un autre dossier de sortie.
- **Ne jamais modifier une migration déjà appliquée.** En ajouter une nouvelle à la place ; c'est l'historique appliqué qui correspond à la base de données.

Tests

`dotnet test` couvre le moteur de calcul, les règles de conformité, les fuseaux granulométriques, la dérivation du statut et le contenu envoyé au portail. Quand une formule change, modifier ou ajouter un test avec elle : plusieurs ont été écrits en reproduisant une valeur connue tirée des chiffriers du laboratoire, et ce sont eux qui prouvent que la transposition est fidèle.

Conventions

- Commentaires en **anglais** ; le français n'est gardé que pour le vocabulaire du domaine (*béton*, *éprouvette*, *granulaires*) et pour les libellés cités de l'interface ou des rapports.
- Les commentaires expliquent **pourquoi**, pas ce que le code dit déjà.
- Les enums sont persistées **sous forme de chaînes**, pour que les valeurs en base survivent à une réorganisation.
- `AppDbContext` est résolu via une **factory** : les composants Blazor et les workers en arrière-plan créent des contextes hors d'une portée de requête.

4.3.11 Limites connues

- **Enrobé** : le préfixe de champ `ee_*` est extrapolé à partir des conventions du béton et des granulats, et n'a jamais été vérifié contre une vraie soumission. Aucune saisie de résultat non plus.
- **Les règles d'échéance** pour les granulats et l'enrobé sont provisoires en attendant les délais du client ; la règle du béton reste aussi à confirmer.
- **Les calculs de granulométrie** devraient encore être validés de bout en bout contre un jeu de données de référence rempli.
- **Proctor** : la densité sèche maximale corrigée reproduit littéralement la formule du chiffrier, et cette formule donne une valeur invraisemblable (elle mélange des pourcentages là où la norme utilise des fractions). Laissée telle quelle délibérément, en attente d'une décision.
- **L'envoi automatique des rapports** est conçu mais pas activé.
- **Mise à jour en temps réel** : le tableau de bord se rafraîchit sur ou après un enregistrement ; SignalR pourrait pousser les mises à jour.
- `SampleTracking` et `BetonResistanceCalculator` ne sont **plus utilisés** par l'application en production. Les deux sont conservés intentionnellement ; ne les supprimer que si on est certain qu'ils ne reviendront pas.

4.4 Chatbot LCL : assistant technique pour techniciens de chantier

Système RAG (Retrieval-Augmented Generation) qui répond aux questions techniques des techniciens à partir des normes BNQ, CSA et autres documents de référence en génie civil.

4.4.1 Architecture

- **db/** : Postgres 16 + pgvector pour stocker les chunks vectorisés
- **ingestion/** : script Python qui lit les PDFs, les découpe en chunks, les vectorise via OpenAI Embeddings et les insère dans Postgres
- **backend/** : (à venir) API ASP.NET Core, endpoint /chat
- **frontend/** : (à venir) PWA Blazor pour les techniciens

4.4.2 Démarrage local

1. Cloner le repo
2. Copier `.env.example` vers `.env` et remplir les valeurs
3. `docker compose up -d`
4. Adminer disponible sur http://localhost:8081 (serveur: `postgres`)

4.4.3 Ingestion des PDFs

1. Placer les PDFs dans `ingestion/documents/`
2. Copier `ingestion/.env.example` vers `ingestion/.env` et remplir
3. `cd ingestion && .\venv\Scripts\Activate.ps1 && python ingest.py`

4.4.4 Statut actuel

- [x] Postgres + pgvector en local
- [x] Script d'ingestion écrit
- [ ] Premier test d'ingestion
- [ ] Backend ASP.NET
- [ ] Frontend Blazor PWA
- [ ] Déploiement Coolify