المدرسة الوطنية للعلوم التطبيقية بتطوان
ⵜⵉⵍⵍⵉⵏ ⵜⴰⵏⴰⵎⵓⵔⵜ ⵜⵉⵖⵔⵉⴼⵜ ⵜⴰⵏⴰⵎⵓⵔⵜ ⵜⴰⵏⵉⵙⵉⵢⵜ
Ecole Nationale des Sciences Appliquées de Tétouan

HOLCIM

ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES DE TÉTOUAN

Filière Génie Informatique

RAPPORT DE PROJET DE FIN D'ANNÉE

Développement agentique

d'une plateforme de gestion des visiteurs pour Holcim Maroc

RÉALISÉ PAR

Abderrahim Sadiki

ENCADRÉ PAR

Omar Naciri
Responsable Business Applications

Omar Bassiry
Architecte de solutions

Abderrahim Tahiri
Professeur encadrant

Effectué au sein de **Holcim Maroc**

Période : du 1er juillet au 30 août 2026

Année universitaire 2026–2027

Dédicace

À mes parents,
pour leur soutien sans faille et les sacrifices consentis
tout au long de mon parcours.

À mes frères et sœurs,
pour leurs encouragements constants.

À mes enseignants,
pour la rigueur et la curiosité qu'ils m'ont transmises.

À mes amis,
pour les moments partagés.

Je dédie ce modeste travail.

Remerciements

Au terme de ce projet de fin d'année, je tiens à exprimer ma gratitude à l'ensemble des personnes qui ont contribué, de près ou de loin, à son aboutissement.

Je remercie d'abord **Monsieur Omar Naciri**, **Responsable Business Applications** de **Holcim Maroc**, mon encadrant professionnel, pour la confiance qu'il m'a accordée en me confiant ce sujet à fort enjeu opérationnel, pour sa disponibilité et pour la qualité de ses orientations et de ses retours hebdomadaires tout au long du projet.

Ma gratitude va tout particulièrement à **Monsieur Omar Bassiry**, **Architecte de solutions**, dont l'accompagnement quotidien a profondément marqué ce travail. Sa disponibilité constante, la clarté de ses explications sur l'architecture du système d'information de Holcim et l'exigence bienveillante de ses relectures m'ont permis de progresser bien au-delà de ce que le sujet demandait. Les choix de conception présentés dans ce rapport doivent beaucoup à nos échanges.

J'adresse également mes remerciements à **Monsieur Abderrahim Tahiri**, professeur et encadrant pédagogique, dont les conseils méthodologiques et la rigueur ont structuré ma démarche.

Ma reconnaissance va aussi à **Holcim Maroc** et au service **Business Applications** pour leur accueil et pour les moyens mis à ma disposition durant la réalisation du projet.

Je remercie enfin le corps professoral et administratif de **l'École Nationale des Sciences Appliquées de Tétouan** pour la formation dispensée, ainsi que les membres du jury qui me font l'honneur d'évaluer ce travail.

Résumé

L'accès aux sites industriels de Holcim Maroc — usines, carrières et sièges — repose encore largement sur des registres papier tenus en guérite. Ce mode de gestion ne permet ni de savoir en temps réel qui se trouve sur un site, ni de garantir qu'un visiteur a bien suivi l'induction sécurité obligatoire, ni de produire les états exigés lors d'un audit ou d'une évacuation.

Ce projet a consisté à concevoir et à réaliser **Gestion Visiteur (GV)**, une plateforme web de gestion du cycle de vie complet des visites, de la pré-déclaration à distance jusqu'à la sortie du site. La solution se compose de deux portails distincts — un portail visiteur exposé sur Internet et un portail back-office réservé au réseau interne — servis par une API unique.

La réalisation a suivi une démarche de **développement agentique** : Claude Code a été utilisé comme outil d'assistance pour accélérer l'analyse, la production du code, les tests et les corrections. Les choix d'architecture, la validation fonctionnelle et la responsabilité du résultat sont toutefois restés sous contrôle humain.

L'architecture retenue est un **monolithe modulaire** sur la plateforme Java : une seule application déployable, découpée en modules aux frontières vérifiées automatiquement à la compilation. Ce choix préserve la simplicité d'exploitation d'un déploiement unique tout en imposant la discipline de séparation d'un système distribué. Le cycle de vie d'une visite est modélisé par une **machine à états** unique, et le cloisonnement des données par site est appliqué au niveau de la couche de persistance plutôt que laissé à la vigilance de chaque requête.

La plateforme couvre aujourd'hui la pré-déclaration, la déclaration spontanée par code QR, l'induction sécurité avec quiz et certificat, le parcours VIP à validation hiérarchique, les opérations de guérite, l'administration des sites, des utilisateurs et des habilitations, la traçabilité des actions, et la restitution statistique.

Mots-clés : gestion des visiteurs, développement agentique, Claude Code, sécurité industrielle, monolithe modulaire, Spring Boot, Next.js, authentification unique, cloisonnement des données, PostgreSQL, Kubernetes.

Abstract

Access to Holcim Maroc's industrial sites — plants, quarries and offices — still relies largely on paper registers kept at the gatehouse. That practice makes it impossible to know in real time who is on a site, to guarantee that a visitor has completed the mandatory safety induction, or to produce the records required during an audit or an evacuation.

This project covers the design and implementation of **Gestion Visiteur (GV)**, a web platform managing the complete life cycle of a visit, from remote pre-declaration through to the visitor leaving the site. The solution consists of two distinct portals — a visitor portal exposed to the Internet and a back-office portal restricted to the internal network — served by a single API.

The implementation followed an **agentic development** approach: Claude Code was used as an assistance tool to accelerate analysis, code production, testing and corrections. Architectural choices, functional validation and responsibility for the result nevertheless remained under human control.

The chosen architecture is a **modular monolith** on the Java platform: one deployable application, split into modules whose boundaries are verified automatically at build time. This preserves the operational simplicity of a single deployment while enforcing the separation discipline of a distributed system. A visit's life cycle is modelled as a single **state machine**, and per-site data partitioning is enforced in the persistence layer rather than left to the care of each individual query.

The platform currently covers pre-declaration, spontaneous declaration via QR code, safety induction with quiz and certificate, the VIP journey with hierarchical approval, gatehouse operations, administration of sites, users and permissions, action traceability, and statistical reporting.

Keywords: visitor management, agentic development, Claude Code, industrial safety, modular monolith, Spring Boot, Next.js, single sign-on, data partitioning, PostgreSQL, Kubernetes.

Table des matières

Dédicace i

Remerciements ii

Résumé iii

Abstract iv

Liste des figures ix

Liste des tableaux xi

Liste des extraits de code xii

Liste des abréviations xiii

Introduction générale 1

1 Contexte général du projet 3

1.1 Présentation de l'organisme d'accueil 3

1.1.1 Holcim Maroc 3

1.1.2 Le service Business Applications 4

1.1.3 Les sites concernés par le projet 5

1.2 Étude de l'existant 5

1.2.1 Le processus actuel 5

1.2.2 Critique de l'existant 6

1.2.3 Justification d'un développement interne 6

1.3 Cahier des charges 7

1.3.1 Besoin exprimé 7

1.3.2 Périmètre fonctionnel 7

1.3.3 Hors périmètre 7

1.3.4 Contraintes 8

1.4 Méthodologie et conduite de projet 8

1.4.1 Démarche adoptée 8

1.4.2 Le développement agentique 9

1.4.3 Outils de travail 12
1.4.4 Planification et état d'avancement 13

2 Analyse et spécification des besoins 15
2.1 Identification des acteurs 15
2.1.1 La population visiteur 15
2.1.2 La population back-office 16
2.2 Besoins fonctionnels 16
2.2.1 Les acteurs et leurs familles 16
2.2.2 Cas d'utilisation du visiteur ordinaire 18
2.2.3 Cas d'utilisation du visiteur protocolaire 19
2.2.4 Cas d'utilisation de l'agent de sécurité 20
2.2.5 Cas d'utilisation de l'administrateur 21
2.2.6 Authentification, sécurité et notifications 21
2.2.7 Conformité et traitements automatiques 22
2.2.8 Récits utilisateurs 23
2.3 Besoins non fonctionnels 25
2.3.1 Le cloisonnement, exigence structurante 25

3 Conception 27
3.1 Architecture générale 27
3.1.1 Diagramme de composants 27
3.1.2 Le choix du monolithe modulaire 28
3.1.3 Architecture interne d'un module 29
3.2 Modélisation du cycle de vie d'une visite 30
3.2.1 Une machine à états unique 30
3.2.2 Deux parcours, deux agrégats 31
3.2.3 Le parcours ordinaire, de bout en bout 34
3.3 Modèle de données 35
3.3.1 Schéma de la base de données 35
3.3.2 Conventions de modélisation 37
3.4 Conception de la sécurité 38
3.4.1 Chaîne d'authentification 38
3.4.2 Modèle d'habilitation 39
3.4.3 Défense en profondeur 39
3.4.4 Évaluation du niveau de sécurité 40
3.4.5 Protection des données personnelles 41

4 Réalisation 44

4.1 Douze notions pour lire ce chapitre 44
4.2 Choix technologiques 46
4.2.1 Pourquoi Spring Boot côté serveur 47
4.2.2 Ce qu'est un module, concrètement 48
4.2.3 Pourquoi Next.js côté navigateur 49
4.2.4 Lecture technique de la stack 50
4.2.5 Chemin d'une requête, étape par étape 52
4.2.6 Le contrat d'API comme source de vérité 52
4.3 Environnement de développement 53
4.3.1 Stack exécutée en local 53
4.3.2 Les services d'infrastructure, un par un 54
4.3.3 Démarrage et cycle de travail 56
4.4 Quatre mécanismes en détail 57
4.4.1 La machine à états : une table, et une seule porte 58
4.4.2 Le cloisonnement : un filtre au niveau de la session 59
4.4.3 Les droits, recalculés plutôt que transportés 60
4.4.4 L'idempotence, ou pourquoi deux clics ne font qu'une visite . . . 61
4.5 Organisation du code 61
4.5.1 Organisation d'une fonctionnalité côté client 62
4.6 Réalisation des principales fonctionnalités 62
4.6.1 Comment le système fonctionne, dans son ensemble 62
4.6.2 Lecture fonctionnelle de bout en bout 63
4.6.3 Authentification et gestion des accès 64
4.6.4 Parcours du visiteur 66
4.6.5 Opérations de guérite 70
4.6.6 Parcours des visites protocolaires 74
4.6.7 Administration 76
4.6.8 Traçabilité et restitution 83
4.6.9 Un seul poste, deux rendus 85
4.7 Bilan des fonctionnalités livrées 87
4.8 Difficultés rencontrées et solutions apportées 87
4.8.1 Comprendre un périmètre métier étendu 87
4.8.2 Conduire seul un projet complet 88
4.8.3 Respecter le délai sans sacrifier la qualité 88
4.8.4 L'évaluation sécurisée du quiz 88
4.8.5 Éviter le double envoi des rapports planifiés 88
4.8.6 Maîtriser l'ordre des opérations et la transaction 88

5 Tests, qualité et déploiement 89

5.1 Stratégie de tests 89
5.1.1 Le test d'architecture, un niveau à part 90
5.2 Intégration continue 90
5.2.1 Construction et publication des images 92
5.3 Déploiement 93
5.3.1 Images Docker multi-stage 93
5.3.2 Ce que font Kubernetes et Helm 94
5.3.3 Le chart du projet 95
5.3.4 Séparation entre local, recette et production 96
5.4 Bilan de la vérification 97
5.4.1 Scénarios de sécurité vérifiés 98

Conclusion générale et perspectives **99**

Bibliographie et webographie **104**

A Annexes **106**
A.1 Extrait du contrat d'API 106
A.2 Format des erreurs 106
A.3 Exemple de migration de schéma 107
A.4 Une règle d'architecture exécutable 107
A.5 Extrait de la chaîne d'intégration continue 108
A.6 Glossaire métier 109

Table des figures

1.1 Cycle de travail avec un agent : la production est déléguée, la décision et la preuve ne le sont pas 10
1.2 Ordonnancement des huit sprints et enchaînement de leurs dépendances 14

2.1 Acteurs du système, regroupés par famille 17
2.2 Cas d'utilisation — parcours du visiteur ordinaire 18
2.3 Cas d'utilisation — parcours du visiteur VIP 19
2.4 Cas d'utilisation — guérite et tableaux de bord 20
2.5 Cas d'utilisation — administration et référentiels 21
2.6 Cas d'utilisation — authentification, sécurité et notifications 22
2.7 Cas d'utilisation — conformité, données et automatisation 23

3.1 Diagramme de composants de la plateforme 27
3.2 Machine à états du cycle de vie d'une visite ordinaire 30
3.3 Séquence — déclaration et validation d'une visite protocolaire 32
3.4 Séquence — induction VIP, quiz et délivrance du certificat 33
3.5 Séquence — pré-déclaration d'un visiteur ordinaire 33
3.6 Séquence — enregistrement de l'entrée et de la sortie en guérite 35
3.7 Schéma des données du domaine — classes, associations et listes de valeurs 36
3.8 Séquence — connexion d'un utilisateur interne par le service d'identité 38
3.9 Information du visiteur sur le traitement de ses données 42

4.1 Frontière d'un module : seuls `api/` et `events/` sont atteignables de l'extérieur 48
4.2 Espaces de stockage et objets conservés dans MinIO 55
4.3 Courriels émis par la plateforme, interceptés en développement 56
4.4 Page de connexion du back-office 64
4.5 Page d'authentification du fournisseur d'identité Holcim 65
4.6 Écran de gestion des rôles et habilitations 66
4.7 Page d'accueil du portail visiteur sur écran d'ordinateur 67
4.8 Identification du visiteur par code à usage unique 67
4.9 Portail visiteur — entrée dans le parcours — Espace visiteur connecté . 68
4.10 Pré-déclaration réelle — sélection et formulaire 69
4.11 Induction ordinaire — contrôle de consultation et certification 69

4.12 Passeport visiteur et suivi de ses propres visites 70
4.13 Écran d'accueil de la guérite 71
4.14 Contrôle d'un passeport avant enregistrement de l'entrée 71
4.15 Écrans opérationnels de la guérite 72
4.16 Confirmation avant déclenchement de la procédure d'évacuation 73
4.17 Accès mobile à l'induction VIP 74
4.18 Démarrage du quiz de sécurité — Question sans indication de la bonne réponse 75
4.19 Quiz évalué côté serveur et certificat obtenu 76
4.20 Parcours de la visite protocolaire 76
4.21 Suivi et accueil des visites protocolaires — Guérite dédiée aux visites VIP 77
4.22 Gestion des sites 77
4.23 Gestion des comptes utilisateurs 78
4.24 Administration des référentiels 78
4.25 Modèles de courriels utilisés par les parcours 79
4.26 Paramètres et référentiels propres à un site 79
4.27 Administration des guérites d'un site : création, activation et désactivation 80
4.28 Paramétrage des dérogations et contrôle des habilitations 81
4.29 Messages émis par la plateforme au cours d'un parcours complet 82
4.30 Journal d'audit 83
4.31 Historique des transitions d'une visite 84
4.32 Tableau de bord des indicateurs 84
4.33 Configuration et suivi des rapports planifiés 85
4.34 Poste de guérite en rendu tactile, activé par le paramètre « mode tablette » de la fiche site 86

5.1 Dépôt du projet et historique des contributions 90
5.2 Pipeline GitLab réussi avec les étapes test, build, image et package . . . 92
5.3 Déploiement : trois charges dans le cluster, services d'infrastructure à l'extérieur 95

Liste des tableaux

1.1 Sites pris en charge par la plateforme 5
1.2 Périmètre fonctionnel du projet 7
1.3 Outils utilisés durant le projet 12
1.4 Planification de la réalisation, sprints du 1er juillet au 28 août 2026 . . 13

2.1 Acteurs du système et leurs responsabilités 16
2.2 Extrait des récits utilisateurs 24
2.3 Exigences non fonctionnelles 25

3.1 Responsabilité de chaque module 29
3.2 Signification des états d'une visite 31
3.3 Évaluation raisonnée des contrôles de sécurité 40

4.1 Définitions des notions techniques employées 45
4.2 Technologies retenues et justification 46
4.3 Rôle technique de chaque couche 51
4.4 Services de l'environnement local 53
4.5 Synthèse des livrables fonctionnels 87

5.1 Niveaux de test et périmètre couvert 89
5.2 Étapes du pipeline GitLab CI 91
5.3 Bilan des tests automatisés 97
5.4 Le projet en chiffres 100

Liste des extraits de code

4.1 Démarrage des services d'infrastructure 56
4.2 Démarrage du serveur applicatif 56
4.3 Démarrage des deux portails 57
4.4 Vérifications avant validation 57
4.5 Table des transitions autorisées (extrait) 58
4.6 Le garde de transition 58
4.7 Déclaration du filtre de périmètre 59
4.8 Arborescence du dépôt 61
5.1 Construction locale des trois images 93
5.2 Installation dans l'environnement de recette 96
5.3 Commandes de vérification exécutées 98
A.1 Extrait de la spécification OpenAPI 106
A.2 Exemple de réponse d'erreur 106
A.3 Migration créant la table des visites 107
A.4 Règle imposant une décision explicite de portée à chaque entité 107
A.5 Étape de test du pipeline GitLab CI 108

Liste des abréviations

Sigle	Signification
ACL	*Anti-Corruption Layer* — couche d'adaptation vers un système externe
API	*Application Programming Interface* — interface de programmation
BO	Back-office — portail d'administration interne
CI/CD	*Continuous Integration / Continuous Delivery* — intégration et livraison continues
CNDP	Commission nationale de contrôle de la protection des données à caractère personnel
CSP	*Content Security Policy* — politique restreignant les ressources chargées par une page
CRUD	*Create, Read, Update, Delete*
CSV	*Comma-Separated Values* — format d'export tabulaire
GV	Gestion Visiteur — nom du projet
HSTS	*HTTP Strict Transport Security* — imposition du protocole chiffré
HTTP	*HyperText Transfer Protocol*
IAM	*Identity and Access Management* — gestion des identités et des accès
JPA	*Jakarta Persistence API* — correspondance objet-relationnel en Java
JSON	*JavaScript Object Notation*
JWT	*JSON Web Token* — jeton d'authentification signé
LDAP	*Lightweight Directory Access Protocol* — annuaire d'entreprise
OCI	*Open Container Initiative* — norme de format des images de conteneur

Sigle	Signification
OIDC	*OpenID Connect* — protocole d'authentification fédérée
ORM	*Object-Relational Mapping*
OTP	*One-Time Password* — code à usage unique
PDF	*Portable Document Format*
QR	*Quick Response* — code-barres bidimensionnel
RGPD	Règlement général sur la protection des données
REST	*Representational State Transfer*
SGBD	Système de gestion de base de données
SMTP	*Simple Mail Transfer Protocol*
SPA	*Single Page Application*
SPI	*Service Provider Interface* — interface implémentée par un module fournisseur
SSO	*Single Sign-On* — authentification unique
SQL	*Structured Query Language*
TLS	*Transport Layer Security* — chiffrement des échanges réseau
UML	*Unified Modeling Language*
UUID	*Universally Unique Identifier* — identifiant opaque non séquentiel
VIP	*Very Important Person* — visite protocolaire
WAF	*Web Application Firewall* — pare-feu applicatif

Introduction générale

Un site industriel n'est pas un lieu ouvert. Une cimenterie concentre des engins lourds, des zones de tir, des silos et des installations haute température ; toute personne qui y pénètre sans avoir été identifiée, enregistrée et informée des consignes de sécurité constitue un risque pour elle-même et pour les équipes en poste. La maîtrise des flux de visiteurs n'est donc pas une commodité administrative : c'est une obligation de sécurité, et une exigence d'audit.

Chez Holcim Maroc, cette maîtrise repose encore, sur la plupart des sites, sur un registre papier tenu par l'agent de guérite. Le visiteur décline son identité à l'arrivée, l'agent la recopie, remet un badge, et consigne l'heure d'entrée. Le procédé est simple, mais il porte en lui plusieurs limites structurelles. Il ne permet pas de savoir, à un instant donné, combien de personnes se trouvent sur le site — information pourtant vitale lors d'une évacuation. Il ne garantit pas que le visiteur a reçu l'induction sécurité, puisque rien ne relie la formation à l'autorisation d'entrée. Il rend l'exploitation statistique laborieuse, chaque analyse supposant une ressaisie. Et il ne laisse aucune trace exploitable en cas de contrôle : un registre manuscrit se perd, se rature, et ne dit pas qui a autorisé quoi.

Problématique. Comment concevoir un système d'information qui garantisse qu'aucun visiteur n'accède à un site industriel sans avoir été identifié, formé aux consignes de sécurité et autorisé, tout en restant assez simple pour être utilisé par un agent de guérite sur une tablette, par temps de forte affluence, sans ralentir le flux ?

Cette question en appelle plusieurs autres, qui ont structuré le travail. Comment permettre au visiteur de préparer sa venue à distance sans créer de compte ? Comment rendre l'induction sécurité vérifiable plutôt que déclarative ? Comment traiter le cas particulier des visites protocolaires, soumises à validation hiérarchique ? Comment garantir qu'un responsable d'un site ne voie jamais les données d'un autre site ? Et comment produire, sans effort de ressaisie, les états attendus par la direction sûreté ?

Objectifs du projet. Le présent travail vise à concevoir et réaliser une plateforme couvrant le cycle de vie complet d'une visite : pré-déclaration à distance, induction sécurité avec évaluation, validation lorsqu'elle est requise, enregistrement de l'entrée en guérite, suivi de la présence, et enregistrement de la sortie. La plateforme doit en outre offrir l'administration des sites et des référentiels, la gestion des utilisateurs et

de leurs habilitations, la traçabilité de toutes les actions sensibles, et la restitution statistique.

Organisation du rapport. Ce document s'articule en cinq chapitres. Le **premier** présente l'organisme d'accueil, le contexte du projet, l'étude de l'existant et la méthodologie retenue. Le **deuxième** conduit l'analyse et la spécification des besoins : acteurs, cas d'utilisation, exigences fonctionnelles et non fonctionnelles. Le **troisième** expose la conception — architecture logicielle, modèle de données, machine à états du cycle de visite, et mécanismes de sécurité. Le **quatrième** décrit la réalisation : environnement technique, choix technologiques, implémentation et interfaces obtenues. Le **cinquième** traite de la qualité, des tests et du déploiement. Une conclusion générale dresse le bilan du travail et ouvre sur les perspectives d'évolution.

CHAPITRE 1

Contexte général du projet

Introduction

> *Ce premier chapitre situe le projet dans son environnement. Il présente l'organisme d'accueil, analyse la situation existante et ses limites, formule le besoin qui en découle, et expose la méthode de travail ainsi que la planification adoptées.*

1.1 Présentation de l'organisme d'accueil

1.1.1 Holcim Maroc

Holcim Maroc est le leader national du secteur des matériaux de construction, présent dans le Royaume depuis 1928 [23]. Son activité couvre la production et la commercialisation de ciment, de béton, de granulats, de mortiers, de plâtre et de chaux. L'entreprise résulte du rapprochement, engagé en 2016, entre Lafarge Ciments et Holcim Maroc — prolongement national de la fusion mondiale des deux groupes cimentiers — et son capital est aujourd'hui majoritairement détenu par une coentreprise entre le groupe Holcim et le fonds d'investissement marocain Al Mada [23].

L'outil industriel s'appuie sur **sept cimenteries** — Agadir-Souss, Bouskoura, Settat, Fès, Tétouan, Meknès et Oujda —, **trois centres de broyage** et **dix-neuf centrales à béton**, auxquels s'ajoutent carrières, plateformes de distribution et sites administratifs dont le siège de Casablanca [24]. L'ensemble mobilise environ **2 500 collaborateurs et sous-traitants** [24].

Ces deux chiffres commandent la conception du système. La dispersion géographique impose qu'un système de gestion des visiteurs soit multi-sites dès l'origine, et que le périmètre de chaque utilisateur soit cloisonné : un agent de sécurité de Meknès n'a aucune raison de voir les visiteurs d'Oujda. La part des sous-traitants, elle, explique pourquoi le visiteur n'est pas un cas marginal mais un flux quotidien : la population qui franchit les portes ne se réduit pas aux salariés.

Une culture où la sécurité est une valeur fondamentale. L'entreprise revendique l'ambition d'être la plus sûre du pays pour ses collaborateurs, ses clients et ses parties prenantes, et mobilise chaque année plusieurs milliers de collaborateurs et de sous-traitants autour de journées dédiées à la santé et à la sécurité. Ce point éclaire directement le sujet de ce projet. Dans une cimenterie coexistent des fours à plus de mille degrés, des engins de carrière, des zones de tir et des silos ; une personne non informée des consignes y est un danger pour elle-même et pour les équipes en poste. Garantir qu'aucun visiteur n'accède à un site sans avoir suivi l'induction sécurité n'est donc pas une formalité administrative : c'est l'application, au flux des visiteurs, d'une exigence que l'entreprise s'impose déjà à elle-même.

1.1.2 Le service Business Applications

Le stage s'est déroulé au siège de Holcim Maroc à Casablanca, situé avenue La Mecque, au sein du service **Business Applications**. Ce service accompagne la conception, l'évolution et l'intégration des applications qui répondent aux besoins internes de l'entreprise. Le projet a été réalisé individuellement sous l'encadrement de Monsieur Omar Naciri, Responsable Business Applications, qui assurait la priorisation et la validation hebdomadaire des résultats, et de Monsieur Omar Bassiry, Architecte de solutions, dont l'accompagnement technique a guidé les choix de conception exposés dans ce rapport.

1.1.3 Les sites concernés par le projet

La version livrée couvre les **sept cimenteries** du groupe et le **siège de Casablanca**, soit huit sites (tableau 1.1). Le modèle ne leur est pas propre pour autant : un site est une donnée administrée, créée depuis le back-office, et rien dans le code ne présuppose leur nombre. Les centres de broyage, les carrières et les plateformes de distribution s'y ajouteraient par simple saisie, sans développement.

Table 1.1 – Sites pris en charge par la plateforme

Code	Libellé	Ville
CASA	Siège Casablanca	Casablanca
AGADIR	Usine d'Agadir-Souss	Agadir
OUJDA	Usine d'Oujda	Oujda
BOUSKOURA	Usine de Bouskoura	Bouskoura
FES	Usine de Fès	Fès
MEKNES	Usine de Meknès	Meknès
SETTAT	Usine de Settat	Settat
TETOUAN	Usine de Tétouan	Tétouan

1.2 Étude de l'existant

1.2.1 Le processus actuel

Le processus d'accueil d'un visiteur se déroule aujourd'hui en quatre temps. Le visiteur se présente à la guérite sans annonce préalable, ou après un simple appel téléphonique à son hôte. L'agent de sécurité recueille son identité, la consigne dans un registre papier, et lui remet un badge physique. Le visiteur accède ensuite au site, où il est en principe pris en charge par la personne visitée. À son départ, il restitue le badge et l'agent note l'heure de sortie — lorsqu'il y pense et que l'affluence le permet.

1.2.2 Critique de l'existant

L'analyse de ce processus fait apparaître sept limites, dont chacune a été traduite en exigence dans le cahier des charges.

1. **Aucune vision temps réel de la présence.** Le registre n'est consultable qu'en guérite, et seulement en le feuilletant. En cas d'évacuation, l'agent ne dispose d'aucune liste fiable et immédiate des personnes présentes.

2. **L'induction sécurité n'est pas vérifiable.** Rien ne relie la remise d'un livret ou la diffusion d'une vidéo à l'autorisation d'entrée. Un visiteur peut entrer sans avoir été informé, et l'entreprise ne peut pas prouver le contraire.

3. **Le temps d'attente en guérite est subi.** Toute la saisie a lieu à l'arrivée, ce qui allonge le passage et crée des files aux heures de pointe.

4. **Les données sont inexploitables.** Toute statistique — fréquentation, motifs, durées — suppose une ressaisie manuelle du registre.

5. **La traçabilité est faible.** Un registre manuscrit ne dit pas qui a autorisé une entrée, ni quand une décision a été modifiée.

6. **Les visites protocolaires ne sont pas outillées.** Les visites de personnalités, qui requièrent une validation hiérarchique, sont gérées par courriel, sans trace structurée.

7. **Le visiteur ne peut rien préparer à l'avance.** Tout se joue au guichet : ni identité, ni motif, ni consignes de sécurité ne peuvent être traités avant l'arrivée, ce qui reporte l'intégralité du contrôle sur le moment le moins favorable pour l'exercer.

1.2.3 Justification d'un développement interne

L'étude de l'existant a conduit à privilégier une solution développée sur mesure. Ce choix s'explique par trois facteurs : l'hébergement des données personnelles sur l'infrastructure du groupe, l'intégration native à l'annuaire et à l'authentification unique existants, et la spécificité du processus d'induction sécurité, propre à l'industrie cimentière. Il permet également d'appliquer un cloisonnement multi-sites dès la conception et de faire évoluer les règles métier sans dépendre du calendrier d'un éditeur externe.

1.3 Cahier des charges

1.3.1 Besoin exprimé

Le besoin, tel qu'il a été formulé lors des entretiens de cadrage, tient en une phrase : *savoir à tout moment qui est présent sur chaque site, garantir que chaque personne présente a été formée aux consignes de sécurité, et pouvoir le prouver*.

1.3.2 Périmètre fonctionnel

Le périmètre retenu couvre les domaines suivants.

Table 1.2 – Périmètre fonctionnel du projet

Domaine	Contenu
Parcours visiteur	Pré-déclaration à distance, déclaration spontanée par code QR, passeport visiteur
Induction sécurité	Diffusion de contenu, quiz évalué côté serveur, certificat vérifiable
Parcours VIP	Déclaration par un assistant, validation hiérarchique, induction dédiée
Opérations guérite	Entrée, sortie, rattrapage, saisie manuelle, liste des présents, évacuation
Administration	Sites et leurs guérites, référentiels, utilisateurs, profils, habilitations, modèles de courriels
Traçabilité	Journal d'audit de toutes les actions modifiant l'état du système
Restitution	Tableaux de bord, statistiques par site, rapports planifiés

1.3.3 Hors périmètre

Ne font pas partie du projet : le contrôle d'accès physique (tourniquets, barrières), la gestion des badges d'employés permanents, la vidéosurveillance, et la gestion des livraisons de marchandises.

1.3.4 Contraintes

Le projet s'inscrit sous plusieurs contraintes fixées par l'entreprise :

- **Hébergement sur site.** L'application doit être déployable sur l'infrastructure Kubernetes interne, sans dépendance à un service cloud externe.
- **Séparation des réseaux.** Le portail visiteur est exposé sur Internet ; le portail back-office reste sur le réseau interne.
- **Authentification unique.** Les utilisateurs internes s'authentifient via le fournisseur d'identité du groupe.
- **Langue.** L'interface d'administration est en français ; le portail visiteur est bilingue français et anglais.

1.4 Méthodologie et conduite de projet

1.4.1 Démarche adoptée

Le projet a été conduit selon une démarche agile, itérative et incrémentale inspirée de Scrum. Le backlog comportait 65 tâches Jira, réparties en 9 epics et 56 récits utilisateurs. La réalisation a été organisée en itérations d'une semaine, chacune associant conception, développement, tests et démonstration. Une revue hebdomadaire avec l'encadrant permettait de recueillir des corrections fonctionnelles, des améliorations d'interface, des ajustements de sécurité et de nouvelles orientations.

Le projet ayant été réalisé par un seul développeur, les rôles Scrum n'ont pas été formalisés comme dans une équipe complète. Monsieur Omar Naciri représentait le besoin et validait les incréments, Monsieur Omar Bassiry arbitrait les questions d'architecture, tandis que je prenais en charge la conception, la réalisation et les tests.

Le suivi dans Jira. Le backlog a été tenu dans Jira, où chaque récit porte un identifiant `GV-XX` qui l'accompagne jusqu'au bout : il figure dans les messages de validation du dépôt, dans les commentaires du code là où une règle métier est subtile, et dans les diagrammes de cas d'utilisation du chapitre suivant. Cette continuité a une valeur pratique immédiate — devant un comportement inattendu, on remonte du code au récit qui l'a demandé, et l'on retrouve le critère d'acceptation d'origine plutôt que de reconstituer l'intention.

Les récits étaient regroupés en neuf epics correspondant aux grands domaines fonctionnels, et affectés aux sprints selon leurs dépendances plutôt que selon leur intérêt : le parcours visiteur ne pouvait pas commencer avant l'authentification et les référentiels, et les restitutions statistiques supposaient l'existence de visites à agréger.

Le classeur de suivi. En complément de Jira, un classeur Excel partagé avec l'encadrant tenait la vue de charge : estimation par récit, avancement, et écart entre la charge restante et la capacité disponible. Ce document n'a pas un rôle décoratif — c'est lui qui a rendu visible, en milieu de projet, que le périmètre initial dépassait la capacité, et qui a permis d'arbitrer explicitement avec l'encadrant plutôt que de découvrir le dépassement en fin de parcours. C'est également ce classeur qui porte le facteur de compression de charge discuté plus bas.

Le rythme de validation. Chaque fin de semaine donnait lieu à une démonstration de l'incrément à l'encadrant, sur l'application réellement exécutée et non sur des maquettes. Ces revues ont produit trois types de retours : des corrections fonctionnelles — un cas d'usage mal interprété —, des ajustements d'interface issus de la connaissance du terrain, et des exigences de sécurité qui n'étaient pas formulées dans le backlog initial. Plusieurs décisions d'architecture consignées dans le dépôt sont nées de ces échanges.

1.4.2 Le développement agentique

Le titre de ce rapport annonce un *développement agentique*. Le terme mérite d'être défini, car il ne désigne pas le simple recours à un outil de complétion de code.

De la complétion à l'agent. Un assistant de complétion propose la suite de la ligne en cours ; il ne voit qu'un fichier et ne peut rien exécuter. Un **agent** de développement — ici **Claude Code** [22], un outil en ligne de commande — travaille sur le dépôt entier : il lit les fichiers qu'il juge utiles, en écrit de nouveaux, lance la compilation, exécute les tests, lit les erreurs obtenues et corrige en conséquence. La différence n'est pas de degré mais de nature : on ne lui dicte pas du code, on lui confie une tâche.

Ce que cela change dans la conduite du travail. La charge de l'ingénieur se déplace. Écrire la centième requête paginée ou le énième formulaire validé cesse d'être le goulot d'étranglement ; ce qui le devient, c'est de *spécifier avec précision* et de *vérifier sans complaisance*. Un agent produit vite du code plausible : plausible n'est pas correct. L'essentiel du temps s'est donc reporté sur la formulation des tâches, la relecture des différences et l'exécution des tests.

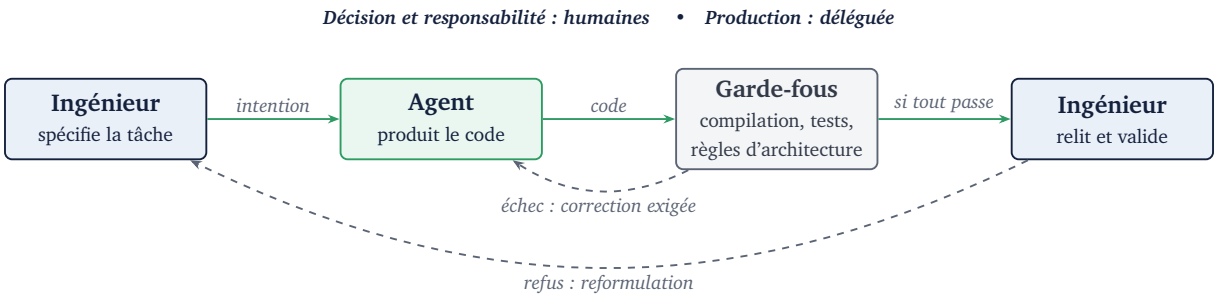


Figure 1.1 – Cycle de travail avec un agent : la production est déléguée, la décision et la preuve ne le sont pas

Le cycle de la figure 1.1 met en évidence ce qui distingue cette pratique d'une simple accélération. Deux boucles de retour y figurent, et ce sont elles qui portent la qualité : celle des *garde-fous*, automatique et impitoyable, qui renvoie le code à l'agent tant qu'une règle est enfreinte ; et celle de la *revue*, humaine, qui renvoie à la reformulation quand le résultat est correct mais ne répond pas au besoin. Un agent sans la première boucle produit de la dette technique à grande vitesse ; sans la seconde, il produit avec application la mauvaise fonctionnalité.

Un dépôt conçu pour être compris par un agent. La conséquence la plus structurante est architecturale. Le dépôt porte un manuel destiné à l'agent — un fichier `CLAUDE.md` de près de quatre cents lignes — qui énonce les contraintes non négociables, oriente chaque type de tâche vers la procédure correspondante, et recense l'état réel du projet. Autour de lui s'organisent **36 règles** de convention, une par fichier, chacune énonçant une règle, sa justification et son mécanisme de contrôle ; **34 procédures** documentées pour les gestes récurrents — créer une entité, écrire une migration, ajouter un contrôle de permission ; et **8 agents spécialisés**, chacun cantonné à un périmètre d'écriture précis, de sorte que deux tâches menées en parallèle sur deux modules différents ne puissent pas se marcher dessus.

Ce dispositif a une contrepartie sur la conception elle-même. Une règle du projet énonce que, *lorsque deux options d'architecture sont également correctes, on retient celle dont l'empreinte cognitive est la plus faible pour un agent*. Le monolithe modulaire, la machine à états unique et le cloisonnement placé dans la couche de persistance découlent en

partie de ce critère : ce sont des structures où la règle est écrite à un seul endroit, donc difficiles à enfreindre par inadvertance.

Les garde-fous, cœur du dispositif. Faire produire du code par un agent n'est acceptable que si l'on dispose d'un moyen mécanique de refuser ce qui viole les conventions. C'est le rôle des **tests d'architecture** décrits au chapitre 5 : treize règles inspectent le code lui-même et font échouer la compilation lorsqu'un module en atteint un autre par l'intérieur, lorsqu'un point d'entrée oublie de déclarer sa permission, ou lorsqu'une nouvelle table ne déclare pas son rattachement à un site. Une convention qu'un relecteur peut laisser passer devient une erreur de compilation, immédiate et non négociable. Les **23 décisions d'architecture** consignées dans le dépôt jouent le même rôle sur les choix : elles empêchent qu'une décision prise en semaine deux soit défaite en semaine six faute d'en avoir gardé la raison.

Ce que l'agent n'a pas fait. La délimitation est nette. Les choix d'architecture, l'interprétation du besoin, les arbitrages de priorité et la validation fonctionnelle sont restés humains. L'agent n'a jamais été autorisé à décider seul d'une évolution du modèle de données ni d'une règle de sécurité. Surtout, aucune de ses affirmations n'a été retenue sans preuve : l'expérience de ce projet est qu'un agent annonce volontiers comme terminé ce qui ne l'est pas. Un exemple concret l'illustre — l'outil de compilation réutilise le résultat d'une suite de tests inchangée et affiche un succès quasi instantané ; pris au mot, ce succès aurait laissé croire que les tests avaient été rejoués. La règle adoptée a donc été de ne jamais se fier au compte rendu, mais de lire la différence de code et de relancer les vérifications soi-même.

Résultat mesuré. Le planning estimait à 1,6 le facteur de compression de charge apporté par cette assistance, soit 62 jours de charge brute ramenés à 38,75 jours pour une capacité de 40 jours sur huit semaines.

Ce que ce chiffre vaut, et ce qu'il ne vaut pas. Il est important d'être honnête sur son statut : c'est une **estimation de planification**, établie a priori pour dimensionner le backlog, et non le résultat d'un protocole de mesure. Aucune comparaison contrôlée n'a été conduite — il aurait fallu réaliser le même périmètre deux fois, avec et sans assistance, ce que le calendrier d'un stage ne permet pas. Les heures effectives, les propositions rejetées, le temps de relecture et les reprises n'ont pas été chronométrés séparément.

Le seul indicateur réellement vérifiable est celui du dépôt : le périmètre décrit dans ce rapport a été produit dans le calendrier des huit sprints, avec la couverture de tests présentée au chapitre 5. Ce constat est cohérent avec le facteur estimé, sans le démontrer.

Une remarque de portée générale mérite en revanche d'être faite : le gain observé tient autant au dispositif qu'à l'outil. Sur un projet dont les conventions sont écrites, les frontières vérifiées automatiquement et les tests nombreux, la vitesse se convertit en avancement. Sur un dépôt sans garde-fous, elle produirait surtout de la dette plus rapidement.

1.4.3 Outils de travail

Table 1.3 – Outils utilisés durant le projet

Usage	Outil
Suivi des tâches	Jira — récits utilisateurs numérotés GV-1 à GV-65
Gestion de versions	Git, hébergé sur GitLab d'entreprise
Intégration continue	GitLab CI
Développement	IntelliJ IDEA, Visual Studio Code
Assistance agentique	Claude Code
Modélisation	Draw.io et PowerPoint
Conteneurisation	Docker et Docker Compose
Test d'API	Swagger UI, curl

1.4.4 Planification et état d'avancement

Ce que ce rapport décrit, et à quelle date. Ce document rend compte de l'état du projet au **30 août 2026**, terme du stage. Celui-ci s'est étendu du 1er juillet au 30 août 2026 ; la réalisation a été conduite sur huit sprints hebdomadaires, du 1er juillet au 28 août.

Cette précision n'est pas une formalité : elle conditionne la lecture de tous les résultats présentés plus loin. Les mesures de tests, les captures d'écran et les états de la chaîne d'intégration se rapportent tous à la même version du code, celle du 30 août 2026.

Le dépôt en porte la trace vérifiable : **161 validations** sur la branche principale, de la première à celle retenue pour ce rapport.

Table 1.4 – Planification de la réalisation, sprints du 1er juillet au 28 août 2026

Sprint	Période	Objectif principal
S1	01/07–10/07	Socle, infrastructure et primitives réutilisables
S2	13/07–17/07	Authentification, sécurité et gestion des identités
S3	20/07–24/07	Administration globale et référentiels
S4	27/07–31/07	Parcours du visiteur ordinaire
S5	03/08–07/08	Parcours VIP, documents et notifications
S6	10/08–14/08	Tableaux de bord, statistiques et rapports
S7	17/08–21/08	Recette métier et déploiement en environnement cible
S8	24/08–28/08	Stabilisation, durcissement et documentation

L'ordonnancement suit les dépendances plutôt que les préférences : le parcours visiteur ne pouvait pas commencer avant que l'authentification et les référentiels ne soient en place, et les restitutions statistiques supposaient l'existence de visites à agréger.

Le classeur de suivi daté du 4 août constituait une photographie intermédiaire du projet. Au terme du stage, les cinq lots fonctionnels sont intégrés et le périmètre retenu est livré ; l'état exact, élément par élément, figure en section 4.7. Le déploiement dans l'environnement de recette de Holcim Maroc et la recette métier ont été conduits avec l'appui de l'encadrant professionnel.

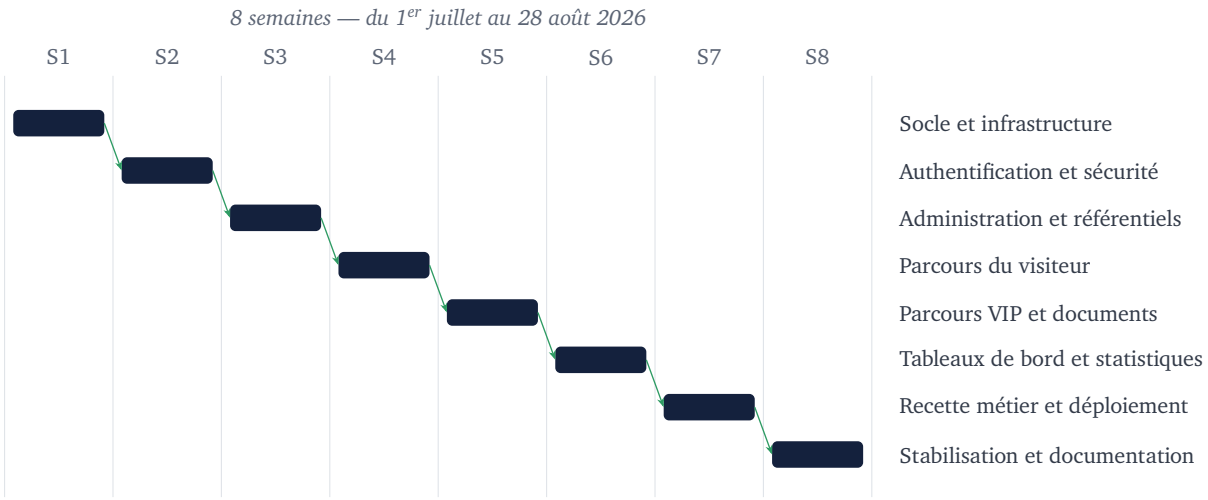


Figure 1.2 – Ordonnancement des huit sprints et enchaînement de leurs dépendances

Conclusion

Ce chapitre a établi que le processus d'accueil actuel, fondé sur le registre papier, ne permet ni de connaître la présence en temps réel ni de garantir l'induction sécurité. Le chapitre suivant traduit ce constat en besoins formalisés.

CHAPITRE 2

Analyse et spécification des besoins

Introduction

> *Ce chapitre identifie les acteurs du système, formalise les besoins fonctionnels sous forme de cas d'utilisation, et énonce les exigences non fonctionnelles qui contraindront la conception.*

2.1 Identification des acteurs

L'analyse a fait apparaître deux populations d'utilisateurs fondamentalement distinctes, qui ne s'authentifient pas de la même manière et n'accèdent pas au même portail.

2.1.1 La population visiteur

Le visiteur est un utilisateur externe, non salarié, qui n'a pas de compte dans le système d'information du groupe. Il accède au portail public depuis son téléphone ou son ordinateur personnel. Son authentification repose sur un code à usage unique reçu par courriel, et non sur un mot de passe : créer un compte pour une visite ponctuelle serait une friction inutile.

2.1.2 La population back-office

Les utilisateurs internes s'authentifient par le fournisseur d'identité du groupe. Six profils ont été identifiés, dont les responsabilités sont décrites dans le tableau 2.1.

Table 2.1 – Acteurs du système et leurs responsabilités

Acteur	Responsabilités
Visiteur	Se pré-déclare, suit l'induction, obtient son passeport, consulte ses visites
Agent de sécurité	Enregistre les entrées et sorties en guérite, consulte les présents, déclenche l'évacuation
Responsable sûreté	Administre son site, ses référentiels, valide les visites VIP, exploite les statistiques
Directeur de site	Consulte l'activité de son site, valide les visites VIP de niveau supérieur
Directeur sûreté	Consulte l'activité de tous les sites, exploite les rapports consolidés
Déclarant VIP	Déclare une visite protocolaire pour le compte d'un dirigeant
Administrateur	Gère les sites, les comptes, les profils, les habilitations et la configuration

2.2 Besoins fonctionnels

Les besoins fonctionnels ont été formalisés en sept diagrammes de cas d'utilisation, un par domaine, plutôt qu'en un seul diagramme d'ensemble : le périmètre couvre sept familles d'acteurs et une soixantaine de cas, qu'un diagramme unique aurait rendus illisibles. Chaque cas porte la référence du récit utilisateur correspondant, ce qui permet de remonter du diagramme au backlog puis au code.

2.2.1 Les acteurs et leurs familles

Ce premier diagramme pose le vocabulaire. Il distingue trois ensembles que rien ne permet de confondre par la suite.

Les **acteurs de la zone interne** sont les salariés : l'agent de sécurité en guérite, le responsable sûreté qui administre son site, le directeur de site, le directeur sûreté qui a la vision de tous les sites, le déclarant VIP — souvent une assistante de direction — et l'administrateur. Ils héritent tous d'un acteur générique, *utilisateur de la zone*

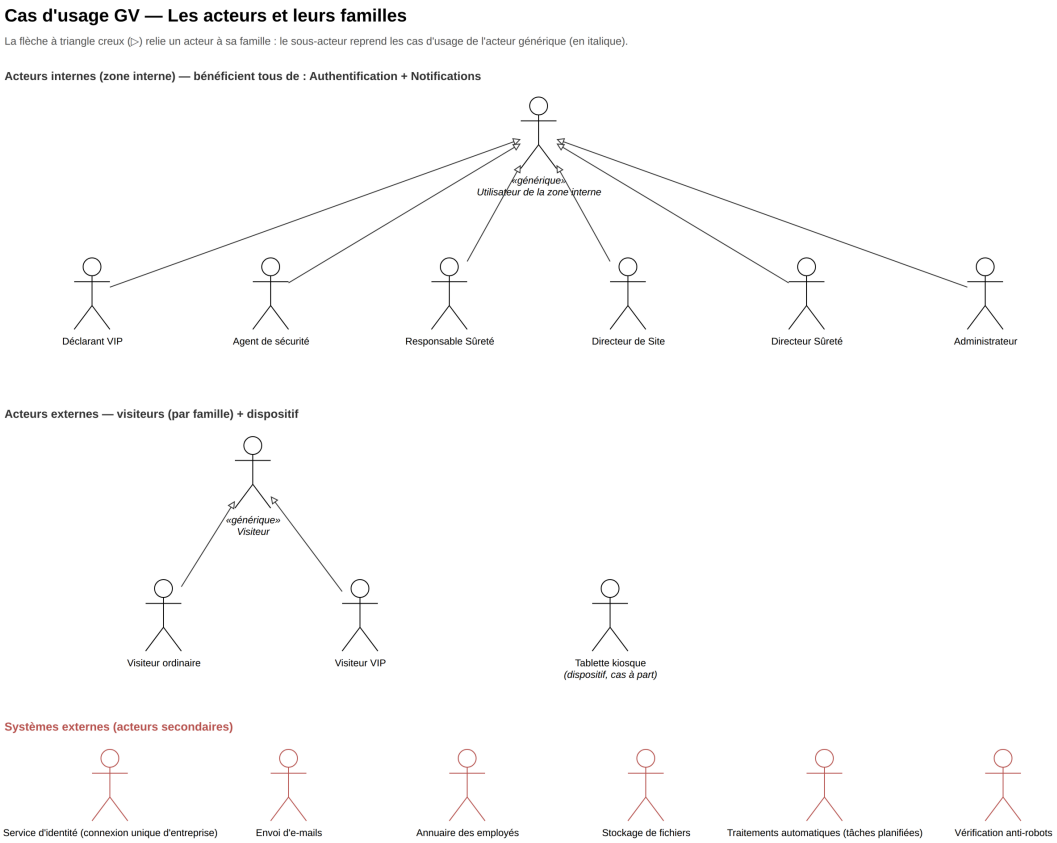


Figure 2.1 – Acteurs du système, regroupés par famille

interne, porteur de ce qui leur est commun : l'authentification par le service d'identité de l'entreprise.

Les **acteurs externes** sont les visiteurs, eux-mêmes déclinés en visiteur ordinaire et visiteur VIP. Ils n'ont pas de compte : le visiteur ordinaire est identifié par son adresse électronique et un code à usage unique, le visiteur VIP par le lien reçu pour son induction. C'est ce qui les sépare des acteurs internes, et non le type d'appareil depuis lequel ils se connectent.

Les **systèmes externes** apparaissent enfin comme acteurs secondaires : le service d'identité d'entreprise, l'envoi de courriels, l'annuaire des employés, le stockage de fichiers, la vérification anti-robots, et les traitements automatiques. Les faire figurer comme acteurs plutôt que comme détails d'implémentation est délibéré : chacun est un point de dépendance dont l'indisponibilité a un effet fonctionnel identifiable.

2.2.2 Cas d'utilisation du visiteur ordinaire

Pré-déclaration. Le visiteur saisit à distance son identité, le motif de sa venue, la date prévue et la personne visitée. Le système enregistre une visite à l'état initial et lui adresse un passeport comportant un code QR. La pré-déclaration est *idempotente* : une double soumission, fréquente sur mobile, ne crée pas deux visites.

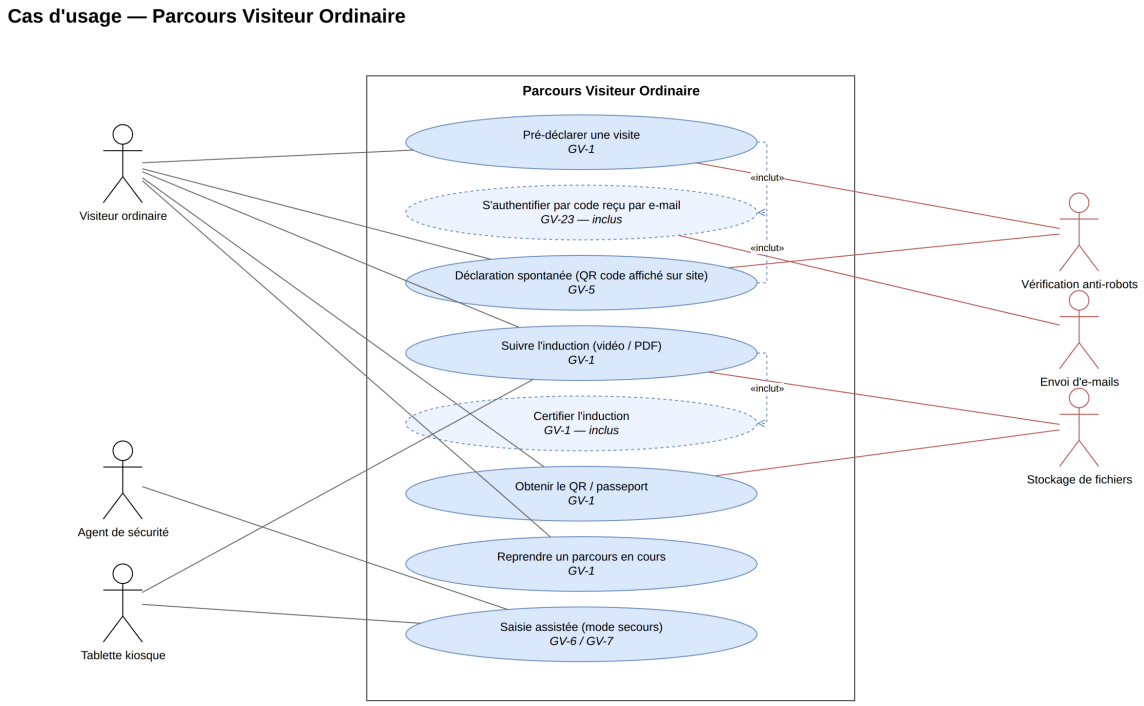


Figure 2.2 – Cas d'utilisation — parcours du visiteur ordinaire

Déclaration spontanée. Un visiteur arrivé sans annonce scanne le code QR affiché à l'entrée du site. Il accède au même formulaire, pré-renseigné du site concerné.

Induction sécurité. Deux parcours complémentaires existent. Dans le parcours ordinaire, le visiteur consulte le contenu du site — vidéo ou document — puis certifie l'avoir compris avant d'obtenir son passeport. Dans le parcours VIP, une évaluation complète cette induction. Le quiz est corrigé *côté serveur* : la projection des questions envoyée au navigateur ne contient pas l'indication de la bonne réponse, faute de quoi il suffirait d'inspecter la page pour tricher. La réussite délivre un certificat vérifiable par code QR.

Deux relations du diagramme méritent lecture. La certification de l'induction et l'authentification par code reçu par courriel sont des cas *inclus* : ils ne sont pas optionnels, ils font partie intégrante du parcours et ne peuvent pas être contournés. La *saisie assistée* en revanche est un mode de secours, déclenché par l'agent sur son propre poste

lorsque le visiteur ne peut pas mener le parcours lui-même — sans smartphone, sans réseau, ou simplement en difficulté devant l'écran. Prévoir ce mode dès l'analyse évite le sort habituel de ce genre de système : être contourné le jour où il ne couvre pas le cas réel.

2.2.3 Cas d'utilisation du visiteur protocolaire

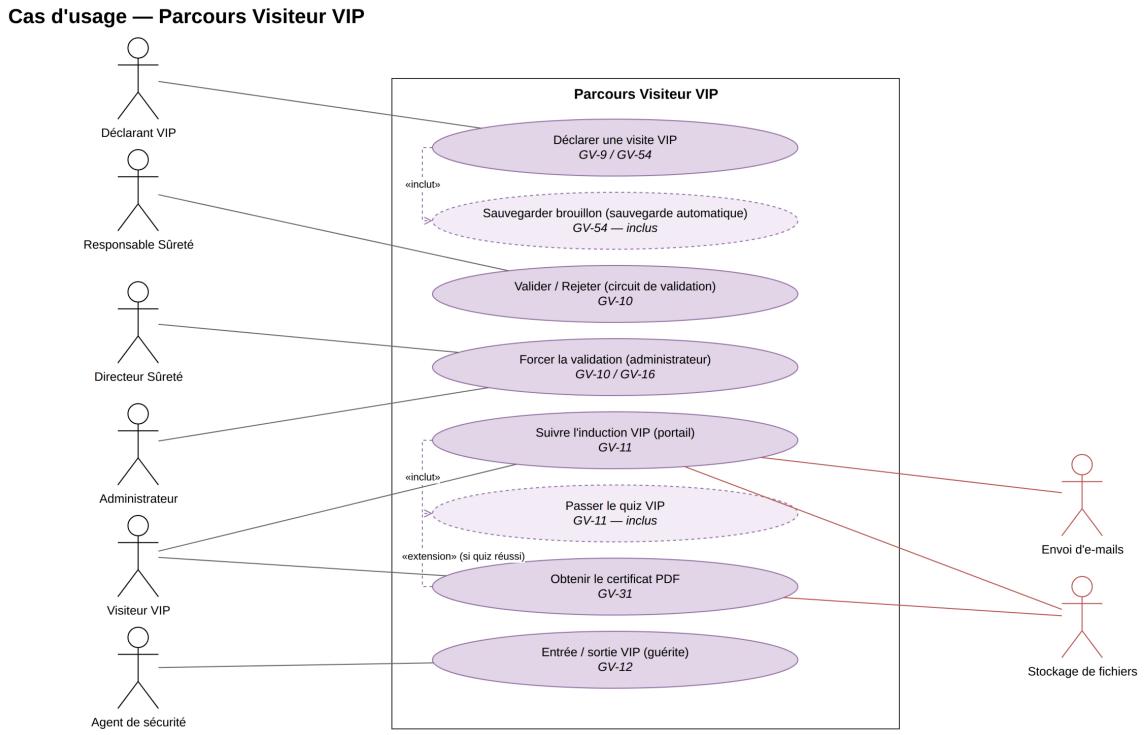


Figure 2.3 – Cas d'utilisation — parcours du visiteur VIP

Le parcours VIP se distingue par l'intervention de trois acteurs supplémentaires. Le **déclarant** — typiquement une assistante de direction — remplit la visite pour le compte du dirigeant attendu ; le formulaire enregistre automatiquement un brouillon, parce qu'une déclaration de délégation se remplit rarement d'une traite. Le dossier passe ensuite en **validation** : le responsable sûreté valide ou rejette, un rejet exigeant obligatoirement un motif. L'administrateur dispose d'une validation forcée, réservée aux situations où la chaîne hiérarchique n'est pas joignable dans le temps disponible — capacité sensible, donc tracée.

Le visiteur VIP suit ensuite une induction dédiée sur le portail, avec son quiz. La relation d'*extension* portée par le diagramme est significative : l'obtention du certificat PDF n'a lieu *que si le quiz est réussi*. Le certificat n'est pas un document remis d'office, c'est la preuve d'un résultat.

2.2.4 Cas d'utilisation de l'agent de sécurité

Entrée en guérite. L'agent scanne le code du visiteur ou le recherche par son nom. Le système vérifie que l'induction est acquise et que la visite est autorisée, puis enregistre l'entrée sur la guérite choisie. L'agent peut y joindre le numéro du badge physique remis au visiteur ; c'est une information de traçabilité, jamais une condition.

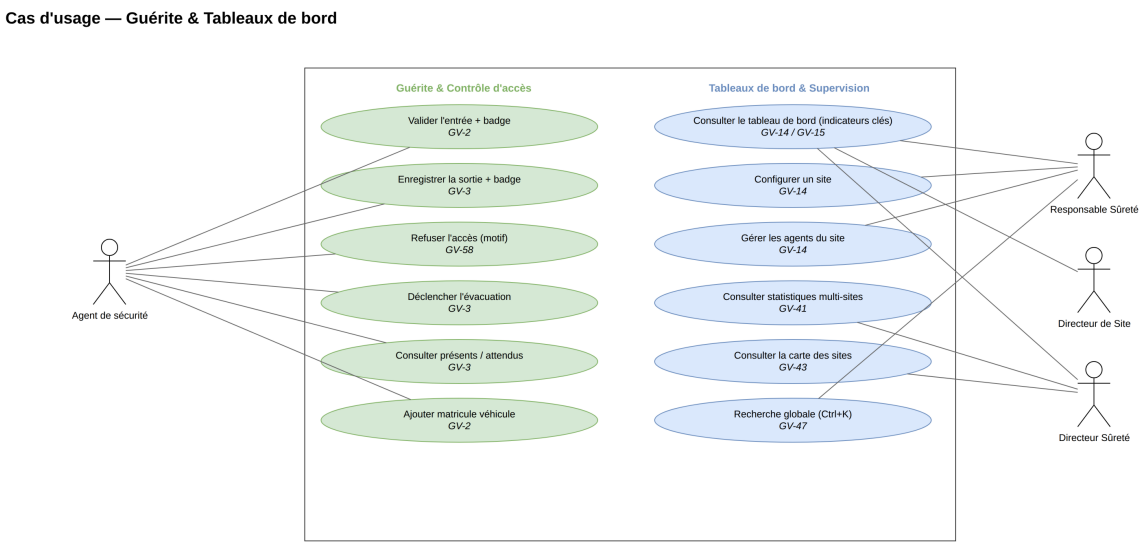


Figure 2.4 – Cas d'utilisation — guérite et tableaux de bord

Sortie et rattrapage. L'agent enregistre la sortie du visiteur. Le rattrapage permet de régulariser une visite dont l'entrée n'a pas été saisie sur le moment.

Évacuation. Sur déclenchement, le système produit la liste des personnes présentes sur le site et notifie les responsables.

Refus d'accès. L'agent peut refuser une entrée, en enregistrant le motif. Ce cas est aussi important que la validation : sans lui, un refus resterait une décision orale, invérifiable, et le registre ne dirait pas pourquoi une personne attendue n'est jamais entrée.

Le même diagramme porte les **tableaux de bord**, parce qu'ils s'adressent aux mêmes données vues à des altitudes différentes : le responsable sûreté suit son site et le configure, le directeur de site en consulte les indicateurs, le directeur sûreté consolide plusieurs sites et dispose d'une carte. La recherche globale, accessible au clavier, traverse ces surfaces — un agent qui cherche un nom ne doit pas avoir à savoir dans quel écran il se trouve.

2.2.5 Cas d'utilisation de l'administrateur

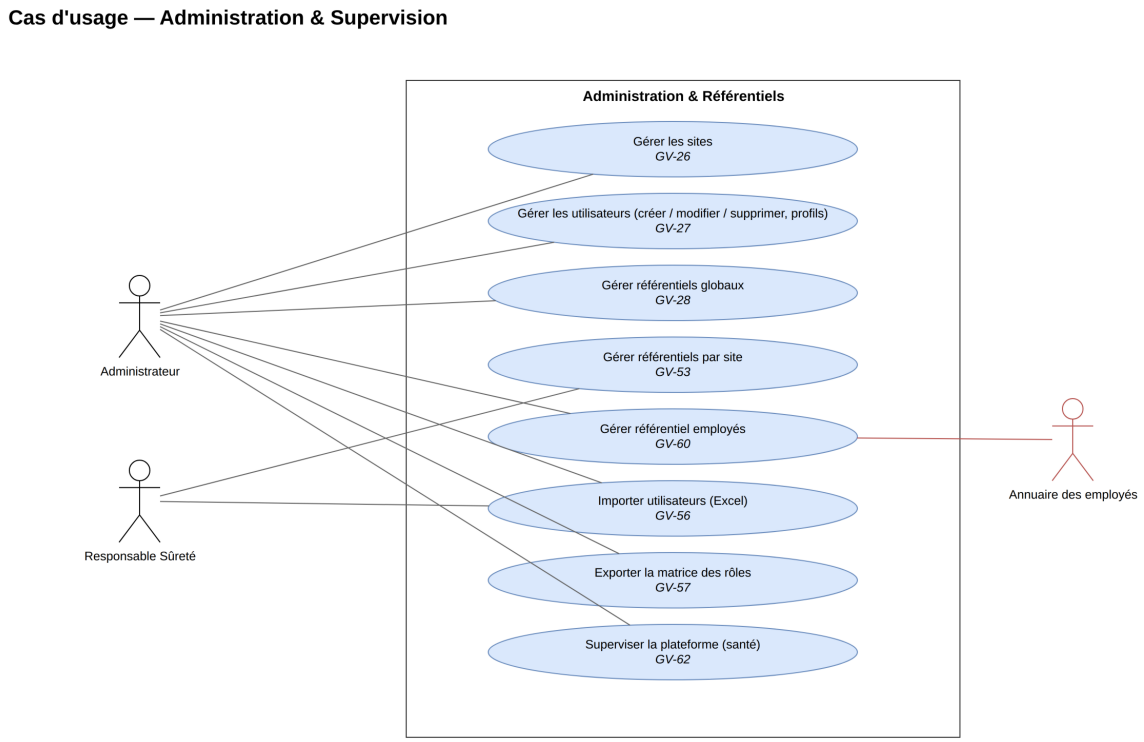


Figure 2.5 – Cas d'utilisation — administration et référentiels

L'administration ne crée pas de valeur métier directement : elle conditionne tout le reste. Les sites, les utilisateurs et leurs profils, les référentiels globaux — pays, motifs de visite, types de visiteur — et les référentiels propres à chaque site y sont gérés. Deux cas méritent l'attention parce qu'ils traduisent une réalité organisationnelle : l'**import Excel** des utilisateurs, parce qu'une reprise initiale ne se fait pas à la main pour plusieurs centaines de comptes, et l'**export de la matrice des rôles**, parce que la revue périodique des habilitations est une obligation de contrôle interne qui exige un document communicable.

2.2.6 Authentification, sécurité et notifications

Ce diagramme rassemble ce qui traverse tous les parcours. Les deux voies d'authentification y apparaissent côte à côte : la connexion unique d'entreprise pour les salariés, avec une voie de secours par identifiant et mot de passe pour le cas où le service d'identité serait indisponible, et le code reçu par courriel pour les visiteurs. La déconnexion automatique après inactivité, le renouvellement de mot de passe, la cloche de notifications, le paramétrage de l'envoi de courriels, celui des modèles de messages et la configuration des rapports planifiés complètent l'ensemble.

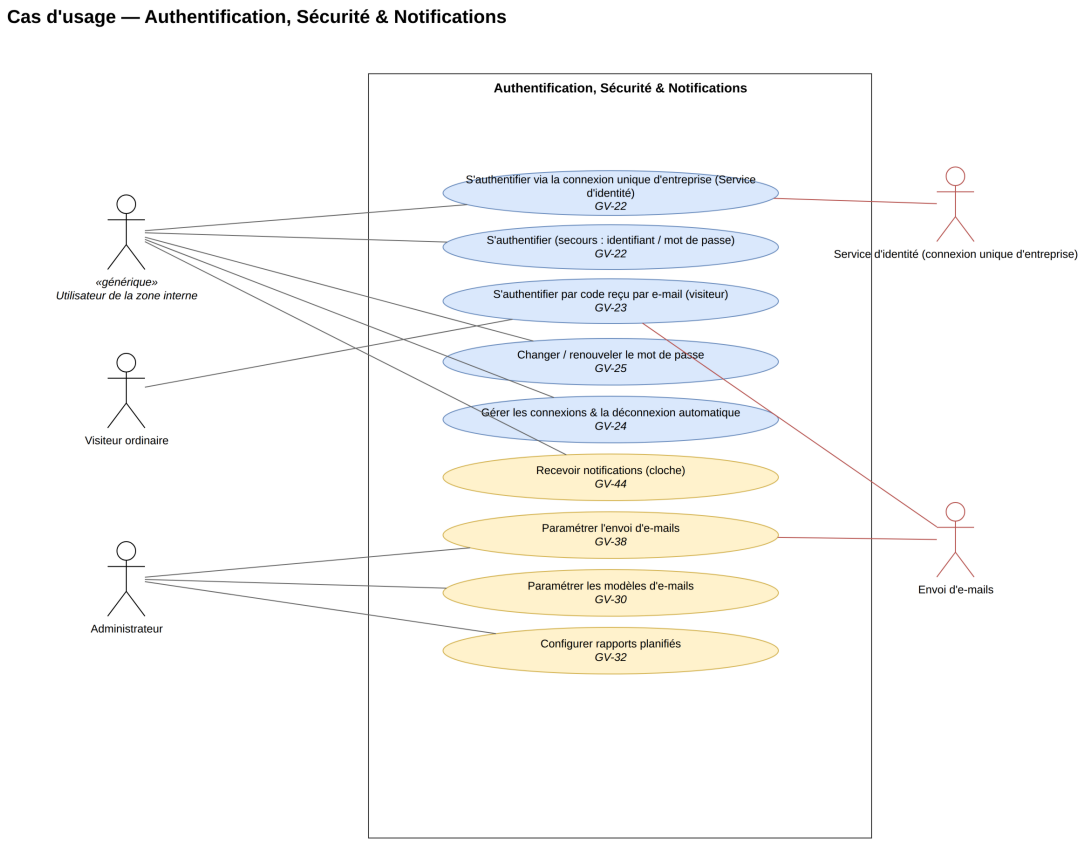


Figure 2.6 – Cas d'utilisation — authentification, sécurité et notifications

2.2.7 Conformité et traitements automatiques

Ce dernier diagramme regroupe deux familles souvent oubliées d'une analyse fonctionnelle, et pourtant décisives ici.

La **conformité** d'abord : consultation des journaux d'audit, historique d'une visite, configuration de la conservation et de la suppression des données au titre de la loi 09-08, gestion du consentement, exercice de leurs droits par les visiteurs, et gestion des langues. Ces cas ne servent aucun utilisateur au quotidien ; ils servent l'entreprise le jour d'un contrôle.

Les **traitements automatiques** ensuite, dont la particularité est de n'avoir aucun acteur humain déclencheur : désactivation des comptes restés inactifs, synchronisation de l'annuaire des employés, génération et envoi des rapports planifiés, vérification des seuils d'alerte — par exemple une présence anormalement longue sur un site. Les représenter comme des cas d'utilisation à part entière, avec un acteur *traitements automatiques*, évite qu'ils ne se dissolvent dans l'implémentation : ce sont des comportements attendus du système, donc spécifiables et testables.

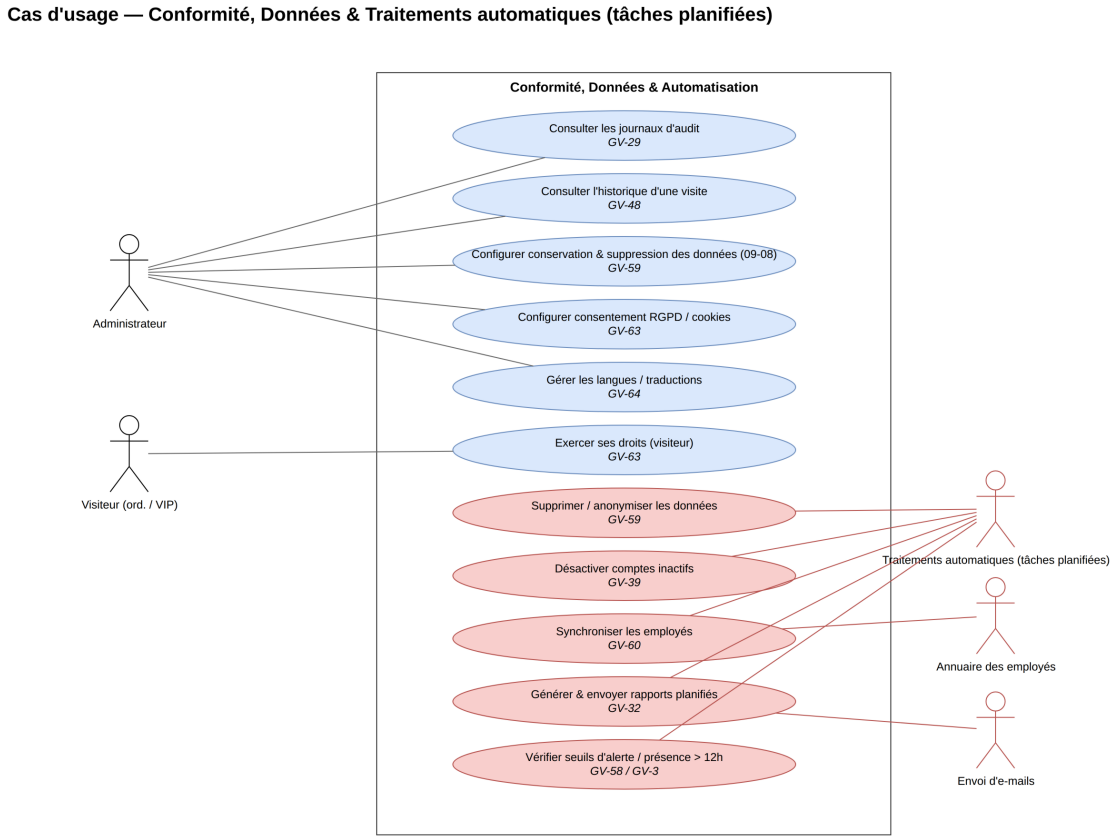


FIGURE 2.7 – Cas d'utilisation — conformité, données et automatisation

2.2.8 Récits utilisateurs

Le besoin a été formalisé en 56 récits utilisateurs numérotés, regroupés en 9 epics — soit les 65 éléments du backlog présentés au chapitre 1. Le tableau 2.2 en présente un extrait représentatif.

Table 2.2 – Extrait des récits utilisateurs

Réf.	Récit
GV-1	En tant que visiteur, je veux me pré-déclarer et obtenir mon passeport, afin de garantir ma conformité sécurité avant mon arrivée.
GV-2	En tant qu'agent, je veux enregistrer une entrée en scannant un code, afin de fluidifier le passage en guérite.
GV-3	En tant qu'agent, je veux consulter les présents et déclencher une évacuation, afin de sécuriser le site.
GV-9	En tant que déclarant, je veux déclarer une visite VIP, afin de la soumettre à validation.
GV-11	En tant que visiteur VIP, je veux suivre une induction dédiée, afin d'obtenir mon certificat.
GV-26	En tant qu'administrateur, je veux gérer les sites et leurs paramètres, afin d'adapter le système à chaque implantation.
GV-61	En tant que système, je veux une machine à états unique, afin que le cycle de vie d'une visite soit cohérent.

2.3 Besoins non fonctionnels

Table 2.3 – Exigences non fonctionnelles

Catégorie	Exigence
Sécurité	Authentification unique pour les internes, code à usage unique pour les visiteurs ; autorisation par permission sur chaque point d'entrée de l'API.
Cloisonnement	Un utilisateur ne voit que les données des sites de son périmètre. Une ressource hors périmètre est *introuvable*, non *interdite*.
Traçabilité	Toute action modifiant l'état du système laisse une trace horodatée et attribuée.
Ergonomie	Le portail visiteur est conçu pour mobile d'abord ; les écrans de guérite adoptent un rendu tactile, cibles d'au moins 56 pixels, lorsque le site déclare équiper ses agents de tablettes.
Performance	Les listes sont paginées côté serveur, avec une taille de page plafonnée.
Disponibilité	L'application est déployée en plusieurs instances derrière un répartiteur de charge.
Maintenabilité	Les frontières entre modules sont vérifiées automatiquement à la compilation.

2.3.1 Le cloisonnement, exigence structurante

Une exigence mérite d'être isolée, car elle a orienté toute l'architecture. Un responsable sûreté de Settat ne doit accéder à aucune donnée de Casablanca. Une réponse naïve consisterait à ajouter un filtre sur chaque requête ; l'expérience montre qu'un tel filtre finit toujours par être oublié quelque part, et que l'oubli est invisible en revue comme en test tant que l'environnement ne comporte qu'un seul site.

Le choix retenu est donc d'appliquer le cloisonnement *au niveau de la couche de persistance* : chaque table porte explicitement sa position — rattachée à un site, ou globale — et le filtre est appliqué automatiquement à toute lecture. Une vérification automatisée refuse la compilation si une nouvelle table ne déclare pas sa position.

Un corollaire important en découle. Lorsqu'un utilisateur demande une ressource située hors de son périmètre, le système répond « **non trouvé** » et non « **accès interdit**

». Répondre « interdit » révélerait en effet l'existence de la ressource, et permettrait à un utilisateur d'énumérer l'activité des autres sites sans jamais en lire le contenu.

Conclusion

Les besoins étant formalisés, le chapitre suivant expose les choix de conception qui y répondent.

CHAPITRE 3

Conception

Introduction

> *Ce chapitre présente l'architecture de la solution, le modèle de données, la modélisation du cycle de vie d'une visite, et les mécanismes de sécurité.*

3.1 Architecture générale

3.1.1 Diagramme de composants

La solution se compose de deux applications web distinctes servies par une API unique, elle-même adossée à une base de données relationnelle et à un stockage d'objets.

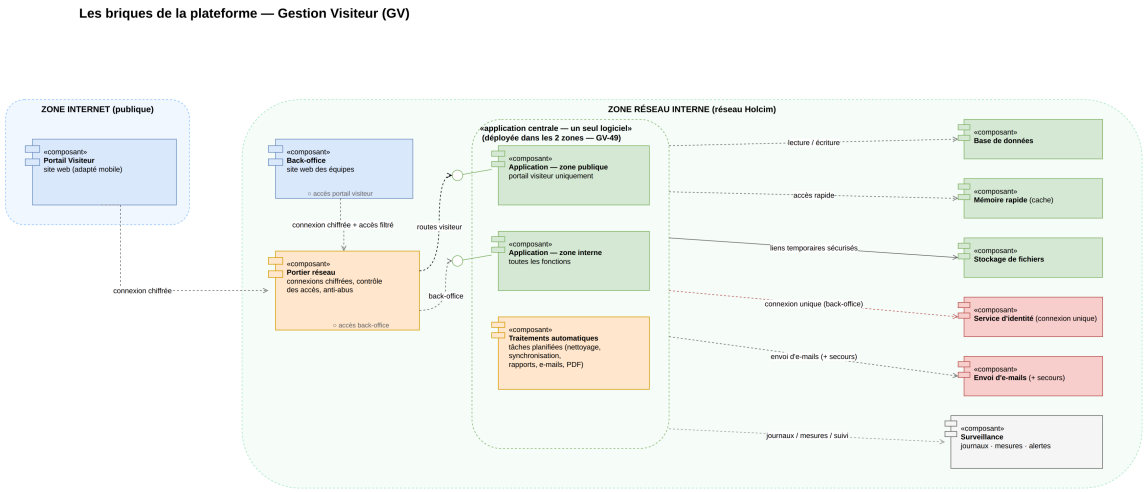


Figure 3.1 – Diagramme de composants de la plateforme

La lecture du diagramme se fait de gauche à droite, et sa division la plus importante est celle des **deux zones réseau**.

À gauche, la **zone Internet** ne contient qu'une chose : le portail visiteur. C'est la seule surface qu'une personne extérieure peut atteindre. À droite, la **zone interne** du réseau Holcim abrite tout le reste — le back-office, l'application serveur et les services d'infrastructure.

Entre les deux se tient un **portier réseau** par lequel tout passe : il termine les connexions chiffrées, filtre les accès et applique les protections contre les abus. Le back-office n'est joignable que depuis l'intérieur ; le portail visiteur est publié, mais rien de ce qu'il appelle ne l'est directement.

Le point le plus notable est au centre : l'application est **un seul logiciel**, dont le schéma prévoit un déploiement en deux instances — l'une ne desservant que le portail visiteur, l'autre offrant toutes les fonctions. Le code serait identique, le périmètre exposé ne le serait pas. L'intérêt est double : ne pas écrire deux applications, avec le risque qu'une règle de sécurité corrigée dans l'une soit oubliée dans l'autre, et empêcher qu'une faille du portail public n'ouvre l'accès aux fonctions d'administration.

Ce qui est déployé aujourd'hui. Il faut distinguer cette cible du livré. **Le chart de déploiement décrit une seule instance du serveur**, que les deux portails appellent — c'est la topologie du schéma de déploiement présenté au chapitre 5, qui fait foi pour l'existant. Le dédoublement en deux instances suppose une segmentation réseau entre zones que l'infrastructure cible doit fournir, et relève donc de la mise en production. Dans l'état livré, c'est le portier réseau, et non la duplication de l'application, qui sépare les deux surfaces.

Un troisième composant, les **traitements automatiques**, porte les tâches planifiées : nettoyage, synchronisation, rapports, envois de courriels et production de documents. Le schéma les distingue pour rendre visible que le système agit aussi sans utilisateur ; dans le livré, ils s'exécutent au sein de l'application principale.

À droite figurent enfin les services d'infrastructure — base de données, mémoire rapide, stockage de fichiers, service d'identité, envoi de courriels, surveillance. Aucun n'est joignable depuis un navigateur : ils ne sont atteints que par l'application.

3.1.2 Le choix du monolithe modulaire

Le découpage en microservices a été écarté [3]. Un tel découpage aurait imposé, pour une équipe réduite et un déploiement sur site, le coût d'exploitation d'un système distribué — orchestration, cohérence répartie, observabilité — sans bénéfice proportionné : rien dans le besoin n'exige que la guérite et l'administration soient déployées ou mises à l'échelle séparément.

Le **monolithe modulaire** [2, 3] conserve l'avantage du découpage sans en payer le prix opérationnel. L'application est un seul exécutable et une seule base de données, mais son code est découpé en modules dont les frontières sont réelles : un module ne peut appeler un autre qu'à travers une interface publique explicitement déclarée. Cette

règle n'est pas une convention documentaire ; elle est vérifiée à chaque compilation, et sa violation fait échouer la construction.

Sept modules portent le métier et cinq rendent des services techniques transverses. Le tableau 3.1 en précise la responsabilité.

Table 3.1 – Responsabilité de chaque module

Module	Responsabilité
Modules métier	
identity	Comptes, profils, permissions, jetons, code à usage unique
site	Sites, paramètres par site, référentiels propres au site
visite	Cycle de vie de la visite ordinaire, opérations de guérite, présence sur site
visite-vip	Cycle de vie de la visite protocolaire, validation, induction dédiée
referentiels	Catalogues globaux : pays, motifs, types de visite, annuaire des employés
notification	Notifications internes du back-office
statistiques	Agrégation et restitution, rapports planifiés
Modules techniques	
persistence	Accès aux données, cloisonnement par site, chiffrement des secrets
security	Authentification, autorisation, filtres HTTP
audit	Journal des actions, publication transactionnelle et écriture différée
blobstore	Stockage des fichiers : documents d'induction, certificats, photos
config	Paramètres d'exécution, envoi de courriels

3.1.3 Architecture interne d'un module

Chaque module suit la même organisation en trois espaces, ce qui rend sa lecture prévisible.

Seuls `api/` et `events/` sont visibles de l'extérieur. Tout le reste est scellé : un module voisin ne peut pas atteindre une entité de persistance ni un contrôleur. Cette contrainte est ce qui permet de faire évoluer l'implémentation d'un module sans rien casser ailleurs.

3.2 Modélisation du cycle de vie d'une visite

3.2.1 Une machine à états unique

Le cœur du domaine est le cycle de vie d'une visite. Il aurait été possible de le représenter par plusieurs indicateurs booléens — *induction faite*, *entré*, *sorti* — mais cette représentation autorise des combinaisons absurdes, comme une visite à la fois sortie et jamais entrée.

Le choix retenu est une **machine à états** : la visite occupe à tout instant exactement un état, et seules les transitions déclarées sont possibles. Toute évolution du dossier passe par une transition nommée, ce qui rend l'ensemble vérifiable et traçable.

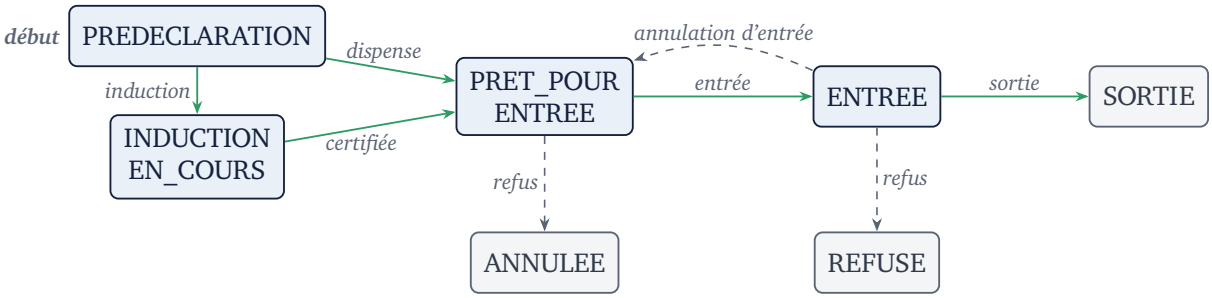


Figure 3.2 – Machine à états du cycle de vie d'une visite ordinaire

Le diagramme se lit de gauche à droite. Une visite naît à l'état `PREDECLARATION` dès que le visiteur s'est déclaré, à distance ou sur place par code QR. Elle passe en `INDUCTION_EN_COURS` lorsqu'il ouvre le contenu de sécurité, puis en `PRET_POUR_ENTREE` une fois l'induction certifiée — c'est le seul état depuis lequel l'agent de guérite peut enregistrer une entrée. Suivent `ENTREE`, tant que la personne est physiquement sur le site, puis `SORTIE` lorsque l'agent enregistre son départ. Deux états terminaux peuvent être atteints avant l'entrée : `REFUSE`, lorsque l'agent refuse l'accès, et `ANNULEE`, lorsque la visite est abandonnée. Aucune flèche ne revient depuis un état terminal : une visite close ne peut pas être réactivée.

La dispense d'induction, et pourquoi elle figure au diagramme. Une transition va directement de `PREDECLARATION` à `PRET_POUR_ENTREE`. Il serait faux d'écrire qu'aucun chemin ne contourne l'induction : ce chemin existe, et il est délibéré. Deux situations l'imposent. Le site peut ne pas exiger d'induction, ce qui est un paramètre de sa configuration. Et certaines catégories de visiteurs — les services de secours au premier chef — ne peuvent pas être retenues le temps d'une formation.

Ce qui compte n'est donc pas l'absence de dérogation, mais son encadrement. La dispense n'est pas laissée à l'appréciation de l'agent : elle exige un *motif* choisi dans une liste fermée, administrée depuis le back-office ; elle n'est accessible qu'aux profils

qui en détiennent la permission ; et son usage laisse une trace nominative dans le journal de la visite. La différence avec le registre papier n'est pas qu'un visiteur ne puisse jamais entrer sans induction — ce serait irréaliste — mais que chaque exception soit décidée par quelqu'un d'identifié, pour un motif déclaré, et retrouvable après coup.

L'intérêt pratique de cette modélisation apparaît à la lecture du code. Une seule méthode `transition()` porte la table des passages autorisés ; toute demande de changement d'état non déclarée lève une exception métier au lieu d'écrire une ligne incohérente en base. Ajouter une règle revient donc à modifier une table, non à retrouver toutes les conditions dispersées dans l'application.

Table 3.2 – Signification des états d'une visite

État	Signification
PREDECLARATION	La visite est annoncée ; l'induction n'a pas commencé
INDUCTION_EN_COURS	Le visiteur consulte le contenu de sécurité ou passe le quiz
PRET_POUR_ENTREE	L'induction est acquise ; le visiteur peut se présenter
ENTREE	Le visiteur est physiquement sur le site
SORTIE	Le visiteur a quitté le site ; son départ est enregistré
REFUSE	L'accès a été refusé — état terminal
ANNULEE	La visite a été annulée avant l'entrée — état terminal

3.2.2 Deux parcours, deux agrégats

La visite protocolaire obéit à des règles différentes : elle est déclarée par un tiers, soumise à validation hiérarchique, et son induction dispose de son propre contenu et de son propre quiz. Plutôt que d'ajouter des conditions dans le parcours ordinaire, les deux parcours ont été séparés en deux modules distincts, chacun avec ses tables et sa propre machine à états. Ils partagent la discipline de modélisation, et rien d'autre.

Le parcours VIP introduit une étape absente du parcours ordinaire : la décision d'un tiers. Le déclarant remplit un formulaire qui peut porter plusieurs visiteurs et dont le brouillon est enregistré au fil de la saisie ; la visite existe donc à l'état `BROUILLON` avant même d'être soumise. La soumission la fait passer en `EN_ATTENTE_VALIDATION` et notifie les valideurs du site concerné — par la cloche du back-office et par courriel.

Le diagramme montre ensuite une *alternative* : valider ou rejeter. Le rejet impose un motif. La validation, elle, déclenche l'envoi des liens d'induction au visiteur et informe

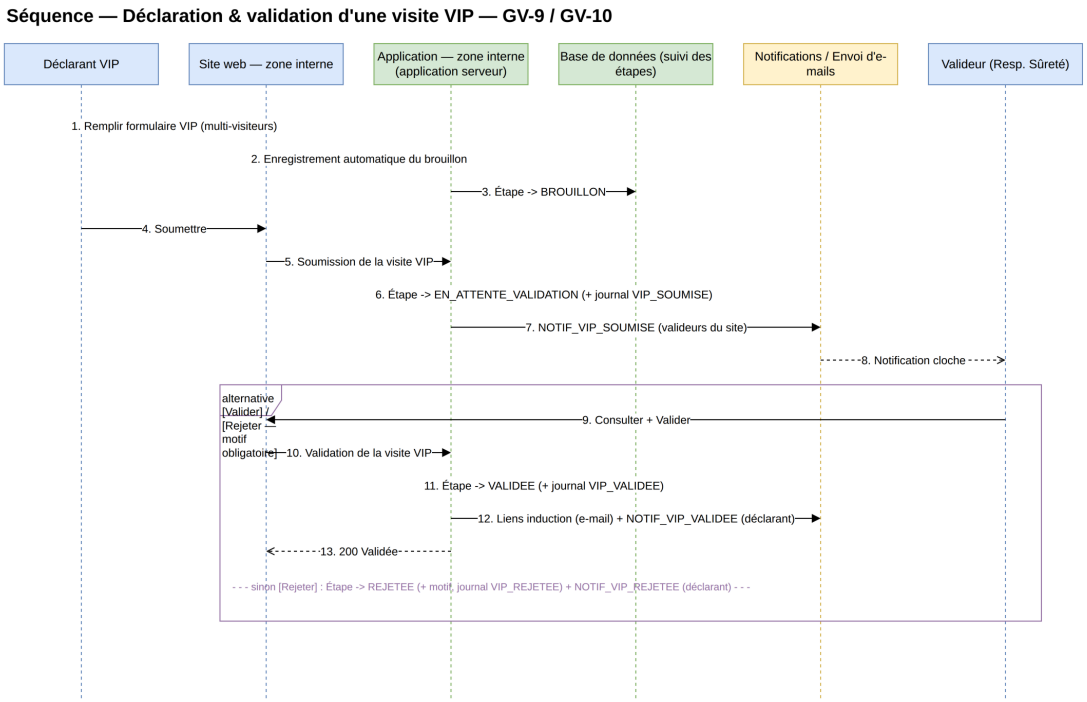


FIGURE 3.3 – Séquence — déclaration et validation d'une visite protocolaire

le déclarant. Chaque décision laisse une trace nominative : sur une visite de dirigeant, savoir qui a autorisé quoi n'est pas un luxe.

Cette dernière séquence porte la règle de sécurité la plus délicate du projet. Le visiteur ouvre un lien personnel, le serveur le vérifie et charge les contenus ; les médias sont servis par des liens temporaires valables quatre heures. Le visiteur parcourt le contenu bloc par bloc puis répond au quiz.

Le **calcul du score a lieu sur le serveur, et le diagramme précise qu'il n'est jamais exposé**. C'est la correction d'un défaut identifié en cours de projet : une première version envoyait au navigateur les questions accompagnées de la bonne réponse, ce qui rendait la triche accessible à quiconque ouvrait les outils de développement. En cas d'échec, le visiteur apprend qu'il lui reste un nombre donné de tentatives — mais pas son score, qui permettrait de converger par élimination. Ce n'est qu'à la réussite que le certificat PDF et son QR de vérification sont produits et stockés, puis transmis par un lien valable vingt-quatre heures.

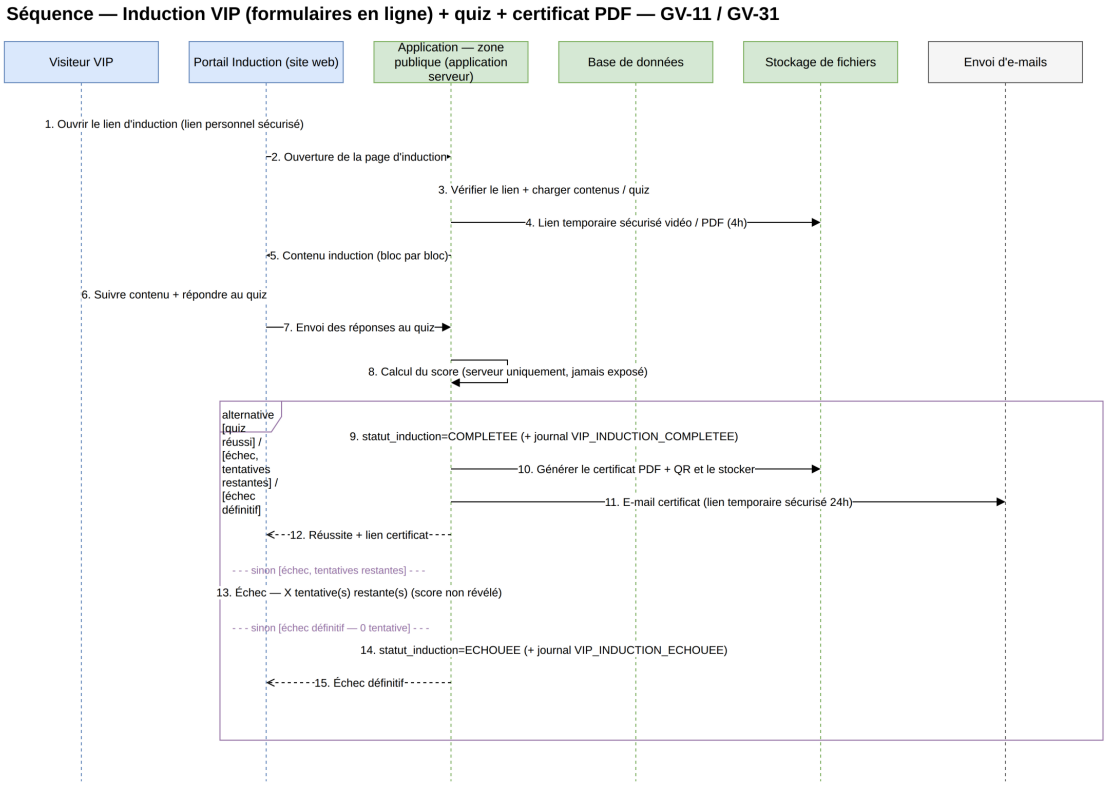


FIGURE 3.4 – Séquence — induction VIP, quiz et délivrance du certificat

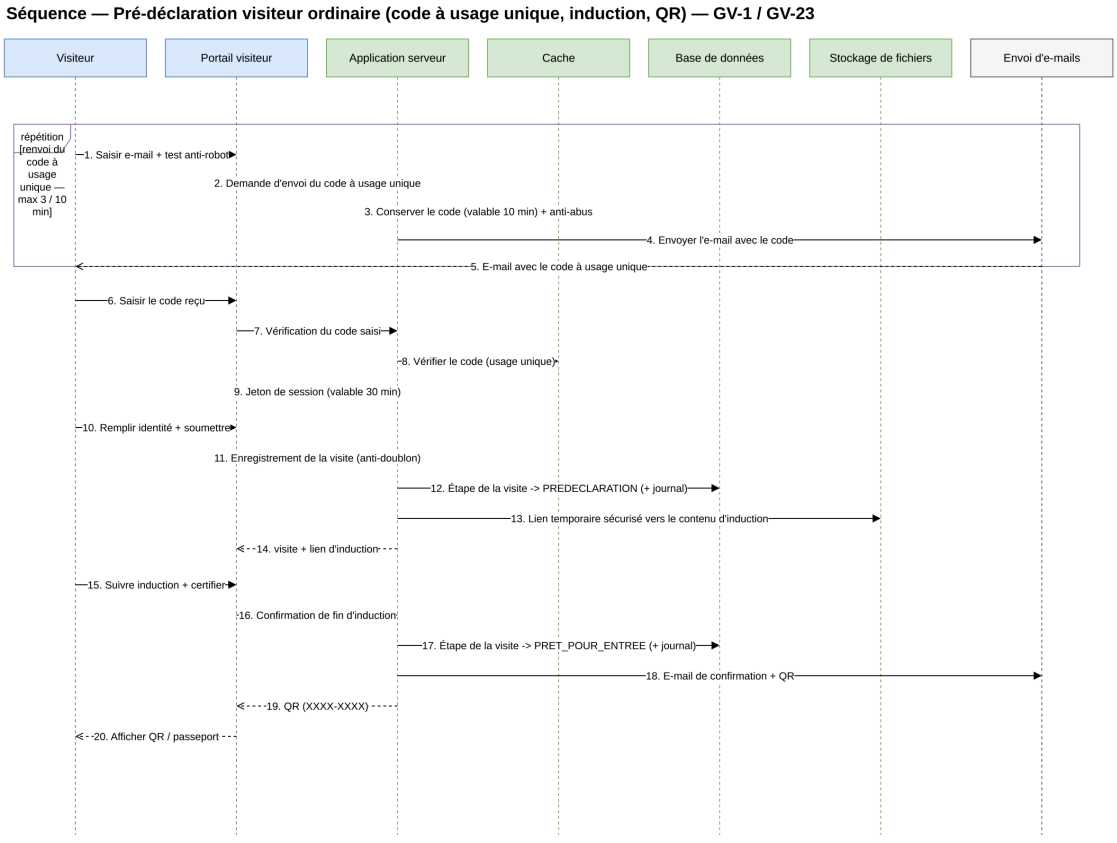


FIGURE 3.5 – Séquence — pré-déclaration d'un visiteur ordinaire

3.2.3 Le parcours ordinaire, de bout en bout

Cette séquence est celle qu'un lecteur devrait retenir s'il n'en retenait qu'une : elle montre comment une personne sans compte obtient un droit d'entrée vérifiable.

Le visiteur saisit son adresse électronique, protégée par un test anti-robots — le portail étant exposé sur Internet, sans cette protection un automate pourrait déclencher des envois en masse. Le serveur conserve un code à usage unique valable dix minutes et l'expédie par courriel ; le renvoi est limité à trois demandes par tranche de dix minutes. La vérification du code ouvre une session courte, et c'est seulement alors que le formulaire d'identité devient accessible.

L'enregistrement de la visite comporte une protection contre les doublons : sur mobile, une double soumission est fréquente, et elle ne doit pas créer deux visites. La visite naît à l'état `PREDECLARATION`, avec une entrée au journal. Le serveur renvoie alors un *lien temporaire sécurisé* vers le contenu d'induction : le document n'est pas servi par une adresse permanente qu'il suffirait de partager, mais par une adresse qui expire.

Une fois l'induction suivie et certifiée, la visite passe à `PRET_POUR_ENTREE` — le seul état depuis lequel une entrée pourra être enregistrée — et le visiteur reçoit son passeport et son code QR par courriel. Rien dans cette chaîne ne repose sur la bonne foi du navigateur : chaque transition est décidée et journalisée côté serveur.

La séquence de guérite reprend le fil là où le visiteur arrive physiquement. L'agent scanne le code QR ou recherche la visite par son nom ; le serveur la charge et vérifie son état. Le choix de la guérite, le numéro de badge éventuellement noté et la validation de l'entrée forment une seule opération, elle aussi protégée contre les doublons.

Le point remarquable est l'étape de contrôle explicite : le passage de `PRET_POUR_ENTREE` à `ENTREE` est *vérifié* avant d'être appliqué. Un visiteur dont l'induction n'est pas acquise se trouve dans un autre état, et la transition est refusée — non par une condition écrite dans l'écran de l'agent, mais par la machine à états elle-même. La sortie suit la même logique.

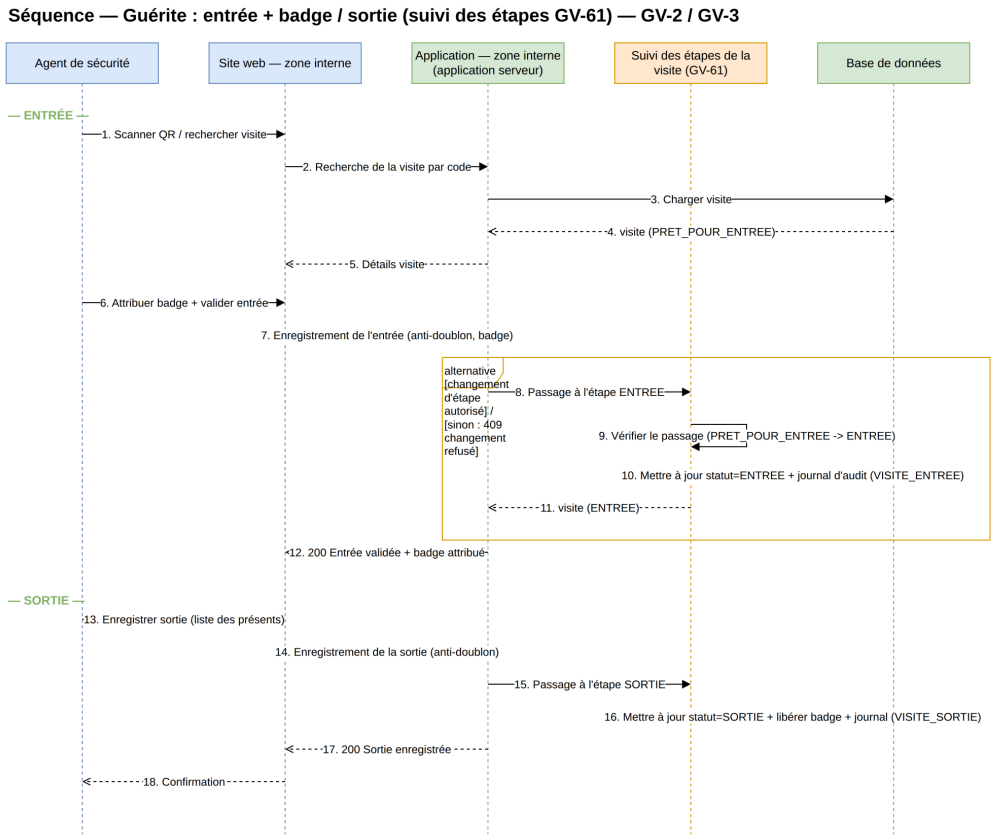


FIGURE 3.6 – Séquence — enregistrement de l'entrée et de la sortie en guérite

3.3 Modèle de données

3.3.1 Schéma de la base de données

Le schéma compte une quarantaine de tables regroupant les visites ordinaires, les visites protocolaires et leur induction, les comptes, les habilitations, les référentiels et la traçabilité. La séparation fonctionnelle des tables suit les frontières des modules applicatifs.

Le diagramme retient les classes structurantes plutôt que la totalité des tables. Il est organisé par couleur en cinq territoires, et cette organisation est elle-même une lecture de l'architecture.

Comptes et accès. `Utilisateur` et `EmployeHolcim` spécialisent une abstraction `Personne` ; un utilisateur porte son compte dans le service d'identité, son état d'activité et sa dernière connexion, et se voit affecter un `Profil`. Deux classes matérialisent les sessions : `JetonSession` pour les utilisateurs internes et `CodeOtp` pour les visiteurs, ce dernier portant explicitement sa durée de validité, son nombre de tentatives et son caractère à usage unique. Une note du diagramme énonce le point le plus important de tout le modèle : *le visiteur ordinaire n'a pas de compte*. Il est identifié par son adresse électronique, et ses

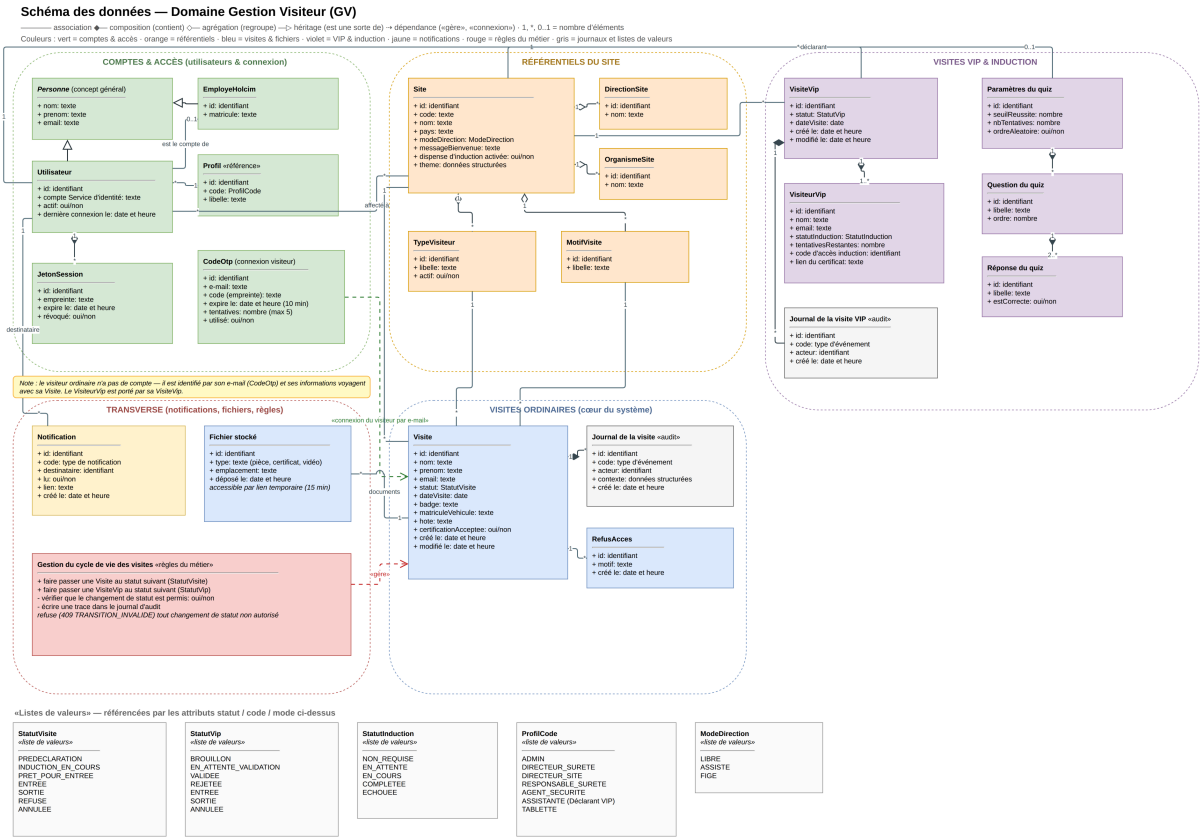


FIGURE 3.7 – Schéma des données du domaine — classes, associations et listes de valeurs

informations voyagent avec sa visite. C'est ce qui évite de créer un compte permanent pour une personne qui ne reviendra peut-être jamais.

Référentiels du site. `Site` est la classe centrale du produit. Elle ne porte pas seulement un nom et un pays, mais la *configuration* du site : message d'accueil, thème, activation d'une dispense d'induction, mode de direction. Deux sites du même groupe peuvent ainsi appliquer des règles d'accueil différentes sans duplication de code. Autour d'elle gravitent les catalogues — types de visiteur, motifs de visite — et les référentiels qui lui sont propres, organismes et directions.

Visites ordinaires. `Visite` est le cœur du système. Elle porte l'identité du visiteur, son statut, sa date, son hôte, le numéro de badge noté à titre informatif, le matricule du véhicule et l'acceptation de la certification. Deux classes l'accompagnent : `Journal de la visite`, qui conserve chaque événement avec son acteur, et `RefusActes`, qui enregistre le motif d'un refus.

Visites VIP et induction. `VisiteVip` et `VisiteurVip` forment un ensemble séparé, avec son propre journal. La configuration du quiz y figure — seuil de réussite, nombre de tentatives, ordre aléatoire — ainsi que les questions et leurs réponses. L'attribut `estCorrecte` porté par la réponse est précisément celui que la projection envoyée au navigateur ne doit jamais contenir.

Transverse. Notifications, fichiers stockés — accessibles par lien temporaire — et surtout une classe de *règles du métier*, qui explicite le mécanisme central : faire passer une visite au statut suivant, vérifier que le changement est permis, écrire une trace, et refuser toute transition non autorisée. Cette classe n'est pas une table : elle documente le fait que le cycle de vie est une règle du domaine, pas une colonne.

Listes de valeurs. Le bandeau inférieur énumère les valeurs possibles des statuts et des profils. Les faire figurer explicitement évite l'erreur classique consistant à découvrir en fin de projet qu'un statut employé dans un écran n'existe nulle part dans le modèle.

Une association mérite enfin d'être signalée, car elle a corrigé un défaut réel. Le rattachement d'un utilisateur à ses sites n'est pas une simple table de liaison : il porte une *origine*, qui distingue une affectation saisie par un administrateur d'une affectation déduite d'une désignation comme responsable ou directeur de site. Sans cette distinction, retirer une désignation supprimait aussi l'accès accordé séparément.

La table `ref_utilisateur_site` mérite un commentaire : elle matérialise le périmètre d'un utilisateur, et sa colonne `origine` distingue une affectation saisie manuellement d'une affectation résultant d'une désignation comme responsable ou directeur de site. Sans cette distinction, retirer une désignation supprimait aussi l'accès qu'un administrateur avait accordé séparément.

3.3.2 Conventions de modélisation

Plusieurs règles ont été appliquées uniformément à toutes les tables.

- **Identifiants opaques.** Les clés primaires sont des identifiants universels, non des entiers séquentiels : un identifiant séquentiel exposé dans une adresse permettrait de deviner le volume d'activité et d'énumérer les enregistrements.
- **Horodatages avec fuseau.** Tout instant est stocké avec son décalage horaire. Un horodatage sans fuseau devient ambigu dès qu'il est lu depuis un autre pays, ce qui est le cas lors des audits.
- **Position de cloisonnement obligatoire.** Chaque table déclare si elle est rattachée à un site ou globale.

- **Nommage.** Les colonnes suivent une convention française en minuscules, les identifiants de code restent en anglais.
- **Migrations versionnées.** Le schéma évolue par migrations numérotées et appliquées dans l'ordre, jamais par modification manuelle.

3.4 Conception de la sécurité

3.4.1 Chaîne d'authentification

L'authentification distingue nettement deux étapes : *prouver qui l'on est*, confié au fournisseur d'identité du groupe, et *déterminer ce que l'on peut faire*, qui reste la responsabilité de l'application.

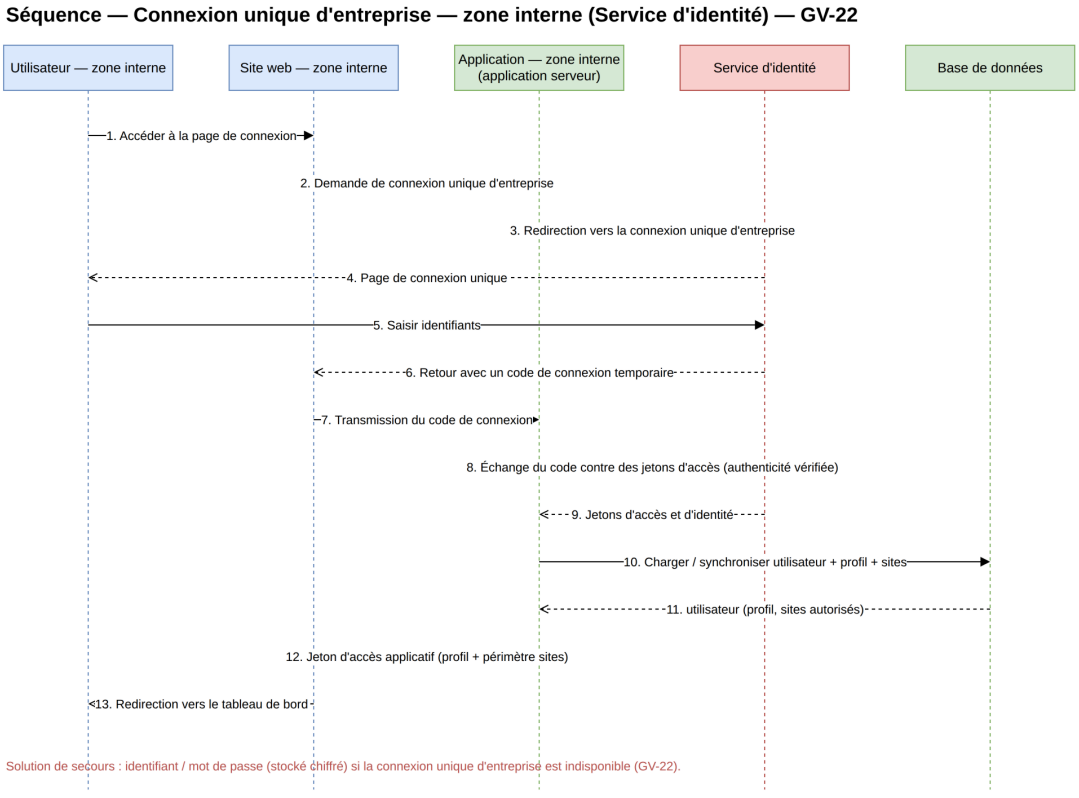


Figure 3.8 – Séquence — connexion d'un utilisateur interne par le service d'identité

La séquence détaille cette répartition. L'utilisateur demande à se connecter ; l'application le redirige vers le service d'identité de l'entreprise, où il saisit ses identifiants — que l'application ne voit jamais. Le service renvoie un code temporaire, que le serveur échange contre des jetons dont il vérifie l'authenticité. Ce n'est qu'ensuite que l'appli-

cation fait son propre travail : charger l'utilisateur, son profil et la liste des sites qui lui sont autorisés, puis émettre *son* jeton applicatif, porteur du profil et du périmètre.

Cette double émission n'est pas une complication gratuite. Le service d'identité sait qui est la personne ; il ignore tout des sites de Holcim, des profils de sûreté et des permissions du produit. Inversement l'application n'a aucune raison de connaître un mot de passe. Chacun répond à la question dont il détient la réponse.

Il aurait été plus simple d'inscrire la liste des permissions directement dans le jeton. Mais un jeton est valable un certain temps : si l'administrateur retire une permission à un profil, tous les jetons déjà émis continueraient de l'accorder jusqu'à leur expiration. En n'y inscrivant que les rôles et en reconstruisant les permissions à chaque requête — derrière un cache de courte durée — un retrait d'habilitation prend effet en quelques secondes.

3.4.2 Modèle d'habilitation

Les droits sont exprimés par des **codes de permission** suivant la forme `module.ressource.verbe`, par exemple `visite.visite.create`. Chaque profil se voit attribuer un ensemble de permissions, et chaque point d'entrée de l'API déclare la permission qu'il exige.

3.4.3 Défense en profondeur

Trois barrières indépendantes protègent les données.

1. **L'authentification** vérifie l'identité de l'appelant.
2. **L'autorisation** vérifie qu'il détient la permission requise par l'opération demandée.
3. **Le cloisonnement** restreint, au niveau de la base, les enregistrements visibles à ceux de son périmètre.

Ces barrières répondent à des questions différentes et ne se substituent pas l'une à l'autre : la deuxième dit *ce que l'on peut faire*, la troisième *sur quoi on peut le faire*.

3.4.4 Évaluation du niveau de sécurité

Le terme *sécurisé* ne doit pas être employé comme une promesse absolue. Le niveau atteint est donc évalué contrôle par contrôle, à partir du code et des tests disponibles. Cette lecture situe la solution à un niveau **robuste pour une recette d'entreprise et un PFA**, sans prétendre à une homologation de production ni à une certification de type ISO 27001.

Table 3.3 – Évaluation raisonnée des contrôles de sécurité

Dimension	Contrôle effectivement mis en œuvre	Appréciation
Identité interne	Fédération OpenID Connect avec le SSO Holcim ; le mot de passe n'est jamais géré par l'application.	Fort, sous réserve de la recette SSO cible.
Identité visiteur	OTP aléatoire à six chiffres, haché avec BCrypt, expirant, à usage unique, limité en essais et en fréquence.	Fort pour un accès temporaire.
Session web	Jeton conservé côté serveur web ; cookies `HttpOnly`, `SameSite=Lax` et `Secure` sous HTTPS.	Fort si TLS est activé.
Autorisation	Contrôles par permission sur les points d'entrée et recalcul périodique des droits après modification d'un profil.	Fort et testable.
Cloisonnement	Filtre de lecture par site, contrôle des écritures et réponse 404 hors périmètre pour éviter l'oracle d'existence.	Fort, défense en profondeur.
Surface publique	Limitation par adresse IP, jetons non prédictibles, URLs temporaires pour les fichiers et réponses d'erreur normalisées.	Bon ; à compléter par un WAF en production.
Quiz et certificats	Bonnes réponses absentes de l'API publique, correction serveur, nombre d'essais borné et QR de vérification distinct.	Fort sur le risque de fraude simple.
Traçabilité	Événements d'audit horodatés et corrélés, sans journalisation du code OTP en clair.	Bon ; conservation à valider avec l'exploitation.

Les réponses de l'API ajoutent en outre une politique CSP stricte, `X-Content-Type-Options: nosniff`, l'interdiction d'affichage en cadre et HSTS lorsque la requête arrive en HTTPS.

Les secrets de production ne sont pas intégrés aux images : ils doivent être injectés par le gestionnaire de secrets de l'environnement.

3.4.5 Protection des données personnelles

La plateforme traite, par nature, des données à caractère personnel : identité du visiteur, adresse électronique, pièce d'identité, photographie et horaires de présence sur un site. Cette dimension n'est pas un ajout réglementaire postérieur à la conception ; elle a orienté plusieurs décisions.

Le texte applicable est d'abord la **loi marocaine 09-08** [5] relative à la protection des personnes physiques à l'égard du traitement des données à caractère personnel, dont la **CNDP** — Commission nationale de contrôle de la protection des données à caractère personnel — est l'autorité de contrôle. Holcim appartenant par ailleurs à un groupe international, les principes du **RGPD** européen ont été pris comme référence de conception [7]. Il faut souligner que cela *ne vaut pas conformité automatique* à la loi 09-08 : les deux textes diffèrent notamment par leurs formalités préalables, et se conformer à l'un ne dispense pas d'instruire l'autre.

Formalité applicable : à faire qualifier. La nature exacte de la formalité — déclaration ou demande d'autorisation préalable — dépend des données effectivement collectées et de leurs finalités. Le projet prévoit la saisie d'un numéro de pièce d'identité lorsque le site l'exige ; selon les formalités publiées par la CNDP [6], ce type de donnée peut appeler une autorisation plutôt qu'une simple déclaration. La qualification relève du responsable de traitement et de la fonction juridique de l'entreprise, non du projet : elle est identifiée ici comme un préalable à la mise en production, et non tranchée.

Une réserve sur le consentement. Le rapport décrit plus loin un consentement recueilli avant la soumission de la déclaration. Cette mécanique est correcte techniquement, mais sa qualification juridique demande vérification : un consentement dont dépend l'accès au site n'est pas nécessairement « libre » au sens des textes. Une autre base légale — l'intérêt légitime de sécurisation des sites industriels, ou une obligation légale de sécurité — est probablement plus solide. Ce point est également renvoyé à la fonction juridique.

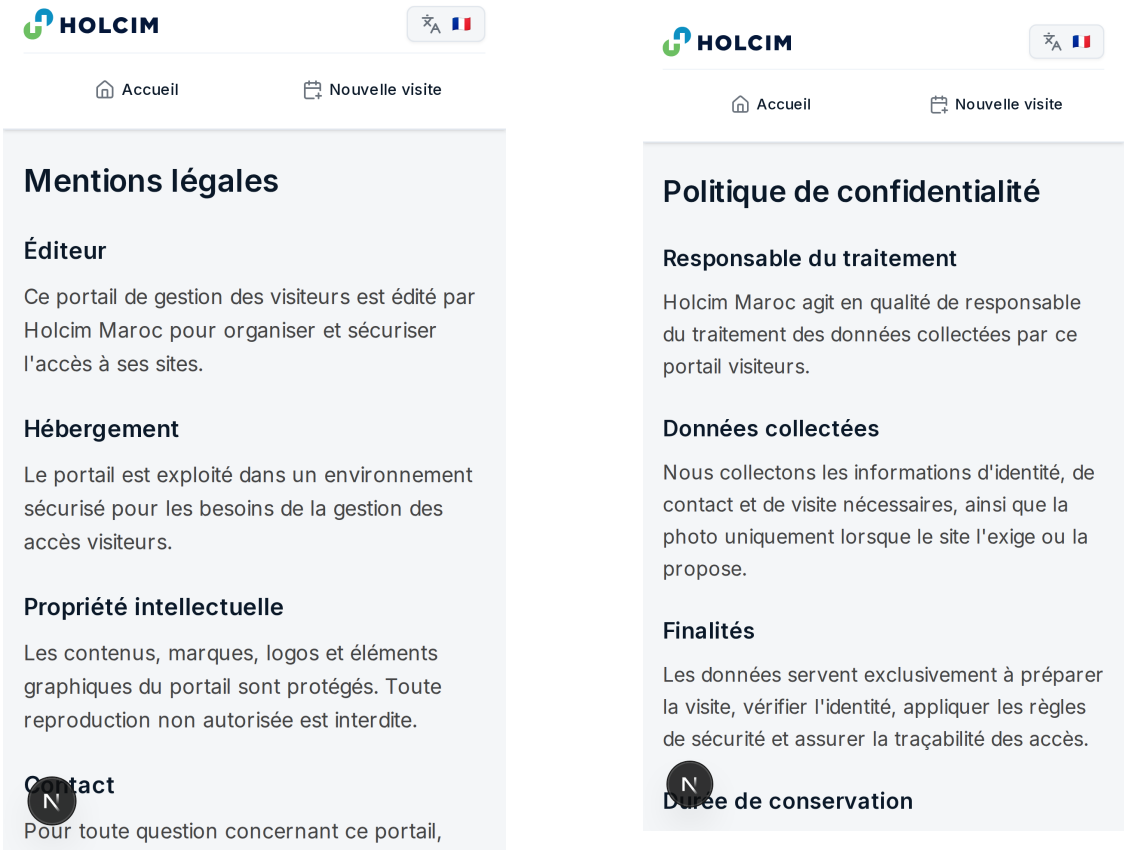
Les paragraphes qui suivent décrivent ce qui a effectivement été mis en œuvre sur le plan technique.

Minimisation. Seules les données nécessaires au contrôle d'accès sont demandées. Le visiteur ne crée pas de compte : il n'a donc ni mot de passe à confier, ni profil persistant à maintenir. La photographie n'est requise que lorsque le site l'exige dans sa configuration, et non par défaut.

Consentement explicite. La déclaration ne peut être soumise sans acceptation de la politique de confidentialité. Le consentement n'est pas un simple état d'interface : il est enregistré avec la visite, horodaté, ce qui permet de démontrer sur quelle version du texte le visiteur s'est prononcé.

Durée de conservation. Les données de visite sont conservées le temps requis par les obligations de sécurité et d'audit, puis purgées. Les photographies, qui constituent la donnée la plus sensible du dispositif, font l'objet d'une purge automatique distincte et plus courte que celle du dossier de visite lui-même.

Information des personnes. Le portail visiteur publie ses mentions légales et sa politique de confidentialité, accessibles sans authentification depuis toutes les pages du parcours. Le visiteur peut donc consulter le traitement dont il fait l'objet *avant* de transmettre la moindre donnée.



Mentions légales Politique de confidentialité

Figure 3.9 – Information du visiteur sur le traitement de ses données

Cloisonnement comme mesure de protection. Le périmètre par site décrit plus haut n'est pas seulement une règle métier : il limite le nombre de personnes ayant accès à un dossier donné, ce qui est une mesure de protection des données au sens propre.

Exercice des droits. Le droit d'accès et le droit à l'effacement s'exercent par demande auprès du responsable de traitement, qui dispose dans le back-office des écrans de consultation et de suppression nécessaires pour y répondre. La purge technique des dossiers échus est implémentée et pilotée par configuration ; elle est livrée inactive par défaut, car fixer une durée de conservation est une décision du responsable de traitement et non un choix technique.

Ce qui relève de l'exploitant. Une homologation de production suppose de confirmer le TLS de bout en bout, la rotation des secrets, la sauvegarde et la restauration, les alertes de supervision, l'analyse des dépendances, le test d'intrusion et les règles du proxy ou du WAF. Ces contrôles relèvent de l'infrastructure d'accueil autant que du code ; ils ne sont donc pas déclarés acquis sur la seule base des vérifications applicatives.

Conclusion

La conception étant posée, le chapitre suivant décrit sa mise en œuvre.

CHAPITRE 4

Réalisation

Introduction

> *Ce chapitre présente les technologies retenues, l'environnement de travail, l'organisation du code, et les interfaces obtenues.*

4.1 Douze notions pour lire ce chapitre

Ce chapitre emploie un vocabulaire technique. Les douze notions ci-dessous suffisent à le suivre entièrement ; elles sont définies une fois ici et réemployées ensuite sans être redéfinies.

Table 4.1 – Définitions des notions techniques employées

Notion	Définition
API	Point d'entrée par lequel un programme en appelle un autre. Ici, l'ensemble des adresses que les deux interfaces web sollicitent pour lire ou modifier des données.
REST	Style d'API où chaque adresse désigne une ressource — une visite, un site — et où le verbe HTTP exprime l'action : lire, créer, modifier, supprimer.
JSON	Format texte d'échange de données entre le serveur et le navigateur, lisible par un humain.
OpenAPI	Fichier décrivant l'API dans un format normalisé : adresses, paramètres, réponses, erreurs. Il sert à engendrer automatiquement du code des deux côtés.
Jeton (JWT) [9]	Petit document signé par le serveur, remis après authentification, que le porteur présente à chaque appel. La signature empêche d'en modifier le contenu.
SSO	Authentification unique : l'utilisateur prouve son identité auprès d'un service central de l'entreprise, et non auprès de chaque application.
ORM	Couche traduisant automatiquement les objets du programme en lignes de base de données, et inversement.
Migration	Fichier décrivant une évolution du schéma de la base — ajouter une table, une colonne. Numérotées et appliquées dans l'ordre, elles rendent l'évolution reproductible.
Transaction	Groupe d'opérations en base qui réussissent ou échouent ensemble. Un échec en cours annule les précédentes.
Conteneur	Paquet contenant une application et tout ce dont elle a besoin pour s'exécuter. Il se comporte à l'identique sur un poste de développement et sur un serveur.
Kubernetes [21]	Système qui exécute des conteneurs sur un ensemble de machines : il les démarre, les surveille, les redémarre et les remplace.
Intégration continue	Chaîne déclenchée à chaque modification déposée : elle compile, teste et fabrique les livrables automatiquement.

4.2 Choix technologiques

Table 4.2 – Technologies retenues et justification

Couche	Technologie	Justification
Langage serveur	Java 21	Typage fort, robustesse et écosystème mature
Cadre serveur	Spring Boot 4	Configuration, sécurité, persistance et observabilité intégrées
Modularité	Spring Modulith 2	Vérifie automatiquement les frontières entre modules
Persistance	JPA / Hibernate	Correspondance objet-relationnel ; filtres de cloisonnement
Base de données	PostgreSQL 16	Transactions, types avancés et verrous consultatifs
Migrations	Flyway 11	Versionnement du schéma et application ordonnée
Langage client	TypeScript 5	Typage statique et réduction des erreurs d'intégration
Cadre client	Next.js 16 / React 19	Rendu hybride et routage par système de fichiers
Style	Tailwind CSS 4	Cohérence visuelle et composants réutilisables
État serveur	TanStack Query	Cache, invalidation et rechargement gérés
Contrat d'API	OpenAPI / Orval	Une source de vérité et des clients TypeScript générés
Identité	OpenID Connect / JWT	Raccordement au SSO Holcim, OTP visiteur et autorisation applicative
Stockage fichiers	MinIO / S3	Sépare les fichiers volumineux de la base
Sécurité applicative	Spring Security	Filtres HTTP, contrôle des permissions par point d'entrée
Cache mémoire	Caffeine	Évite de recalculer les permissions à chaque requête
Documents PDF	OpenPDF	Génère le certificat d'induction côté serveur (ADR-0018)
Codes QR	ZXing	Produit les QR du passeport et du certificat
Fichiers Excel	fastexcel	Import et export des utilisateurs sans charger le fichier en mémoire (ADR-0011)

suite page suivante

Couche	Technologie	Justification
Composants d'interface	Metronic	Base graphique des deux portails, adaptée à la charte Holcim
Tests	JUnit, Vitest, Testcontainers, Playwright	Vérification du serveur, des portails et des parcours
Conteneurisation	Docker multi-stage	Images reproductibles et limitées au nécessaire
Orchestration	Kubernetes / Helm	Déploiement paramétrable sur l'infrastructure cible

4.2.1 Pourquoi Spring Boot côté serveur

Le choix du socle serveur n'était pas libre : l'application devait s'exécuter sur l'infrastructure interne du groupe, s'intégrer au service d'identité d'entreprise et être maintenue par une équipe dont la plateforme habituelle est Java. Cela posait le cadre. À l'intérieur de ce cadre, quatre raisons ont fait retenir **Spring Boot**.

Un cadre applicatif, pas une bibliothèque. Spring Boot fournit d'emblée ce qu'une application de gestion doit de toute façon posséder : serveur HTTP, injection de dépendances, gestion des transactions, sécurité, validation des données entrantes, points de santé et journalisation structurée. Ces briques auraient sinon dû être assemblées et maintenues une par une — travail sans valeur métier, et premier endroit où se logent les défauts de sécurité.

Le contrôle transactionnel. Le domaine impose des opérations qui doivent réussir ou échouer *en bloc* : enregistrer une entrée, faire transiter la visite vers son nouvel état et écrire la trace d'audit forment une seule unité. La déclaration de la frontière transactionnelle par annotation, au niveau du service applicatif, rend cette exigence lisible dans le code plutôt qu'implicite.

Spring Modulith, la raison décisive. C'est l'extension qui a fait pencher la balance. Elle permet de déclarer des *modules* à l'intérieur d'une application unique et **vérifie leurs frontières à la compilation** [16]. Sans elle, le découpage modulaire n'aurait été qu'une convention de nommage — donc érodée dès la première urgence.

Un écosystème durable. Java et Spring sont des choix de long terme : la personne qui reprendra ce projet dans trois ans trouvera des compétences disponibles et des versions maintenues. Sur une application destinée à être exploitée par une entreprise industrielle, cette considération pèse autant qu'un argument technique.

4.2.2 Ce qu'est un module, concrètement

Le mot *module* étant employé partout dans ce rapport, il vaut la peine d'en donner la définition exacte retenue ici.

Un module est un **répertoire de code** portant une responsabilité métier unique, dont l'intérieur est *inaccessible* depuis les autres modules. Chacun s'organise en trois espaces :

- `api/` — ce que le module offre aux autres : interfaces et types de données. C'est son contrat public.
- `events/` — les faits qu'il publie, par exemple *une visite a été validée*. Un autre module peut y réagir sans que le premier sache qu'il existe.
- `internal/` — tout le reste : entités de persistance, services, contrôleurs. **Rien de ce qui s'y trouve ne peut être appelé de l'extérieur.**

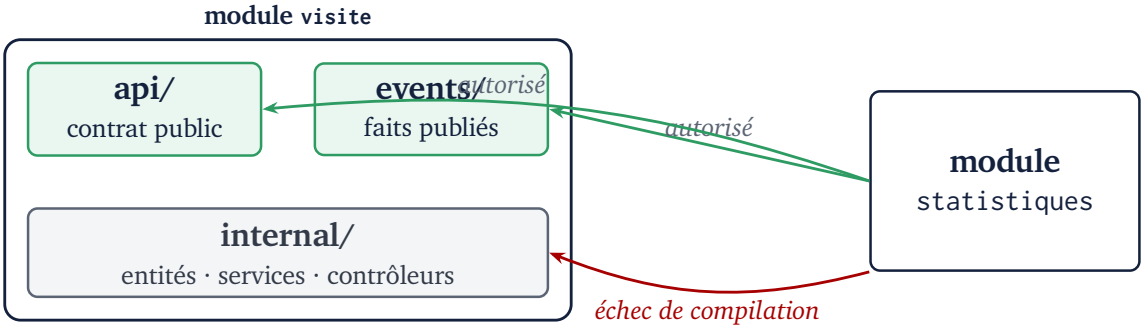


FIGURE 4.1 – Frontière d'un module : seuls `api/` et `events/` sont atteignables de l'extérieur

La vérification est mécanique. Si un développeur — ou un agent — écrit dans le module *statistiques* un appel vers une classe interne du module *visite*, comme le montre la flèche rouge de la figure 4.1, la construction échoue avec un message nommant la règle enfreinte. Ce n'est pas une recommandation : c'est une erreur de compilation, au même titre qu'une faute de syntaxe.

L'intérêt est de conserver les bénéfices du découpage — un module se comprend, se teste et se remplace isolément — *sans* payer le prix d'un système distribué : une seule application à déployer, une seule base, aucun appel réseau entre modules, aucune cohérence répartie à gérer. C'est le compromis que désigne l'expression *monolithe modulaire*.

4.2.3 Pourquoi Next.js côté navigateur

Deux interfaces très différentes devaient être servies : un portail public, ouvert sur Internet, consulté depuis un téléphone en quelques minutes ; et un back-office dense, utilisé toute la journée sur un poste de travail. **Next.js** a été retenu pour trois raisons.

Un serveur devant le navigateur. Next.js n'est pas seulement une bibliothèque d'interface : il exécute aussi du code côté serveur. C'est ce qui permet de conserver le jeton d'authentification dans un cookie inaccessible au JavaScript de la page et de faire transiter tous les appels par ce serveur intermédiaire. Le navigateur ne connaît jamais l'adresse de l'API interne, et un script malveillant injecté dans la page ne peut pas lire le jeton. Une application purement navigateur aurait dû stocker ce jeton dans la page, où il est lisible.

Une seule technologie pour deux portails. Les deux interfaces partagent les mêmes composants, les mêmes formulaires, la même gestion des erreurs. En développant seul, cette mutualisation était la condition pour tenir le périmètre.

Un client d'API engendré. Le contrat OpenAPI produit automatiquement le code TypeScript d'appel à chaque point d'entrée. Une incohérence entre le serveur et l'interface devient une erreur de compilation au lieu d'une panne visible par l'utilisateur.

4.2.4 Lecture technique de la stack

La table précédente donne les noms des technologies. Pour comprendre leur collaboration, il est plus utile de suivre une requête depuis l'écran jusqu'aux données. Les neuf couches ci-dessous sont numérotées dans l'ordre où une requête les traverse.

Table 4.3 – Rôle technique de chaque couche

	Couche	Fonctionnement technique
1	Interfaces web	Next.js exécute les deux portails React. TypeScript vérifie les types à la compilation, Tailwind CSS porte la charte visuelle et TanStack Query synchronise les écrans avec l'état du serveur.
2	Passerelle web	Les appels `/api` passent par le serveur Next.js. Le navigateur conserve une session sécurisée dans un cookie `HttpOnly` et ne contacte pas directement l'API interne.
3	Contrat d'API	OpenAPI décrit les routes, paramètres, réponses et erreurs. Orval engendre les clients TypeScript ; le backend implémente les interfaces Java produites depuis le même contrat.
4	Backend métier	Spring Boot reçoit la requête, authentifie l'appelant, vérifie sa permission et délègue l'opération au module métier concerné. Spring Modulith empêche les modules de contourner leurs interfaces publiques.
5	Données	JPA/Hibernate traduit les objets en opérations SQL et porte le filtre de périmètre par site. PostgreSQL garantit les transactions ; Flyway applique les évolutions versionnées du schéma.
6	Identité	Le personnel utilise le SSO Holcim via OpenID Connect. Le visiteur reçoit un OTP par SMTP. Dans les deux cas, l'application émet ensuite son propre jeton JWT et calcule les permissions effectives.
7	Fichiers et messages	Le stockage compatible S3 conserve les médias d'induction et les certificats. Le relais SMTP transporte les codes, confirmations, invitations et rapports planifiés.
8	Qualité	JUnit et Vitest couvrent les règles isolées, Testcontainers vérifie l'intégration avec une vraie base et Playwright reproduit les actions d'un utilisateur dans Chromium.
9	Livraison	Docker fabrique trois images reproductibles. GitLab CI les teste et les publie ; Helm transforme la configuration en ressources Kubernetes déployables par environnement.

4.2.5 Chemin d'une requête, étape par étape

Prenons l'exemple d'un agent qui valide l'entrée d'un visiteur.

1. L'agent déclenche l'action depuis le back-office React. Le formulaire vérifie d'abord les champs simples avant tout appel réseau.

2. La passerelle Next.js lit la session protégée, ajoute le jeton nécessaire et transmet la requête à l'API Spring Boot.

3. Le contrôleur valide le contrat OpenAPI, puis le service métier vérifie la permission, le site de l'agent et l'état courant de la visite.

4. Une transaction PostgreSQL enregistre l'entrée, applique la transition d'état et ajoute l'événement d'audit. Si une étape échoue, la transaction entière est annulée : aucune donnée partielle n'est conservée.

5. La réponse revient au portail. TanStack Query invalide les données devenues anciennes puis recharge la liste des présents, sans recharger toute la page.

Cette circulation explique la séparation des responsabilités : le frontend guide l'utilisateur, la passerelle protège la session, le backend impose les règles métier et la base garantit la cohérence. Une validation affichée dans le navigateur n'est donc jamais considérée comme une autorisation suffisante ; le contrôle décisif reste côté serveur.

4.2.6 Le contrat d'API comme source de vérité

Un choix mérite d'être souligné, car il a évité une classe entière d'erreurs. Le contrat de chaque point d'entrée est décrit dans un fichier de spécification, à partir duquel sont engendrées d'une part l'interface que le contrôleur serveur doit implémenter, d'autre part le client TypeScript utilisé par les portails.

Le compilateur refuse alors qu'un contrôleur s'écarte du contrat, et le client ne peut pas appeler un point d'entrée inexistant. Sans ce mécanisme, une divergence entre le serveur et le client ne se manifesterait qu'à l'exécution, sous forme d'une erreur incompréhensible pour l'utilisateur.

Le contrat n'est pas seulement un fichier de génération : il est publié sous forme navigable, ce qui permet d'essayer un point d'entrée et d'en lire les paramètres et les codes de retour sans ouvrir le code.

4.3 Environnement de développement

4.3.1 Stack exécutée en local

Le poste local utilise un JDK 21 pour le backend, Node.js 22 avec pnpm 9 pour les deux portails et Docker Compose pour les dépendances d'infrastructure. Le backend Spring Boot écoute sur le port 8080, le portail visiteur Next.js sur 3100 et le back-office Next.js sur 3101. Les deux interfaces utilisent la même API, avec une population d'authentification distincte.

Docker Compose reproduit les dépendances techniques sans installer PostgreSQL, MinIO ou le serveur de messagerie directement sur la machine. Des volumes nommés conservent les données entre deux démarrages, tandis que des *healthchecks* empêchent les composants dépendants de démarrer avant que leur service ne soit prêt.

Table 4.4 – Services de l'environnement local

Service	Port	Rôle
API	8080	Application serveur
Portail visiteur	3100	Interface publique
Portail back-office	3101	Interface d'administration
PostgreSQL 16	5433	Base relationnelle locale (port conteneur 5432)
Redis	6379	Service auxiliaire disponible ; non requis par le cœur applicatif actuel
Adminer	8090	Inspection de la base PostgreSQL en développement
RedisInsight	5540	Inspection du service Redis
MinIO API	9000	Stockage S3 des inductions, certificats et ressources
MinIO Console	9001	Administration du stockage objet
Mailpit SMTP	1025	Réception des courriels applicatifs en local
Mailpit Web	8026	Consultation des courriels interceptés

4.3.2 Les services d'infrastructure, un par un

Le tableau précédent énumère les services ; il ne dit pas pourquoi chacun est là. Or ce sont ces choix qui déterminent ce que l'application sait faire — et, en recette, ce que l'exploitation doit lui fournir.

PostgreSQL — la base de données. Elle porte l'intégralité de l'état métier. Trois de ses capacités sont réellement exploitées, et non seulement disponibles : les *transactions*, qui permettent qu'une entrée, la transition d'état de la visite et sa trace d'audit réussissent ou échouent ensemble ; les *contraintes d'unicité*, sur lesquelles repose la protection contre la double soumission ; et les *verrous consultatifs*, qui garantissent qu'un traitement planifié ne s'exécute qu'une fois même lorsque plusieurs exemplaires de l'application tournent en parallèle.

MinIO — le stockage de fichiers. Les documents d'induction, les certificats et les photographies ne sont pas conservés dans la base. Une base relationnelle stocke mal les fichiers volumineux : elle alourdit les sauvegardes, sature la mémoire de cache et ralentit les requêtes qui n'ont rien à voir. MinIO offre une interface compatible avec le standard S3, ce qui permet de remplacer le service local par le stockage objet de l'entreprise en changeant seulement des paramètres. Trois espaces distincts sont créés : un pour les contenus d'induction, un pour les certificats, un pour les ressources propres à chaque site.

Le point important est la manière dont ces fichiers sont servis. Le serveur ne les relaie pas et ne publie pas non plus d'adresse permanente : il produit à la demande un *lien temporaire signé*, valable quelques heures, qui expire ensuite. Un lien de contenu d'induction transmis à un tiers cesse donc de fonctionner de lui-même.

Keycloak — le fournisseur d'identité local. En recette et en production, l'authentification des salariés passe par le service d'identité du groupe. En développement, il faut un équivalent : Keycloak joue ce rôle avec des comptes de test isolés. C'est ce qui permet de travailler et d'exécuter les tests sans dépendre d'un service d'entreprise ni manipuler de véritables identités.

Mailpit — le serveur de messagerie de test. Le parcours visiteur repose sur des courriels — code de connexion, passeport, certificat. Les envoyer réellement pendant le développement serait à la fois inutile et risqué. Mailpit les intercepte tous et les présente dans une interface web : on vérifie le contenu exact du message sans qu'aucun courriel ne quitte la machine. C'est également ce qui a permis d'automatiser le test de bout en bout de la connexion par code, en lisant le courriel reçu au lieu de simuler le code.

Redis — présent, non requis. Le service est déclaré dans l'environnement et disponible, mais le cœur applicatif ne l'utilise pas aujourd'hui : le cache des permissions vit dans la mémoire de l'application. Redis a été conservé pour la suite, lorsqu'un cache partagé entre plusieurs exemplaires deviendra nécessaire. Il est signalé ici parce qu'un lecteur du dépôt le verrait et se demanderait à quoi il sert.

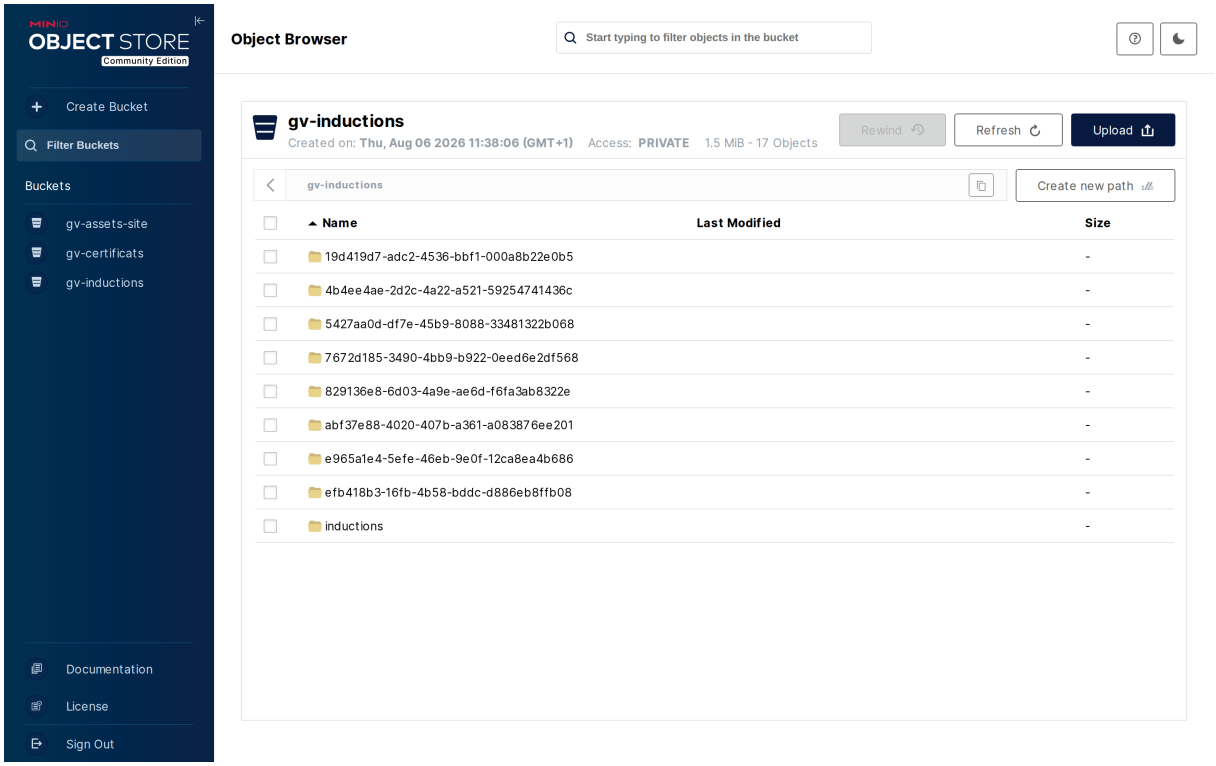


Figure 4.2 – Espaces de stockage et objets conservés dans MinIO

La console de MinIO donne à voir ce que le texte décrit. Les trois espaces figurent dans la colonne de gauche ; celui des contenus d'induction est ouvert et contient dix-sept objets pour 1,5 Mio. Deux observations s'imposent. L'accès est marqué **privé** : aucun objet n'est lisible sans passer par une adresse signée produite par le serveur. Et les objets portent des *identifiants opaques*, non des noms de fichiers — un contenu d'induction ne s'appelle pas `consignes-casablanca.pdf`, ce qui empêche de deviner une adresse ou de déduire quoi que ce soit du nom d'un objet.

La figure 4.3 montre l'intérêt du dispositif : trois demandes de connexion réelles, et le contenu exact du message tel que le visiteur le recevrait — expéditeur, objet, code à six chiffres, durée de validité et mention d'usage unique. Aucun de ces courriels n'a quitté la machine. C'est également par cette interface que le test automatisé de bout en bout lit le code : il ne le simule pas, il le récupère dans le message effectivement émis.

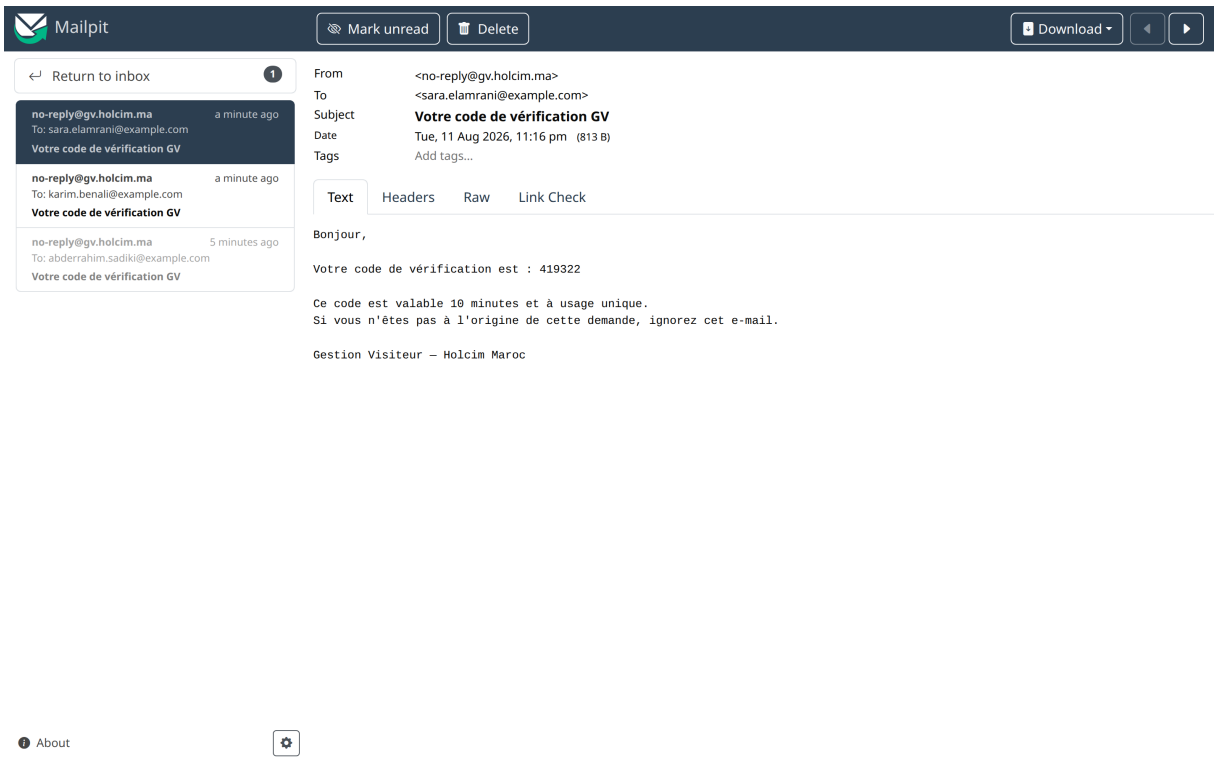


Figure 4.3 – Courriels émis par la plateforme, interceptés en développement

Adminer et RedisInsight — l'inspection. Ces deux interfaces ne participent pas au fonctionnement de l'application : elles permettent d'observer directement le contenu de la base et du cache pendant le développement. Elles ont servi à vérifier, après chaque parcours, que l'état réellement écrit correspondait à ce que l'interface annonçait — la seule façon de détecter un écran qui affiche un succès sans que rien n'ait été enregistré.

4.3.3 Démarrage et cycle de travail

Le démarrage se fait en trois temps, dans cet ordre : les services d'infrastructure, puis le serveur applicatif, puis les deux portails. L'ordre importe — le serveur applique les migrations de schéma au démarrage et échouerait si la base n'était pas encore prête.

```
docker compose up -d
docker compose ps
```

Listing 4.1 – Démarrage des services d'infrastructure

Cette première commande lance la base PostgreSQL, le stockage MinIO, le fournisseur d'identité, le serveur de messagerie de test et les interfaces d'inspection. Les *healthchecks* déclarés dans la composition empêchent un service dépendant de démarrer avant que le service dont il dépend ne réponde.

```
./gradlew :app:bootRun
```

Listing 4.2 – Démarrage du serveur applicatif

Gradle compile les douze modules, applique les migrations Flyway en attente, puis démarre l'application sur le port 8080. L'ajout de l'option `--continuous` active la recompilation à chaque enregistrement de fichier.

```
pnpm -C portals/visitors dev
pnpm -C portals/bo dev
```

Listing 4.3 – Démarrage des deux portails

Chaque portail démarre son propre serveur Next.js — 3100 pour le portail visiteur, 3101 pour le back-office — avec rechargement à chaud. L'arrêt se fait par interruption de chaque processus, puis `docker compose down` pour les services d'infrastructure.

```
./gradlew cleanTest :app:test test    # tests serveur + architecture
./gradlew integrationTest             # tests sur base conteneurisee
pnpm -C portals/bo typecheck && pnpm -C portals/bo test
pnpm -C portals/visitors typecheck && pnpm -C portals/visitors test
```

Listing 4.4 – Vérifications avant validation

Ces quatre commandes constituent la vérification exécutée avant chaque validation dans le dépôt. Le `cleanTest` est délibéré : sans lui, Gradle réutilise le résultat d'une suite inchangée et affiche un succès instantané qui ne prouve rien.

Dans l'environnement cible, le back-office s'appuie sur le SSO Holcim au moyen du protocole OpenID Connect. Les essais locaux utilisent des identités isolées de développement ; ce mécanisme n'est ni une fonction métier ni un composant présenté lors de la démonstration. Mailpit remplace le relais SMTP réel et permet de vérifier les codes OTP, confirmations et rapports sans envoyer de courriels vers l'extérieur. MinIO remplace un stockage S3 et initialise les buckets `gv-inductions`, `gv-certificats` et `gv-assets-site`.

4.4 Quatre mécanismes en détail

Les chapitres précédents ont énoncé quatre garanties : un cycle de vie contraint, un cloisonnement qui ne dépend pas de la vigilance du développeur, des droits recalculés à chaque requête, et une résistance à la double soumission. Cette section montre comment chacune est effectivement obtenue dans le code, car une garantie que l'on ne peut pas exhiber n'en est pas une.

4.4.1 La machine à états : une table, et une seule porte

Le cycle de vie n'est pas une suite de conditions dispersées dans les services : c'est une table de correspondance, déclarée une fois.

```
private static final Map<StatutVisite, Set<StatutVisite>> TRANSITIONS =
    Map.of(
        StatutVisite.PREDECLARATION,
            Set.of(StatutVisite.INDUCTION_EN_COURS,
                   StatutVisite.PRET_POUR_ENTREE,
                   StatutVisite.ANNULEE),
        StatutVisite.INDUCTION_EN_COURS,
            Set.of(StatutVisite.PRET_POUR_ENTREE),
        StatutVisite.PRET_POUR_ENTREE,
            Set.of(StatutVisite.ENTREE, StatutVisite.REFUSE,
                   StatutVisite.ANNULEE),
        ...);
```

Listing 4.5 – Table des transitions autorisées (extrait)

Le contrôle lui-même tient en trois lignes : si le couple *(état courant, état visé)* n'appartient pas à la table, une exception métier est levée et l'appel se termine par un code HTTP 409.

```
S require(S from, S to) {
  if (!isAllowed(from, to)) {
    throw new TransitionInvalideException(from, to);
  }
  return to;
}
```

Listing 4.6 – Le garde de transition

Trois propriétés en découlent. La table est **lisible** : un lecteur y voit l'intégralité du cycle sans parcourir le code. Elle est **exhaustive** : les états terminaux n'ont aucune transition sortante, donc rien ne peut réactiver une visite close. Et elle est **unique** : la modification directe de la colonne de statut hors de cette machine est proscrite, ce qui garantit qu'aucun chemin d'écriture ne contourne le contrôle. Les deux parcours — ordinaire et protocolaire — possèdent chacun leur table, mais partagent ce même garde générique : la discipline est commune, les règles ne le sont pas.

4.4.2 Le cloisonnement : un filtre au niveau de la session

Le périmètre par site n'est pas appliqué requête par requête. Il est déclaré une fois sur la classe de base dont héritent toutes les entités rattachées à un site, sous la forme d'un filtre Hibernate.

```
@MappedSuperclass
@FilterDef(
    name = SITE_FILTER,
    parameters = @ParamDef(name = SITE_FILTER_PARAM, type = UUID.class),
    applyToLoadByKey = true)
@Filter(
    name = SITE_FILTER,
    condition = "site_id IN (:" + SITE_FILTER_PARAM + ")")
public abstract class BaseTenantScopedEntity extends BaseEntity { ... }
```

Listing 4.7 – Déclaration du filtre de périmètre

Trois détails de cette déclaration méritent d'être relevés, car chacun correspond à une faille évitée.

La condition est une **appartenance à un ensemble** et non une égalité : un responsable sûreté peut couvrir plusieurs sites, et le modèle devait le permettre sans dupliquer les requêtes.

L'option `applyToLoadByKey` est la plus importante. Sans elle, le filtre ne s'appliquerait qu'aux requêtes écrites — et un simple chargement par identifiant, la forme la plus courante d'accès, rendrait la ligne d'un autre site. C'est précisément le trou que le filtre est censé fermer.

Enfin, le filtre est activé ou laissé inactif *par session*, selon le périmètre de l'appelant lu au début de la requête. Un administrateur sans restriction voit tout ; un acteur dont le périmètre n'a pas pu être déterminé ne voit *rien*, car l'absence de périmètre est traitée comme une erreur et non comme une permission. Ce choix — échouer en se fermant plutôt qu'en s'ouvrant — est la différence entre une sécurité qui se dégrade proprement et une sécurité qui s'évapore au premier défaut de configuration.

Un contrôle d'architecture complète le dispositif : toute nouvelle entité doit déclarer explicitement si elle est rattachée à un site ou globale, faute de quoi la compilation échoue. Une table introduite sans y penser ne peut donc pas être silencieusement lisible depuis tous les sites.

Les limites de ce mécanisme, et ce qui les couvre. Il faut être précis sur la nature de cette protection : c'est un filtre *applicatif*, porté par la couche de correspondance objet, et non une politique de sécurité au niveau des lignes de la base — la fonctionnalité que PostgreSQL nomme *Row Level Security*. La distinction n'est pas théorique : un filtre applicatif ne s'applique qu'aux requêtes qui passent par lui.

Trois chemins y échappent par construction, et chacun appelle une réponse.

- **Les requêtes SQL natives.** Le filtre réécrit les requêtes exprimées dans le langage de la couche objet, pas le SQL écrit à la main. Le projet impose donc qu'une requête native portant sur une table rattachée à un site déclare explicitement une dérogation motivée, et un contrôle d'architecture fait échouer la compilation en son absence. Le module de statistiques est aujourd'hui le principal utilisateur de cette dérogation : son cloisonnement est réimplémenté à la main et doit être revérifié à chaque évolution.
- **Les traitements planifiés et les imports.** Ils s'exécutent sans utilisateur, donc sans périmètre issu d'une requête. Le mécanisme est *fermé par défaut* : en l'absence de périmètre établi, toute lecture d'une table rattachée à un site échoue au lieu de retourner l'ensemble des lignes. Un traitement global doit donc déclarer son intention, ce qui le rend visible.
- **Un accès direct à la base.** Aucun mécanisme applicatif ne protège d'une connexion administrateur à PostgreSQL. C'est la limite que *Row Level Security* [15] couvrirait, et c'est la piste recommandée pour les tables les plus sensibles avant une exploitation à grande échelle.

Cette défense est donc solide pour les chemins applicatifs et explicite sur ce qu'elle ne couvre pas. Les tests d'intégration comportent des scénarios négatifs — un acteur du site A ne doit ni lire, ni modifier, ni exporter une donnée du site B — mais leur généralisation à toutes les surfaces reste un travail à poursuivre.

4.4.3 Les droits, recalculés plutôt que transportés

Le jeton applicatif ne transporte que des rôles. À chaque requête, le vérificateur reconstitue l'ensemble des permissions effectives à partir du catalogue, derrière un cache de très courte durée.

Le compromis est explicite. Inscrire les permissions dans le jeton supprimerait ce calcul, mais figerait les droits pour toute la durée de vie du jeton : un retrait d'habilitation resterait sans effet jusqu'à expiration. Les recalculer à chaque appel coûterait une

lecture en base par requête. Le cache de quelques dizaines de secondes place le curseur entre les deux : la vérification reste sans accès à la base dans le cas courant, et une modification d'habilitation prend effet en quelques secondes au lieu de plusieurs minutes.

4.4.4 L'idempotence, ou pourquoi deux clics ne font qu'une visite

Sur un téléphone, une soumission de formulaire est fréquemment envoyée deux fois — connexion lente, double appui, retour arrière. Sans protection, le visiteur obtiendrait deux visites et deux passeports.

Le service de pré-déclaration s'appuie sur un garde d'idempotence : la requête porte une clé, et une tentative répétée avec la même clé renvoie le résultat de la première au lieu d'en créer une seconde. La garantie est portée par une contrainte d'unicité en base plutôt que par une vérification préalable en mémoire — deux requêtes simultanées passeraient toutes deux un simple test d'existence, alors qu'une seule peut franchir une contrainte unique. La différence n'apparaît que sous concurrence, c'est-à-dire exactement dans le cas que l'on cherche à couvrir.

4.5 Organisation du code

```
gv/
|-- app/                    # point d'entree de l'application serveur
|-- app-batch/              # squelette reserve aux traitements planifies
|-- app-seed/               # squelette reserve a l'initialisation
|-- modules/
|   |-- domain/             # modules metier
|   |   |-- identity/
|   |   |-- site/
|   |   |-- visite/
|   |   |-- visite-vip/
|   |   |-- referentiels/
|   |   |-- notification/
|   |   `-- statistiques/
|   `-- platform/           # modules techniques
|       |-- persistence/
|       |-- security/
|       |-- audit/
|       |-- blobstore/
|       `-- config/
|-- portals/
|   |-- visitors/           # portail visiteur
|   `-- bo/                 # portail back-office
|-- deploy/helm/            # description du deploiement
```

```
|-- docker/                # services locaux
`-- docs/                  # specification et decisions d'architecture
```

Listing 4.8 – Arborescence du dépôt

Deux entrées de cette arborescence appellent une précision. `app-batch/` et `app-seed/` sont des squelettes : ils portent leur classe de démarrage et leur configuration, mais aucun traitement. Ils ont été créés pour fixer l'emplacement des traitements planifiés et de l'initialisation des données le jour où ils seront nécessaires ; les tâches périodiques existantes s'exécutent aujourd'hui dans l'application principale. Le répertoire `modules/integration/`, prévu par l'architecture pour les adaptateurs vers les systèmes externes, reste vide : il naîtra avec la première synchronisation d'annuaire.

4.5.1 Organisation d'une fonctionnalité côté client

Chaque écran suit une organisation constante : un dossier `_services/` contenant la logique — appels à l'API, schémas de validation, colonnes de tableau — et un dossier `_view/` contenant les composants d'affichage. Cette séparation rend la localisation d'un fichier prévisible et évite qu'une règle métier ne se retrouve enfouie dans un composant graphique.

4.6 Réalisation des principales fonctionnalités

4.6.1 Comment le système fonctionne, dans son ensemble

Avant d'entrer dans les écrans, il est utile de poser le fonctionnement d'ensemble en une page. Le système repose sur quatre mécanismes qui se combinent, et qui expliquent presque tous les comportements observables.

Un état, et des transitions nommées. Toute visite occupe exactement un état. Aucun écran ne modifie directement une donnée de visite : il demande une *transition*. Le serveur vérifie que le passage demandé fait partie des passages autorisés depuis l'état courant, l'applique, et écrit une ligne de journal. Un agent ne peut donc pas enregistrer l'entrée d'un visiteur dont l'induction n'est pas acquise — non parce que le bouton est grisé, mais parce que la transition n'existe pas. Le bouton grisé n'est que la politesse de l'interface.

Une identité, deux portées. Le service d'identité de l'entreprise répond à la question *qui êtes-vous*. L'application répond à la question *que pouvez-vous faire ici*, en émettant son propre jeton porteur des rôles et du périmètre de sites. Les visiteurs, eux, n'ont pas d'identité d'entreprise : un code à usage unique envoyé par courriel leur ouvre une session courte. Deux populations, deux mécanismes, jamais mélangés.

Un périmètre appliqué en profondeur. Le périmètre de sites n'est pas un filtre ajouté à chaque écran : il est appliqué dans la couche d'accès aux données, sur toute lecture d'une table rattachée à un site. Une requête écrite sans y penser reste cloisonnée. Et lorsqu'un identifiant hors périmètre est demandé, la réponse est *introuvable* — jamais *interdit* —, pour ne pas révéler l'existence de ce que l'on n'a pas le droit de voir.

Une trace pour chaque décision. Toute action qui change l'état du système produit un événement d'audit horodaté et attribué. Le mécanisme mérite d'être décrit avec précision, car « synchrone » et « asynchrone » ne suffisent pas à le qualifier : il s'agit d'un **outbox**. L'événement est *publié dans la transaction métier* — il est donc enregistré de façon durable en même temps que l'action, et disparaît avec elle si celle-ci est annulée. Son *écriture* dans le journal, en revanche, a lieu après la validation de la transaction, dans un traitement séparé.

La garantie obtenue est celle-ci : il ne peut exister ni trace d'une opération annulée, ni opération validée sans trace en attente. Un arrêt du serveur entre la validation et l'écriture ne perd rien — la publication étant persistée, elle est reprise au redémarrage. En contrepartie, la trace peut apparaître dans le journal avec un léger différé, ce qui est acceptable pour un journal d'audit et évite qu'une lenteur d'écriture ne ralentisse l'utilisateur.

Ces quatre mécanismes suffisent à expliquer le parcours complet. Une visite naît d'une déclaration, franchit l'induction, devient présentable, est contrôlée en guérite, entre, puis sort — chaque étape étant une transition vérifiée, exécutée par un acteur identifié, dans un périmètre contrôlé, et consignée.

4.6.2 Lecture fonctionnelle de bout en bout

Pour un lecteur qui découvre la plateforme, le fonctionnement global se résume en cinq moments. D'abord, le visiteur s'identifie par son adresse électronique et un code temporaire. Ensuite, il prépare sa visite à distance ou scanne le code du site lorsqu'il arrive sans déclaration. Il complète les informations demandées et, si le site l'exige, suit l'induction de sécurité. L'application produit alors un passeport avec un code d'accès

et un QR. Enfin, l'agent de guérite contrôle ce passeport, enregistre l'entrée puis la sortie. En parallèle, le back-office permet aux équipes Holcim de valider, paramétrer, superviser et auditer ces opérations. Les sections suivantes reprennent ce fil dans le même ordre.

4.6.3 Authentification et gestion des accès

Les utilisateurs internes accèdent au back-office au moyen du SSO d'entreprise Holcim. Après l'authentification, le backend échange l'identité externe contre un jeton applicatif et reconstruit les permissions à partir des profils configurés. Les visiteurs ordinaires utilisent un code à usage unique reçu par courriel, ce qui évite la création d'un mot de passe permanent. La gestion des utilisateurs comprend la création, la modification, la désactivation, l'affectation à plusieurs sites, l'import Excel, la recherche et l'export.

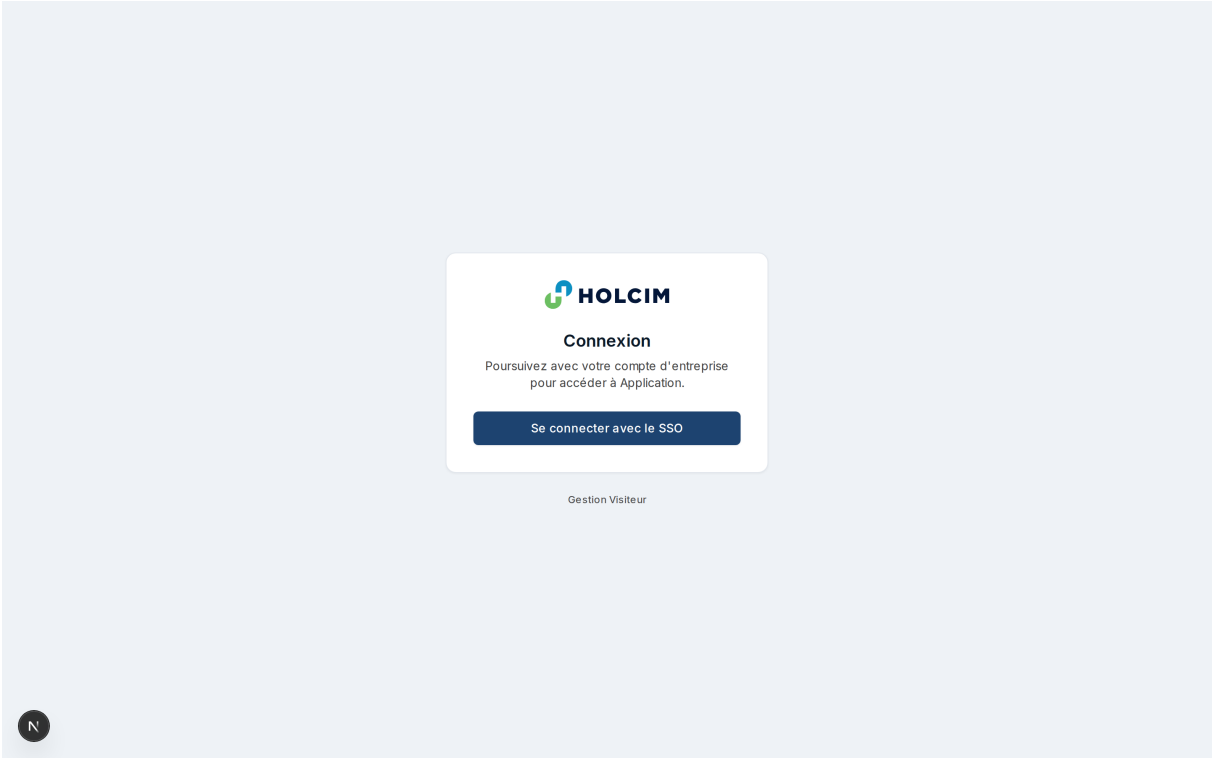


Figure 4.4 – Page de connexion du back-office

La page de connexion du back-office ne comporte ni champ de mot de passe ni formulaire d'inscription : un seul bouton délègue l'authentification au fournisseur d'identité. C'est une conséquence directe du choix d'architecture — l'application ne détient aucun mot de passe d'employé, donc elle n'en demande aucun et n'en stocke aucun.

Sur la voie de secours prévue au backlog. Les récits GV-22 et GV-25 mentionnent une authentification de repli par identifiant et mot de passe. Il faut lever l'ambiguïté : *cette voie n'est pas une authentification locale de GV*. L'application ne comporte aucun magasin de mots de passe, et le modèle de données n'en prévoit aucun. Ce que le backlog désigne est la possibilité, pour le fournisseur d'identité lui-même, d'authentifier un compte par mot de passe lorsque la fédération avec l'annuaire d'entreprise n'est pas disponible — la gestion du secret restant entièrement de son côté. Le changement de mot de passe évoqué par GV-25 s'opère donc **sur le fournisseur d'identité**, seul détenteur du secret : c'est une conséquence directe de la séparation entre l'authentification, qui lui revient, et l'autorisation, que GV assume.

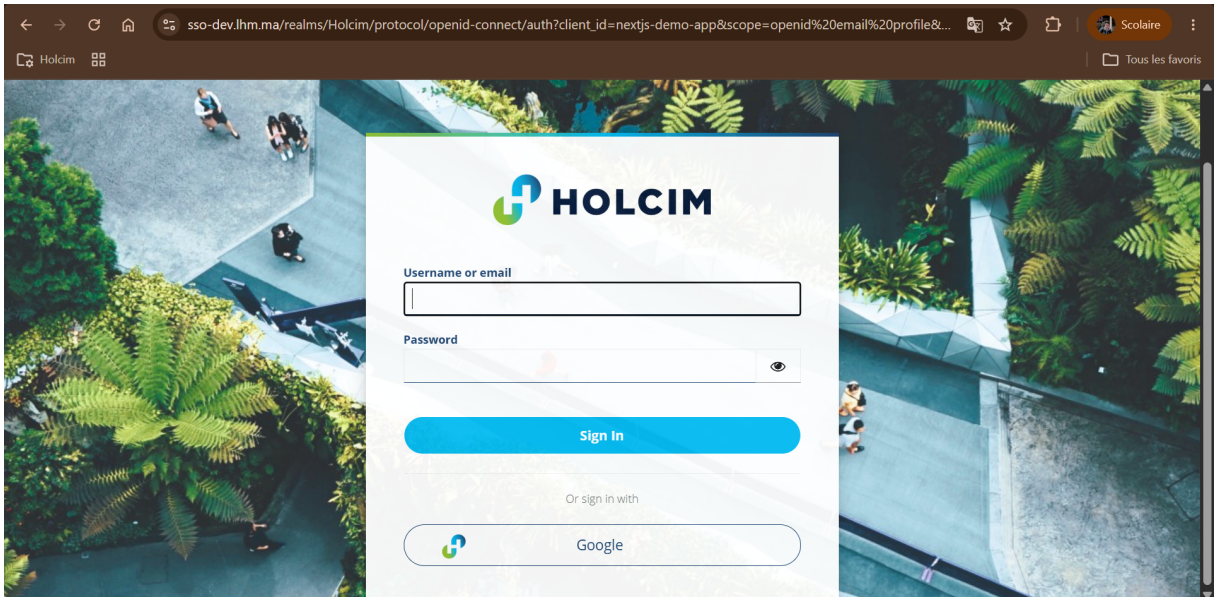


Figure 4.5 – Page d'authentification du fournisseur d'identité Holcim

La figure 4.5 montre l'écran atteint après le clic sur le bouton unique de la page précédente. Deux détails méritent l'attention. L'adresse, d'abord : le navigateur a quitté l'application et se trouve sur le domaine du service d'identité de l'entreprise — c'est là, et nulle part ailleurs, que les identifiants sont saisis. Les paramètres visibles dans cette adresse sont ceux du protocole OpenID Connect : l'application appelante, les informations demandées, et l'adresse de retour. Ensuite, l'écran lui-même est celui de Holcim, avec sa charte : l'utilisateur reconnaît l'environnement dans lequel il s'authentifie habituellement, ce qui est en soi une protection contre les pages d'hameçonnage.

Une fois les identifiants validés, le service renvoie le navigateur vers l'application avec un code temporaire, et la séquence décrite au chapitre précédent se déroule.

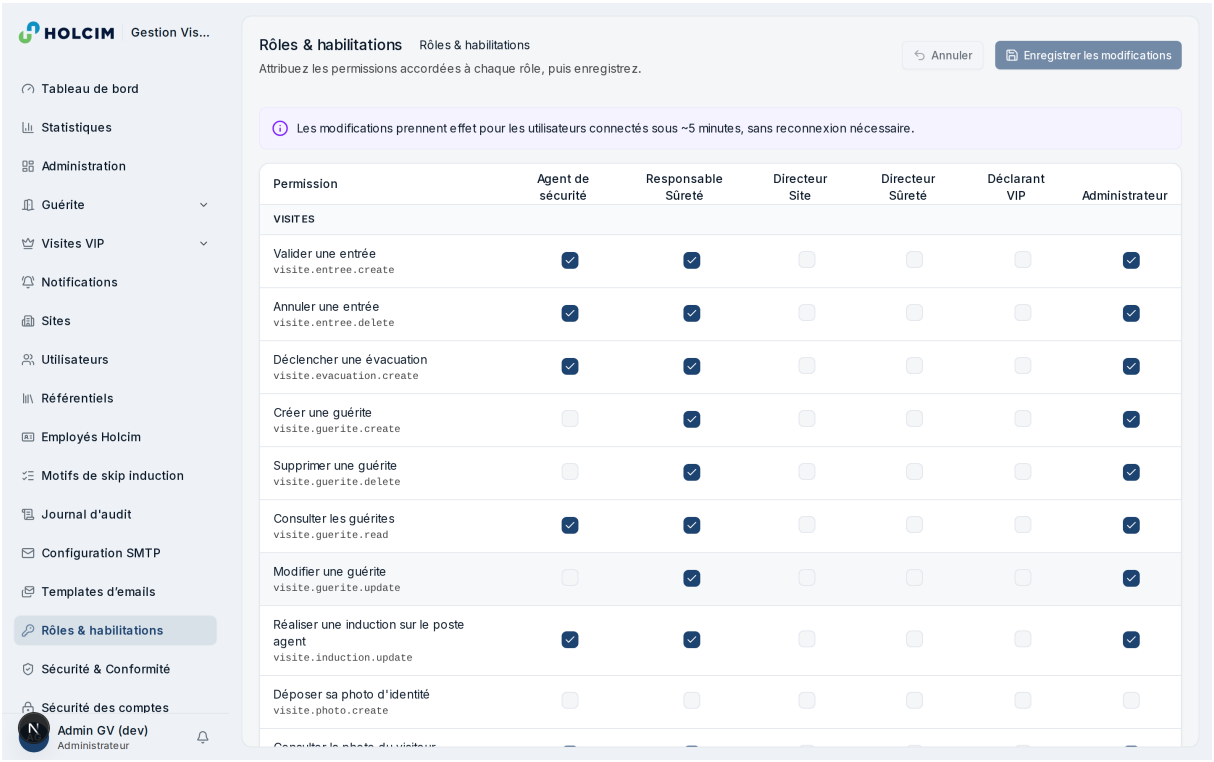


Figure 4.6 – Écran de gestion des rôles et habilitations

4.6.4 Parcours du visiteur

Le portail visiteur est conçu *mobile d'abord* : la majorité des visiteurs y accèdent depuis leur téléphone, souvent debout devant la guérite. Les captures ci-dessous sont donc prises au format d'un téléphone. La figure 4.7 montre d'abord la même page sur un écran d'ordinateur, pour situer l'ensemble.

Trois éléments de cette page méritent d'être relevés. Les **deux portes d'entrée** sont présentées côte à côte : préparer sa visite à l'avance, ou saisir le code du site quand on est déjà devant la guérite — ce sont exactement les deux cas d'usage identifiés à l'analyse. Le **sélecteur de langue** en haut à droite applique l'exigence bilingue français/anglais propre à ce portail, le back-office restant en français seul. Enfin, la **bannière de consentement** n'est pas décorative : le portail n'utilise que les cookies nécessaires à son fonctionnement, et le choix de l'utilisateur est mémorisé.

Le visiteur ne crée pas de compte : il saisit son adresse électronique, reçoit un code à six chiffres et le saisit. Ce choix supprime la gestion d'un mot de passe pour un utilisateur qui ne reviendra peut-être jamais. Le code est haché avant d'être conservé, expire au bout de dix minutes, ne sert qu'une fois, et le nombre de tentatives comme la fréquence des demandes sont bornés — sans quoi une adresse exposée sur Internet deviendrait un moyen d'envoi en masse.

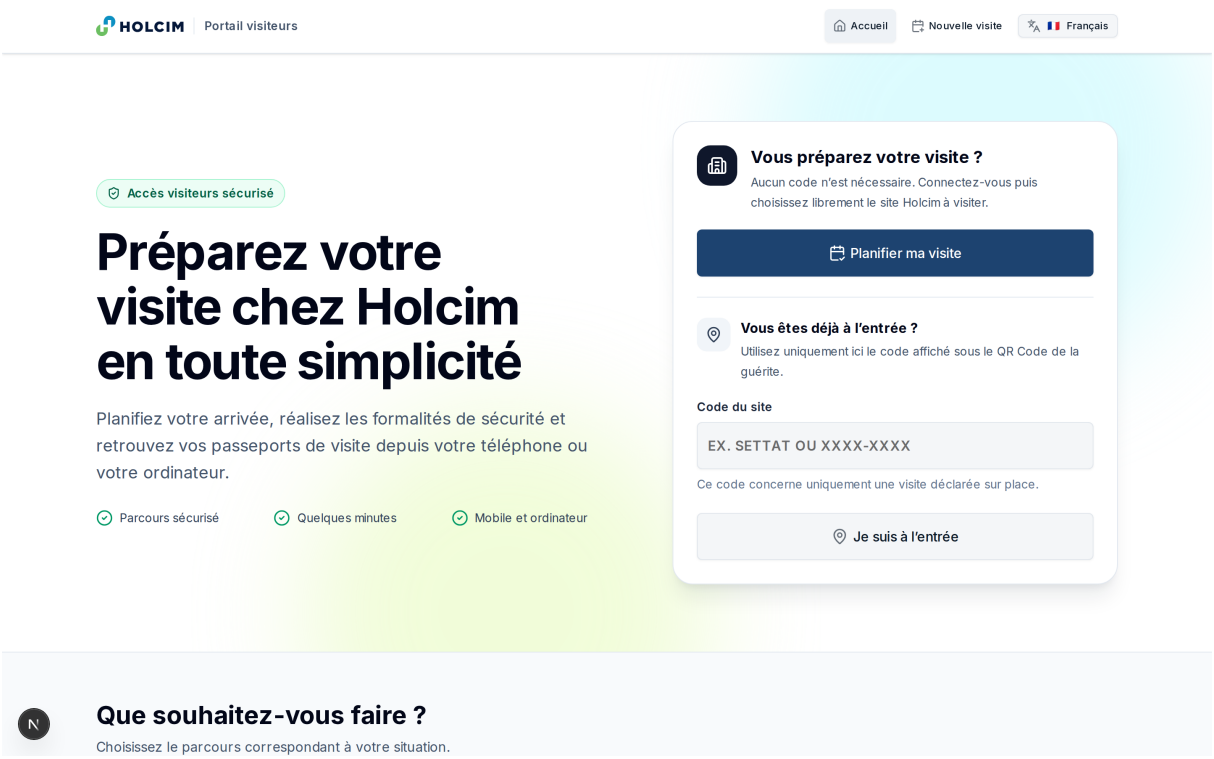
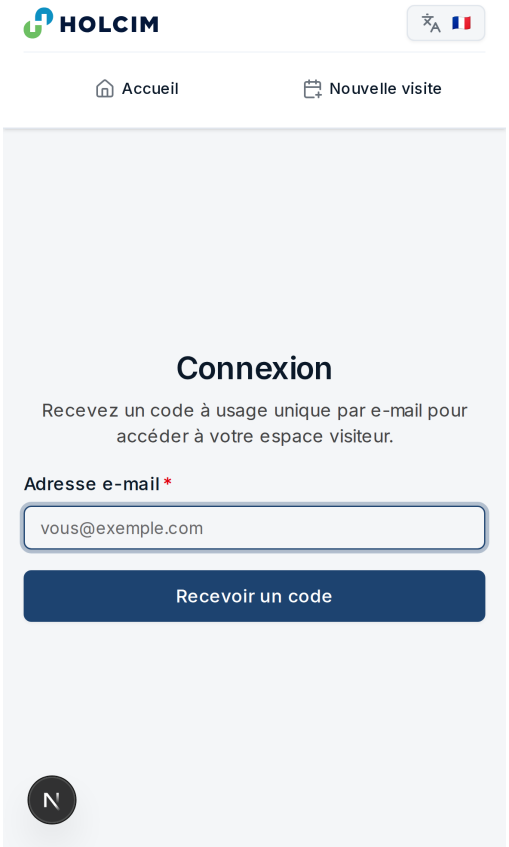
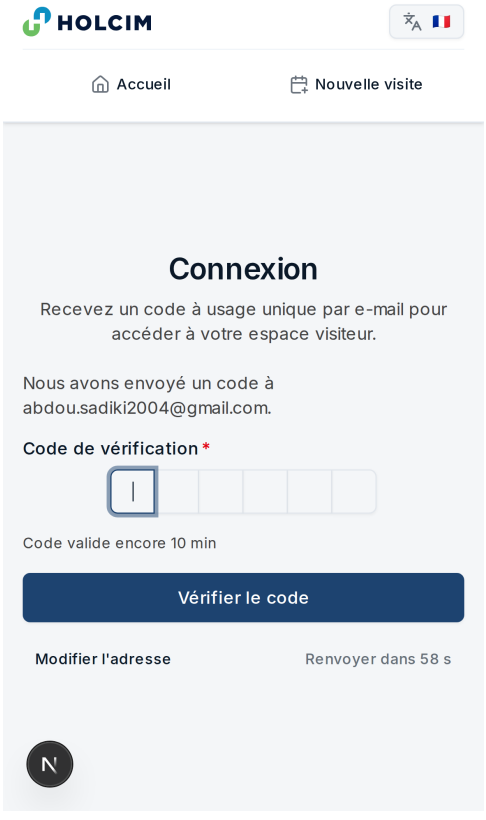


Figure 4.7 – Page d'accueil du portail visiteur sur écran d'ordinateur



Demande du code



Saisie du code reçu

Figure 4.8 – Identification du visiteur par code à usage unique

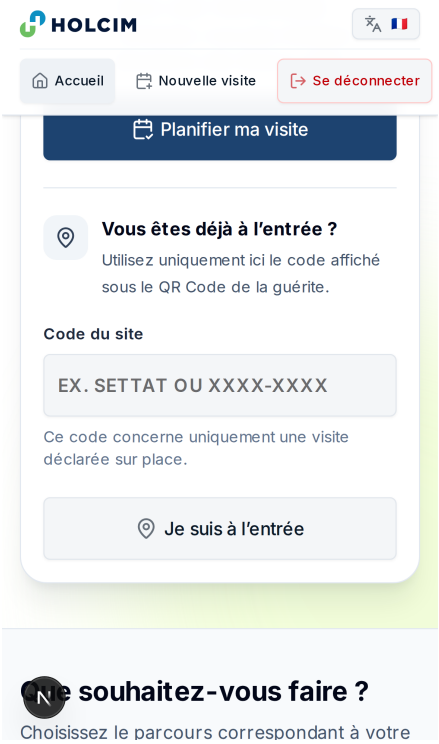


Figure 4.9 – Portail visiteur — entrée dans le parcours — Espace visiteur connecté

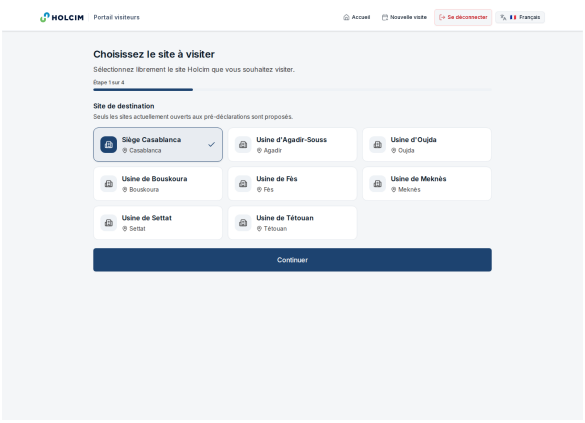
Le formulaire est découpé en trois étapes plutôt que présenté d'un bloc : sur un écran de téléphone, un formulaire long décourage, et le taux d'abandon augmente. Les champs obligatoires portent un astérisque dérivé automatiquement du schéma de validation, ce qui évite qu'un champ change de statut sans que le marqueur suive.

Preuve d'un parcours de pré-déclaration exécuté

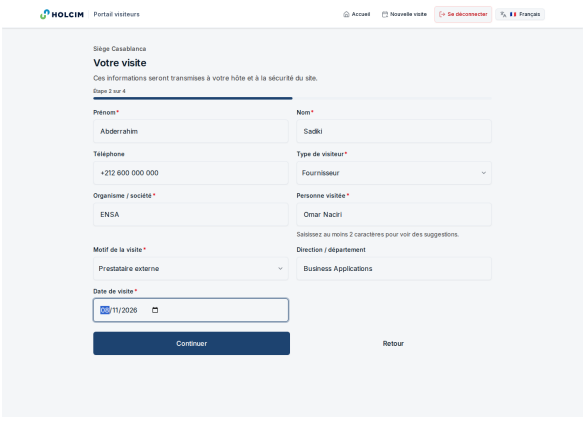
Les écrans suivants ne sont pas des maquettes. Une session visiteur a été ouverte par OTP, le site de Casablanca a été sélectionné et le formulaire a été soumis avec des données de démonstration le 30 août 2026. La capture large rend lisibles les informations métier ; le même composant adopte une colonne unique sur téléphone.

Après validation du formulaire, le backend fournit le document d'induction du site au moyen d'une URL temporaire. Pour un PDF, le bouton de certification reste verrouillé pendant la durée minimale configurée ; pour une vidéo, le lecteur bloque l'avance au-delà de la portion réellement visionnée. Le temps de consultation et l'acceptation sont transmis avec la déclaration.

Une fois la déclaration terminée, le visiteur reçoit une confirmation et retrouve son passeport dans l'espace *Mes visites*. Le QR n'est pas une simple illustration : il transporte le code que l'agent utilise pour retrouver rapidement la visite et appliquer les contrôles d'entrée.

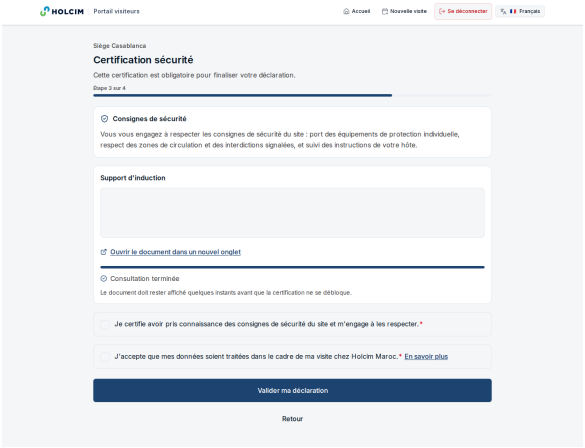


Sélection du site administré

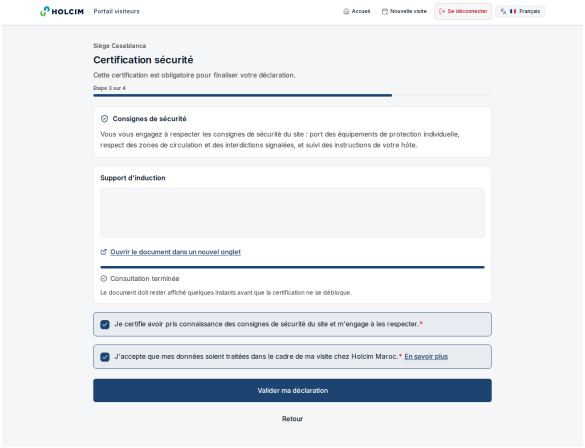


Identité et objet de la visite

Figure 4.10 – Pré-déclaration réelle — sélection et formulaire



Contenu et progression verrouillée

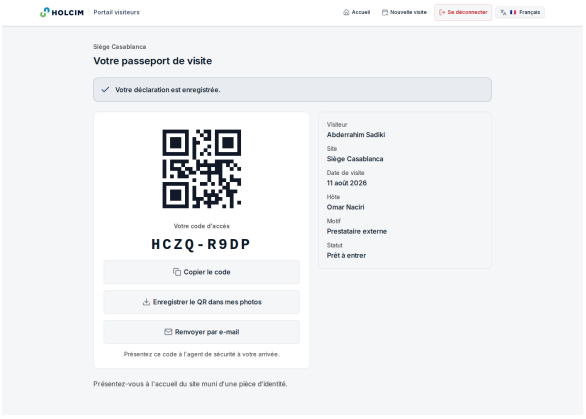


Certification activée puis acceptée

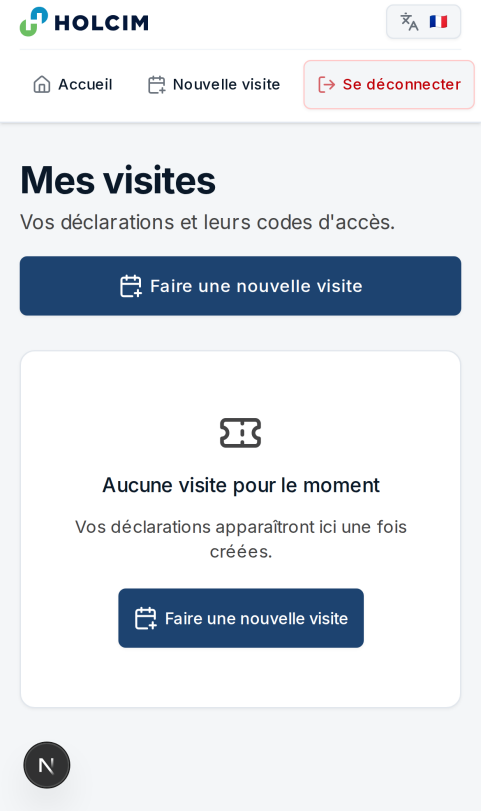
Figure 4.11 – Induction ordinaire — contrôle de consultation et certification

Le visiteur retrouve son passeport à tout moment dans l'espace *Mes visites*, sans avoir à conserver le courriel : s'il se présente sans son téléphone chargé ou sans le message, l'agent peut retrouver la visite par son nom. L'historique lui donne aussi la visibilité sur l'état d'avancement de chaque dossier — déclaré, prêt pour l'entrée, entré, sorti.

Ces preuves proviennent d'un même parcours réellement exécuté. Le code et le QR ne sont pas fabriqués par le navigateur : le serveur les génère après avoir validé l'identité, le site, la date, les référentiels, le consentement et la certification d'induction. Le courriel qui les transporte figure dans la vue d'ensemble des messages (figure 4.29), capturée sur le relais de développement.



Passeport et code QR délivrés



Historique « Mes visites »

FIGURE 4.12 – Passeport visiteur et suivi de ses propres visites

4.6.5 Opérations de guérite

L'espace guérite regroupe les opérations quotidiennes : contrôle du passeport, vérification de l'induction et de l'autorisation, enregistrement de l'entrée sur une guérite du site, puis de la sortie. L'agent peut aussi rechercher un visiteur, effectuer une saisie manuelle, rattraper une entrée non enregistrée et consulter la liste instantanée des présents. Le mode évacuation produit une liste exploitable par site et journalise le déclenchement.

La figure 4.14 montre le geste central du poste. L'agent saisit ou scanne le code du passeport ; la fiche du visiteur s'affiche avec son identité, son motif, son hôte et — c'est le point important — l'état de son induction. Le bouton d'enregistrement de l'entrée n'est actif que si la visite est effectivement à l'état « prêt pour l'entrée » et qu'une guérite est sélectionnée. L'agent n'a donc pas à se souvenir de la règle : l'interface ne lui propose pas une action que le serveur refuserait.

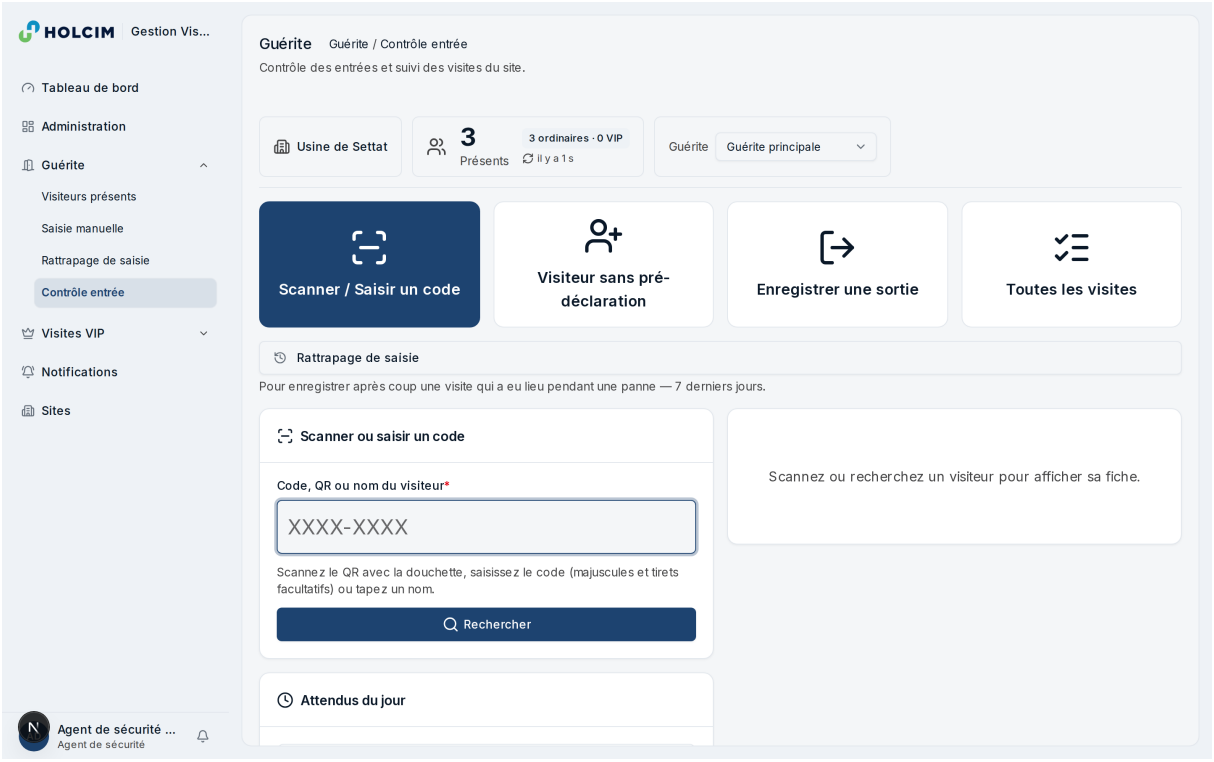


Figure 4.13 – Écran d'accueil de la guérite

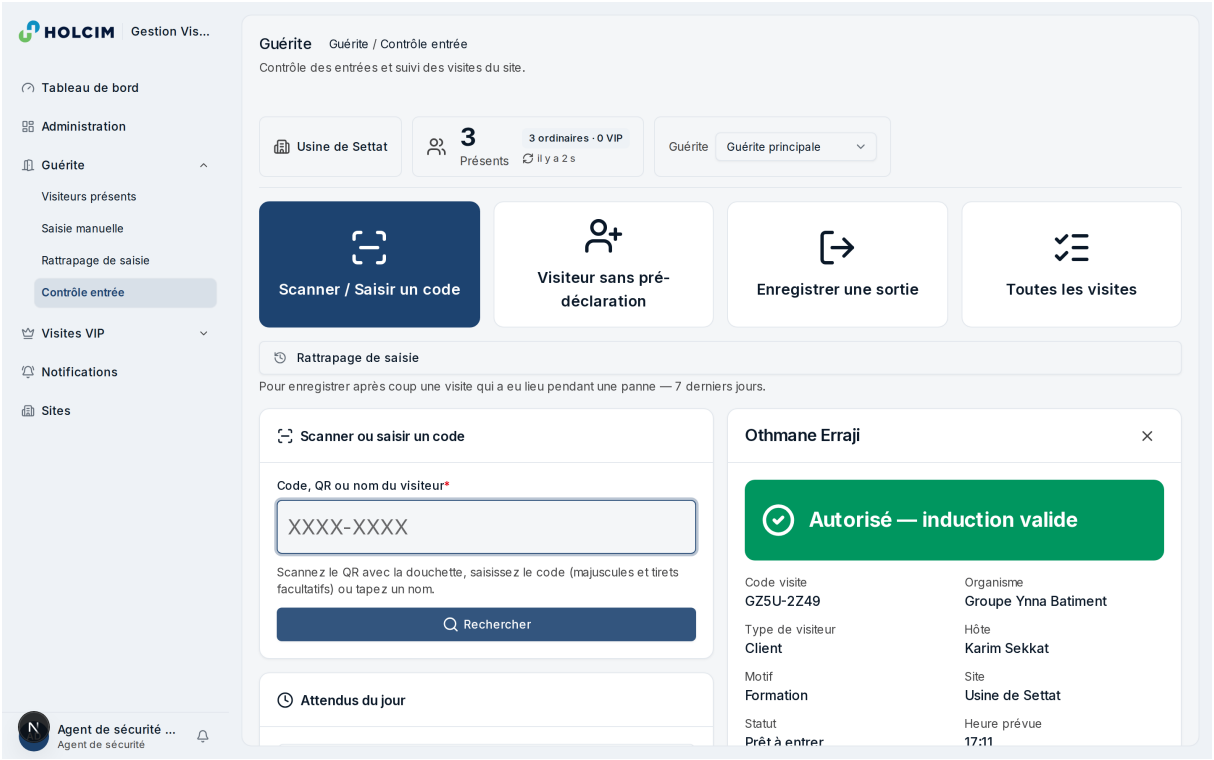
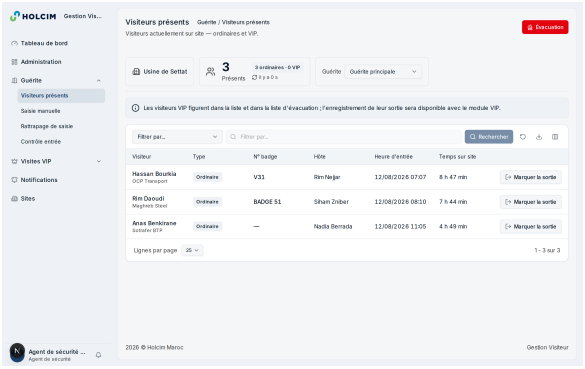
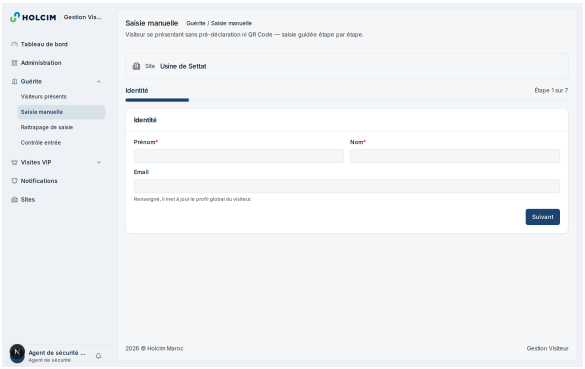


Figure 4.14 – Contrôle d'un passeport avant enregistrement de l'entrée

Le badge est une information, pas une ressource. Le formulaire d'entrée comporte un champ *numéro de badge*, libre et facultatif : l'agent y note le badge physique qu'il vient de remettre, ou ne note rien. Le système ne tient aucun stock de badges, ne leur attribue aucun statut et ne conditionne jamais une entrée à leur disponibilité. Ce choix a été retenu après examen de l'alternative. Faire du logiciel le gardien d'un stock de badges suppose que la base reflète en permanence l'état d'objets manipulés à la main, ce qu'aucune guérite ne peut garantir : il suffit qu'un badge soit rendu sans que la sortie ait été saisie pour qu'il reste marqué occupé, et le visiteur légitime suivant se voit alors refuser l'entrée par l'outil. La contrainte est artificielle, et elle s'exerce précisément au moment où l'agent a le moins de latitude pour la contourner proprement. Le badge physique relève de la sûreté, qui le gère selon ses propres règles ; l'application en conserve la mention parce qu'elle est utile a posteriori, et ne s'arroge pas une autorité qu'elle ne peut pas exercer. La seule donnée réellement obligatoire à l'entrée est la guérite, parce qu'elle est vérifiable et qu'elle situe le passage.



Liste des visiteurs présents



Saisie manuelle d'une visite

Figure 4.15 – Écrans opérationnels de la guérite

La liste des présents est l'écran qui répond directement à la première limite relevée dans l'étude de l'existant : elle donne, à tout instant et sans feuilletage, la population effectivement sur le site.

Le rattrapage mérite une explication, car il traduit une réalité de terrain plutôt qu'un besoin exprimé. Aux heures de forte affluence, une entrée peut être laissée passer sans saisie. Interdire toute régularisation conduirait l'agent à contourner l'outil ; l'autoriser sans trace ruinerait la valeur du journal. La régularisation est donc possible, mais elle est enregistrée comme telle : la visite porte l'horaire réel et le journal conserve le fait qu'elle a été saisie a posteriori, par qui et quand.

Le mode évacuation est la fonction dont l'absence justifiait le projet. Sur déclenchement, le système fige la liste des personnes présentes sur le site, ouvre le document imprimable

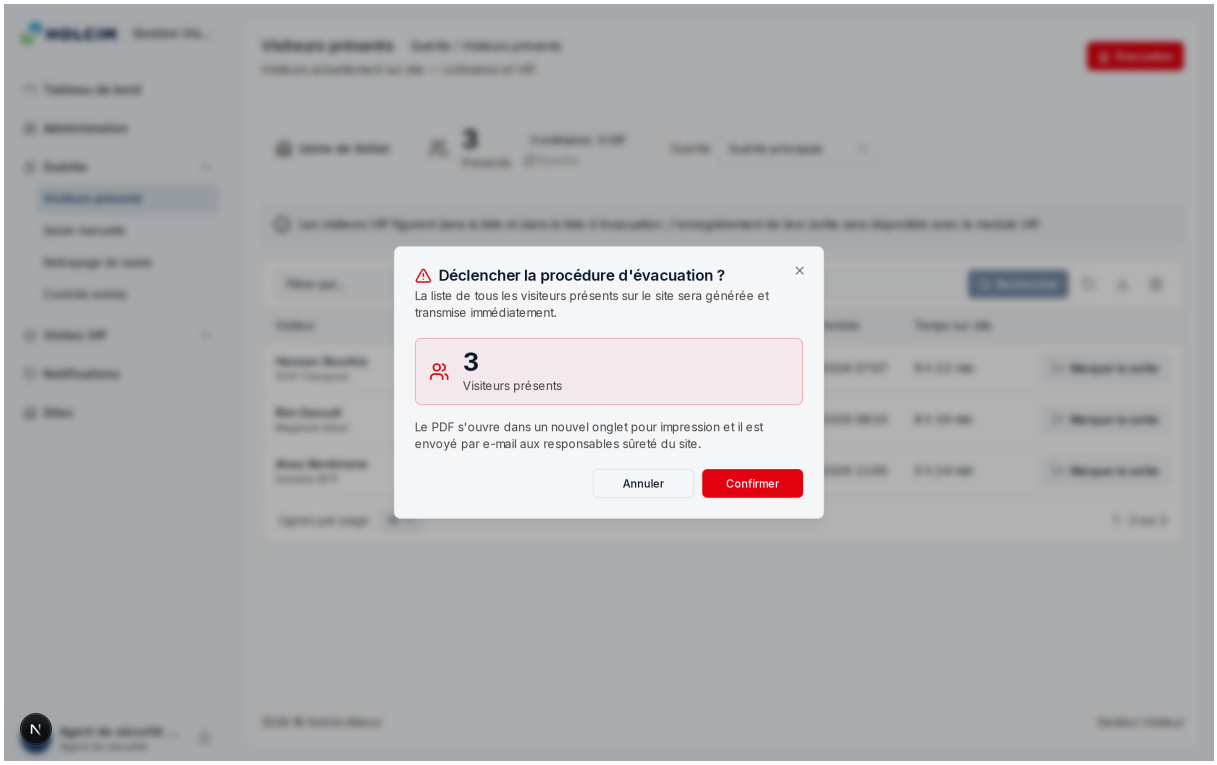


Figure 4.16 – Confirmation avant déclenchement de la procédure d'évacuation

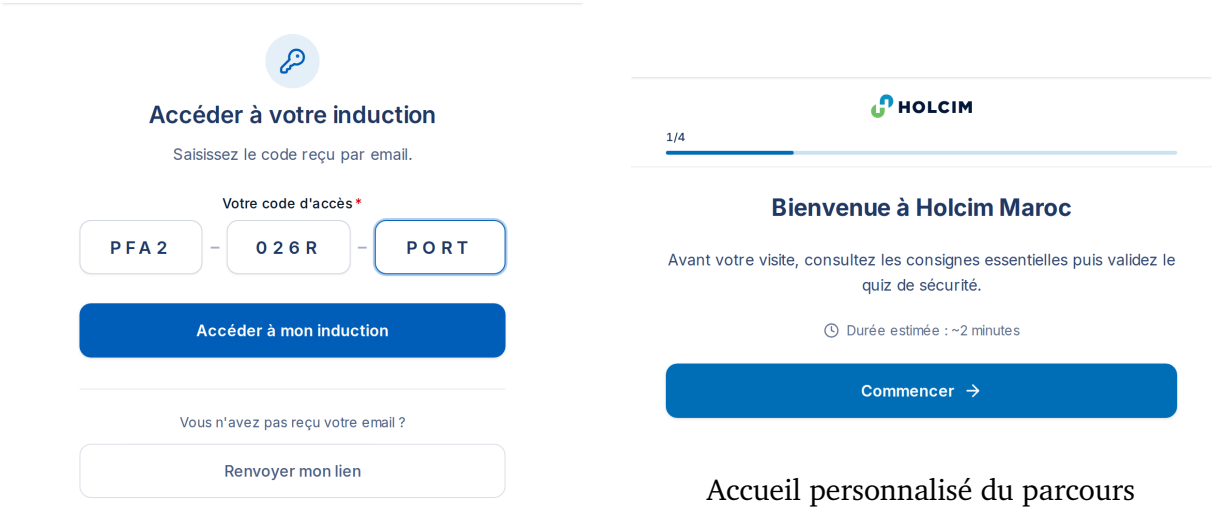
exploitable au point de rassemblement, l'adresse aux responsables sûreté du site, et journalise l'opération avec son auteur et son horodatage. La confirmation intermédiaire visible ci-dessus est délibérée : l'action déclenche un envoi réel et ne doit pas pouvoir partir sur un clic accidentel.

C'est la différence concrète avec le registre papier : la même question — qui se trouve sur le site en ce moment ? — reçoit une réponse en une action au lieu d'un feuilletage, et cette réponse est datée.

4.6.6 Parcours des visites protocolaires

Le parcours VIP est séparé du parcours ordinaire afin de porter ses propres règles. Un déclarant saisit la visite au moyen d'un formulaire guidé, puis le dossier passe par les niveaux de validation configurés. Le visiteur reçoit ensuite un accès à une induction dédiée, composée de contenus configurables et, lorsque le site l'active, d'un quiz. La réussite génère côté serveur un certificat PDF portant un code QR. Le module couvre également le rejet, les motifs de dérogation, le renvoi d'accès, la guérite VIP, l'entrée, la sortie et la traçabilité des décisions.

Une visite VIP de démonstration a été créée et validée pour Casablanca. L'accès a été réalisé au moyen du code imprimable, puis deux questions de sécurité ont été présentées. La configuration de cette exécution locale utilise un quiz court sans média VIP ; le média et la certification du parcours ordinaire sont démontrés dans la figure 4.11.



Saisie du code d'induction

Accueil personnalisé du parcours

Figure 4.17 – Accès mobile à l'induction VIP

Un certificat que l'on peut contrôler. Le code QR imprimé sur le certificat n'est pas décoratif : il pointe vers une page *publique* de vérification, atteignable sans compte. Un agent de guérite qui reçoit un visiteur présentant un certificat imprimé le scanne et obtient immédiatement la réponse du serveur : ce certificat est-il authentique, à quelle visite correspond-il, et est-il encore valable. C'est ce qui distingue un certificat d'une simple attestation imprimable — sans cette page, un document falsifié serait indétectable à l'œil nu. Le jeton contenu dans le QR est imprévisible et ne divulgue que le strict nécessaire au contrôle.

La pièce jointe PDF et le QR sont produits par le backend dans la même transaction que la réussite du quiz. La capture du courriel confirme le résultat dans le relais



Figure 4.18 – Démarrage du quiz de sécurité — Question sans indication de la bonne réponse

local ; comme pour les autres messages, elle ne se substitue pas au test final du SMTP d'entreprise.

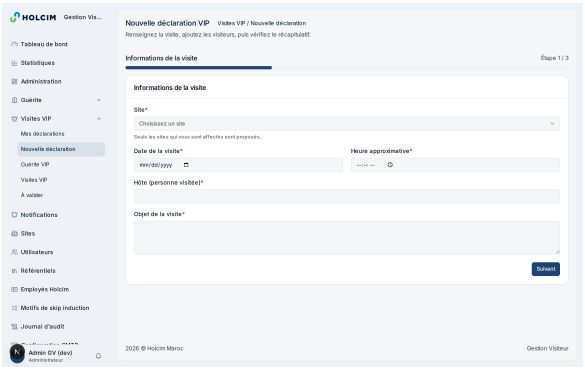


Question sur la conduite d'évacuation

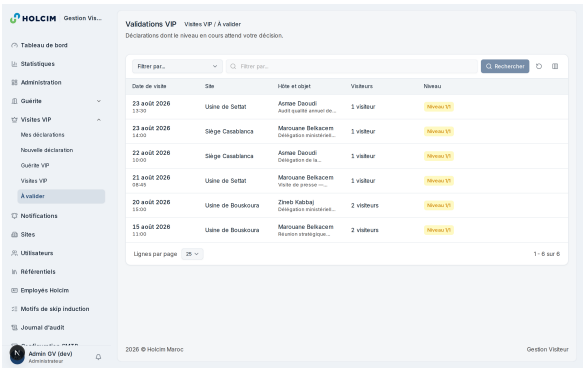


Résultat 2/2, code et QR du certificat

Figure 4.19 – Quiz évalué côté serveur et certificat obtenu



Déclaration guidée



File de validation

Figure 4.20 – Parcours de la visite protocolaire

4.6.7 Administration

L'administration couvre les sites, leurs paramètres et leurs guérites, les référentiels globaux et locaux, les profils, permissions et affectations multi-sites. Elle inclut l'annuaire des employés, l'autocomplétion de l'hôte, les modèles de courriels, la configuration SMTP, les motifs de dérogation et l'export de la matrice des habilitations. Les écrans

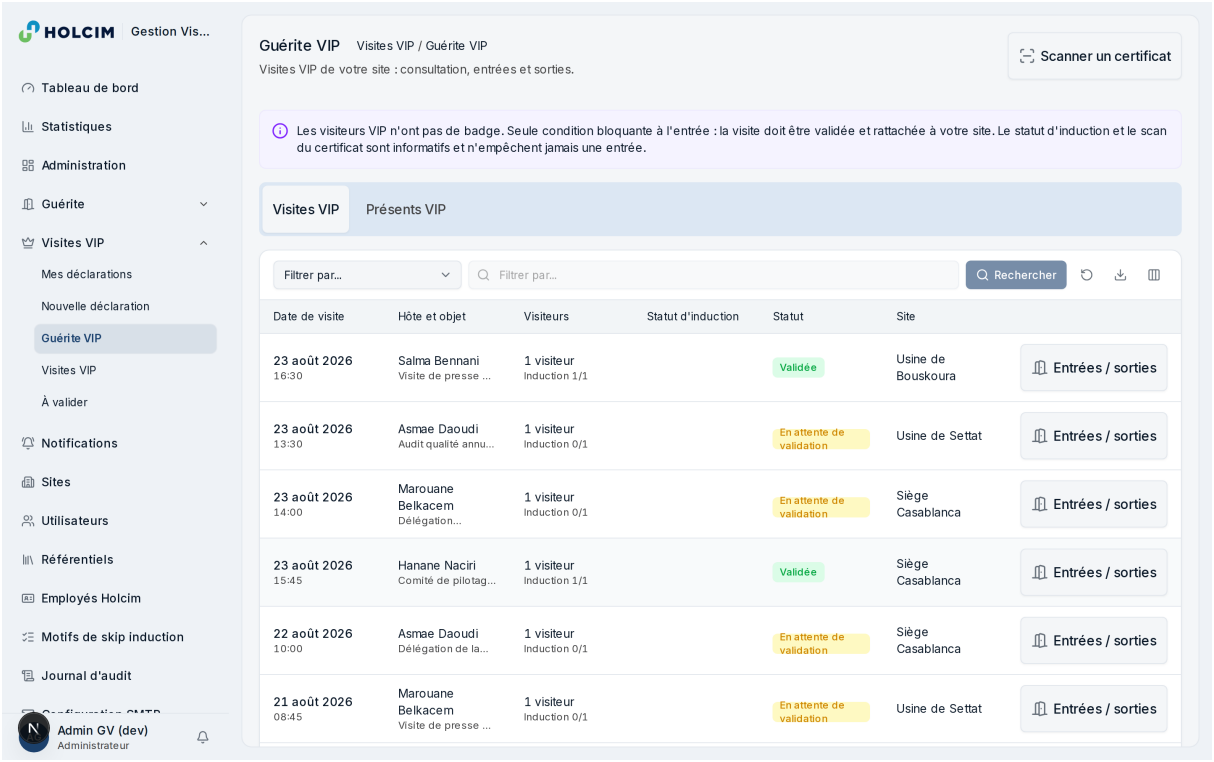


Figure 4.21 – Suivi et accueil des visites protocolaires — Guérite dédiée aux visites VIP

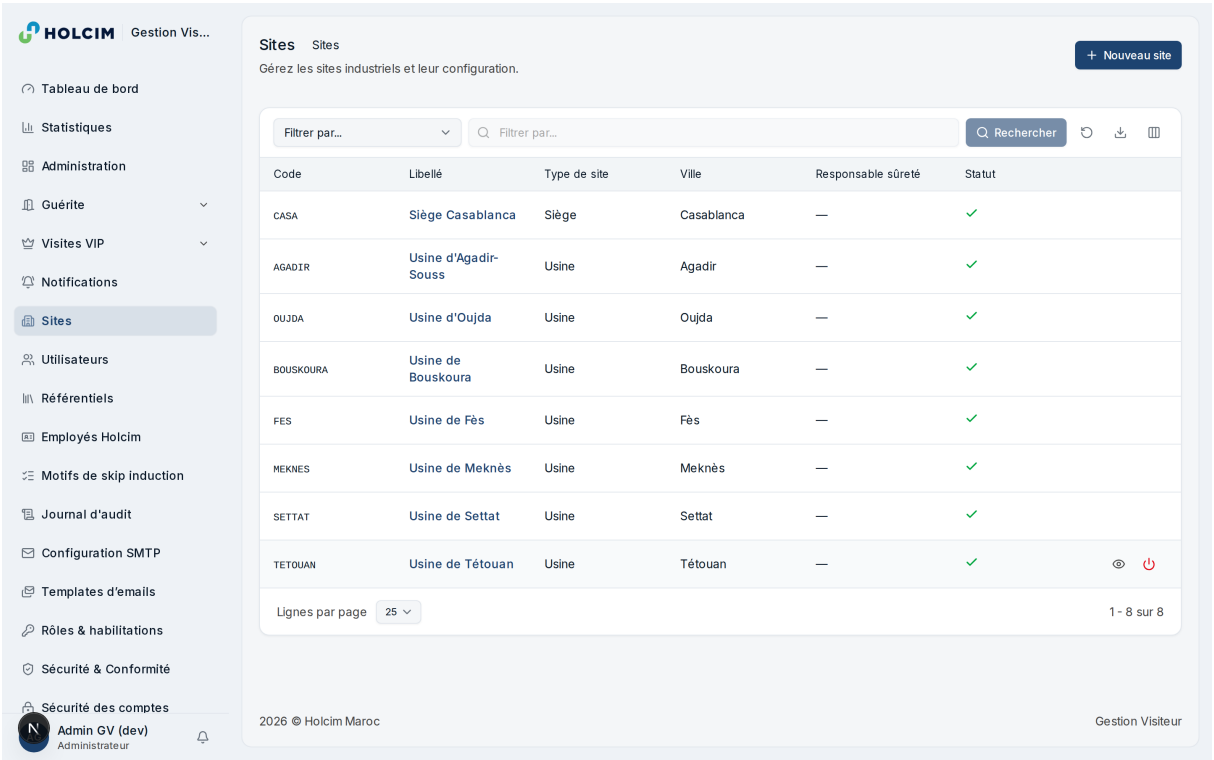


Figure 4.22 – Gestion des sites

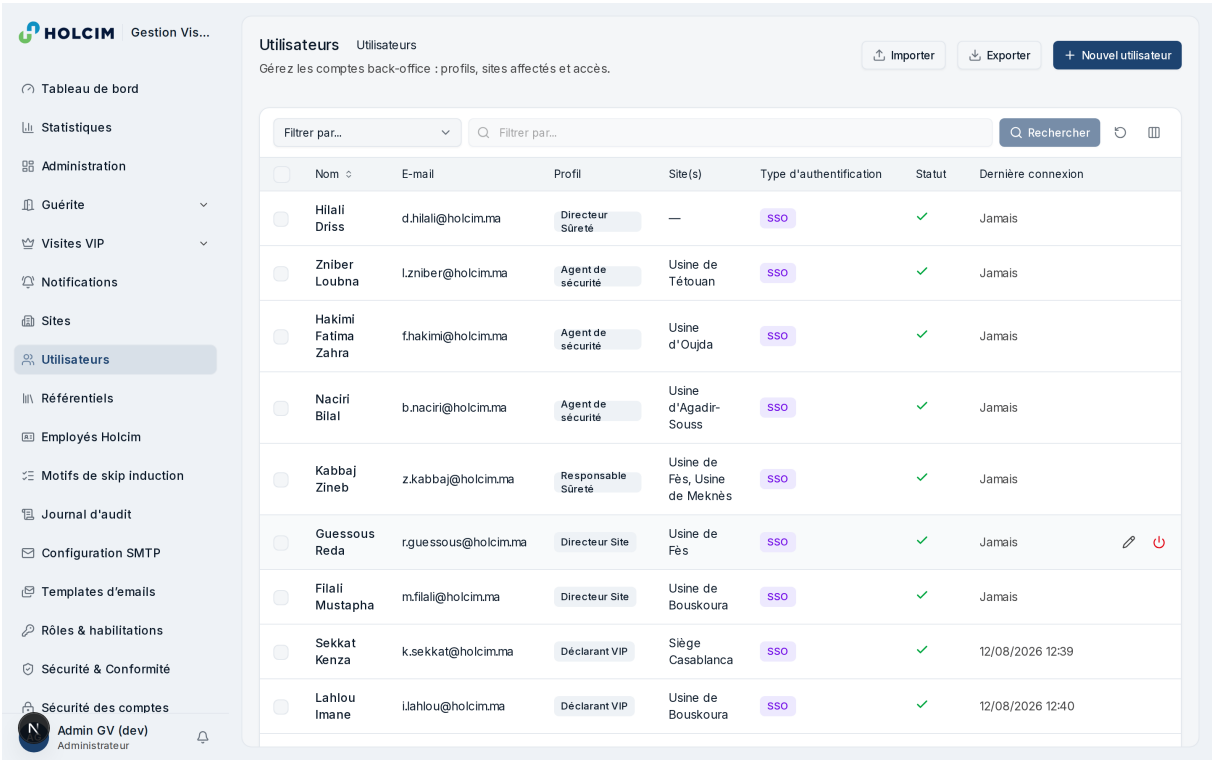
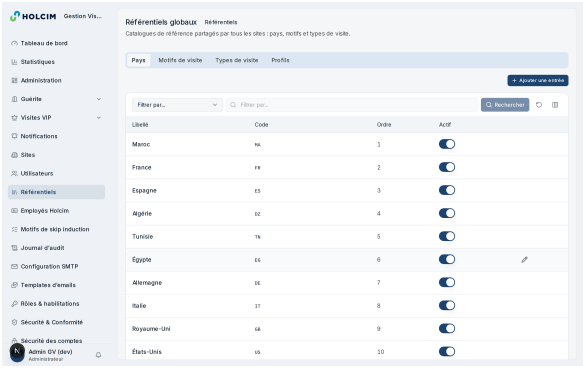
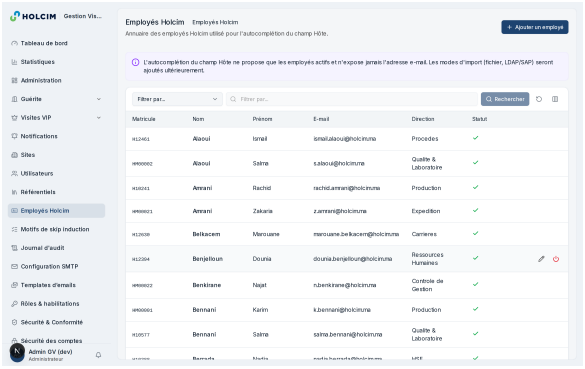


FIGURE 4.23 – Gestion des comptes utilisateurs



Référentiels globaux



Annuaire des employés

FIGURE 4.24 – Administration des référentiels

s'appuient sur des formulaires validés et des listes paginées afin de conserver un comportement homogène.

Les modèles sont administrables plutôt que figés dans le code : l'objet et le corps de chaque message — code de connexion, confirmation de déclaration, invitation à l'induction, certificat — se modifient depuis le back-office, sans livraison applicative.

Chaque site porte sa propre configuration : contenu d'induction, durée minimale de consultation, exigence d'une photographie, quiz VIP actif ou non, mode tablette des

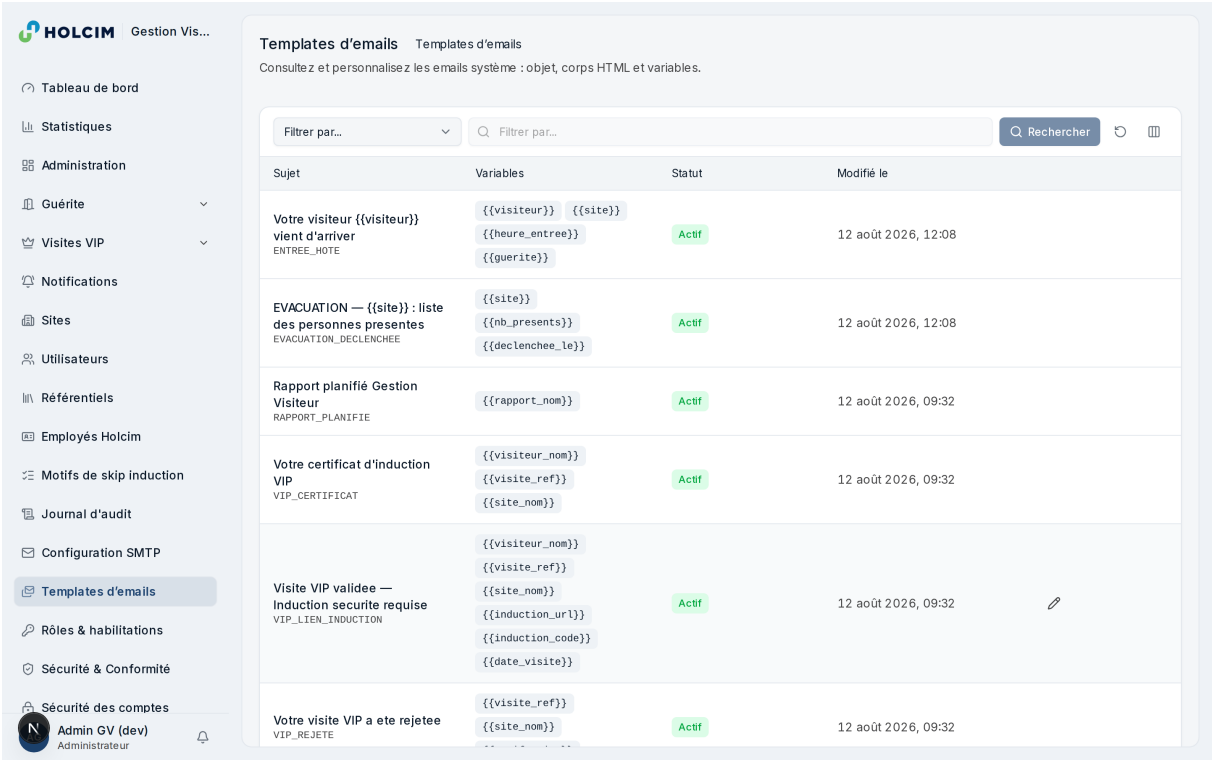


Figure 4.25 – Modèles de courriels utilisés par les parcours

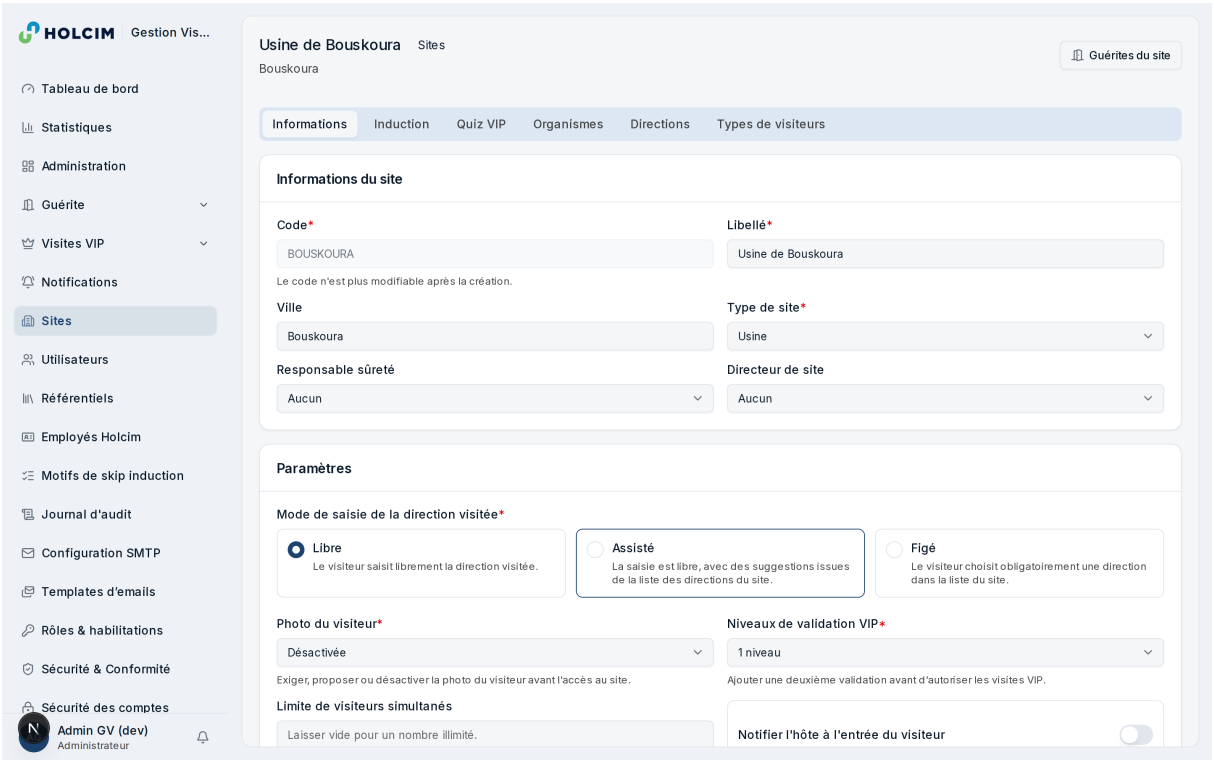


Figure 4.26 – Paramètres et référentiels propres à un site

agents, organismes et directions qui lui sont propres. Deux sites du même groupe peuvent donc appliquer des règles d'accès différentes sans duplication de code.

Les guérites d'un site s'administrent depuis sa fiche. Un onglet de la fiche site gère le référentiel des guérites : création, modification, activation, désactivation et suppression. Le nombre n'est pas contraint — un site peut n'avoir qu'une entrée, un autre en compter vingt entre l'accueil, les poids lourds et une guérite ouverte pour la durée d'un chantier. Le nom est unique au sein d'un site, mais réutilisable d'un site à l'autre : deux implantations peuvent chacune avoir leur « guérite principale » sans que l'une empiète sur l'autre.

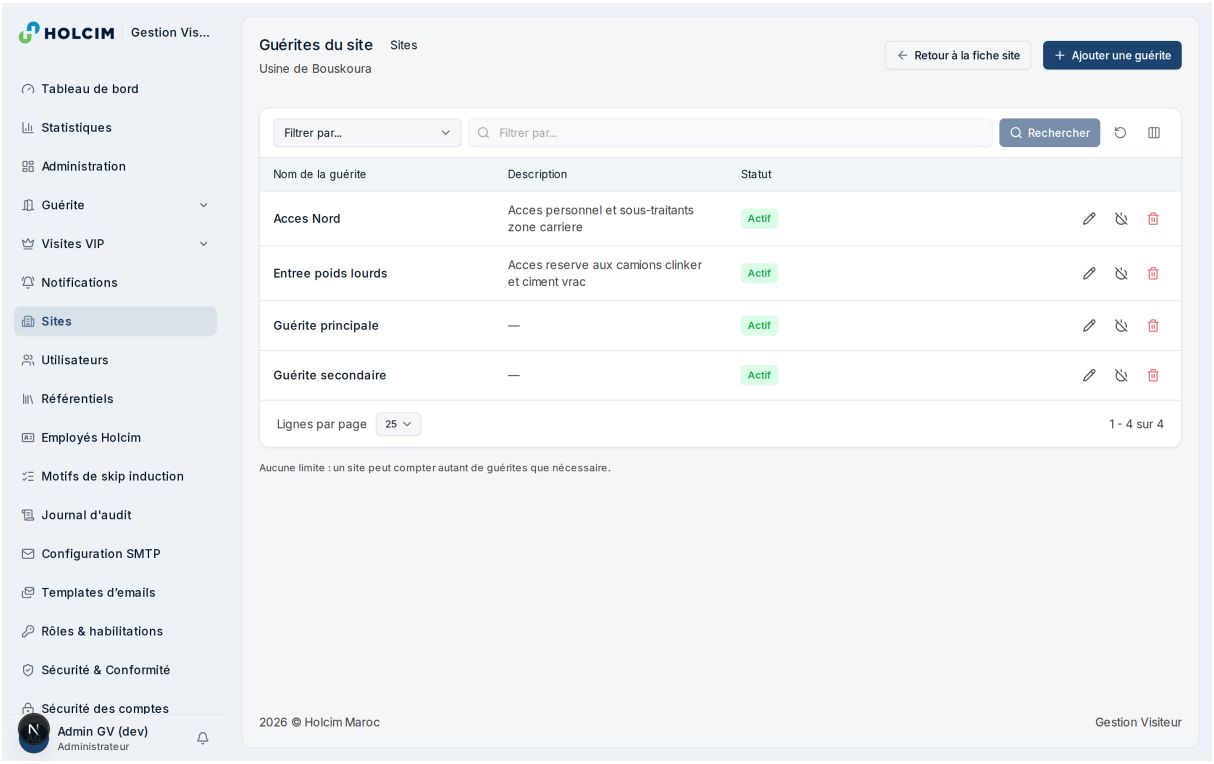
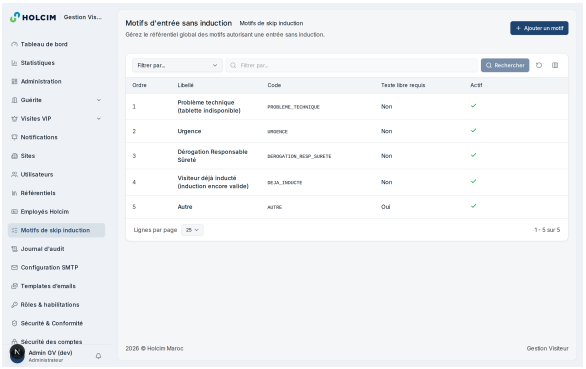


Figure 4.27 – Administration des guérites d'un site : création, activation et désactivation

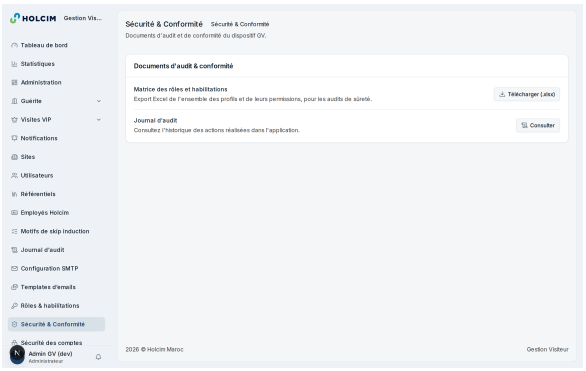
Cet écran compte parce qu'il alimente tout le reste. Le référentiel des guérites est consommé en lecture par l'entrée, la sortie, le rattrapage et la saisie manuelle : c'est lui qui peuple les sélecteurs que l'agent utilise chaque jour, et une entrée ne peut pas être validée sans qu'une guérite soit désignée. Sans écran d'administration, ce référentiel n'aurait pu être alimenté qu'au déploiement, et le nombre de guérites d'un site serait resté figé jusqu'à la livraison suivante.

Deux règles gouvernent le retrait d'une guérite, et leur différence est délibérée. La *désactivation* est toujours possible : la guérite disparaît immédiatement des sélecteurs des agents, mais reste attachée aux visites déjà enregistrées, dont l'historique n'est pas

réécrit. La *suppression définitive* est refusée dès qu'une visite y fait référence ; le système propose alors la désactivation à la place. Le principe est celui appliqué partout ailleurs dans le produit : on ne supprime pas une donnée dont dépend une trace. Chaque opération — création, modification, changement d'état, suppression — est écrite au journal d'audit avec son auteur.



Motifs de dérogation à l'induction



Matrice de conformité des habilitations

Figure 4.28 – Paramétrage des dérogations et contrôle des habilitations

Le premier écran répond à un besoin exprimé tardivement : certaines catégories de visiteurs — les services de secours, par exemple — ne peuvent pas être retenues le temps d'une induction. Plutôt que d'autoriser un contournement libre, l'administrateur déclare la liste fermée des motifs qui en dispensent, et le recours à l'un d'eux est tracé. Le second produit la matrice profils/permissions exportable, utilisée lors des revues d'habilitation.

Un compte oublié est un compte exploitable. L'écran ci-dessus expose le délai au-delà duquel un compte sans connexion est désactivé automatiquement. Le traitement est planifié côté serveur et protégé par un verrou : déployé en plusieurs instances, il ne s'exécute qu'une fois par échéance.

La configuration du serveur d'envoi est elle aussi administrable à chaud : hôte, port, chiffrement, identifiants et adresse d'expédition se modifient sans redémarrage, et un bouton de test envoie un message de contrôle. Les identifiants sont chiffrés en base et jamais renvoyés en clair par l'API. En environnement local le relais pointe vers un serveur de messagerie de test ; en recette, il pointe vers le relais de l'entreprise, la seule différence étant la valeur des paramètres.

La figure 4.29 rassemble en une vue les trois messages produits par une exécution complète : le code de connexion à usage unique, le passeport de visite adressé après la déclaration, et le certificat d'induction VIP avec sa pièce jointe. Ils proviennent tous d'une même session de vérification et démontrent que la chaîne de notification

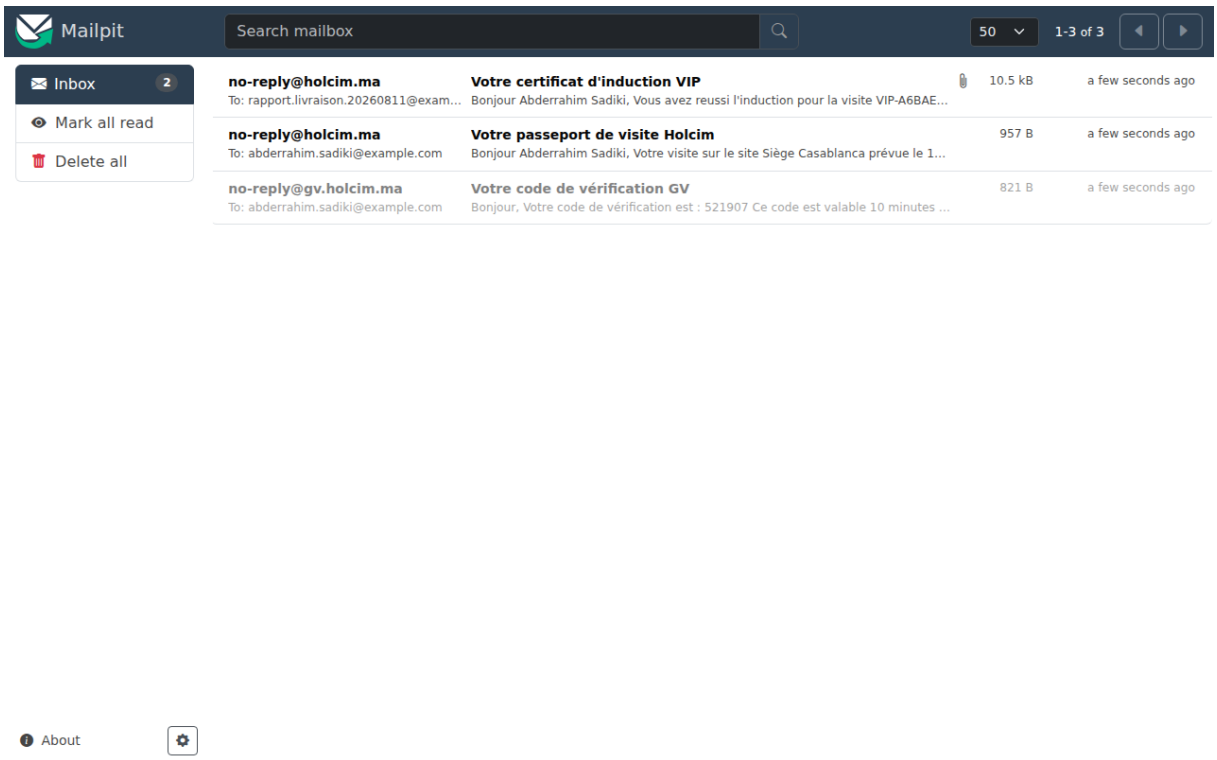


FIGURE 4.29 – Messages émis par la plateforme au cours d'un parcours complet

fonctionne de bout en bout — de la règle métier qui décide de l'envoi jusqu'au message effectivement délivré.

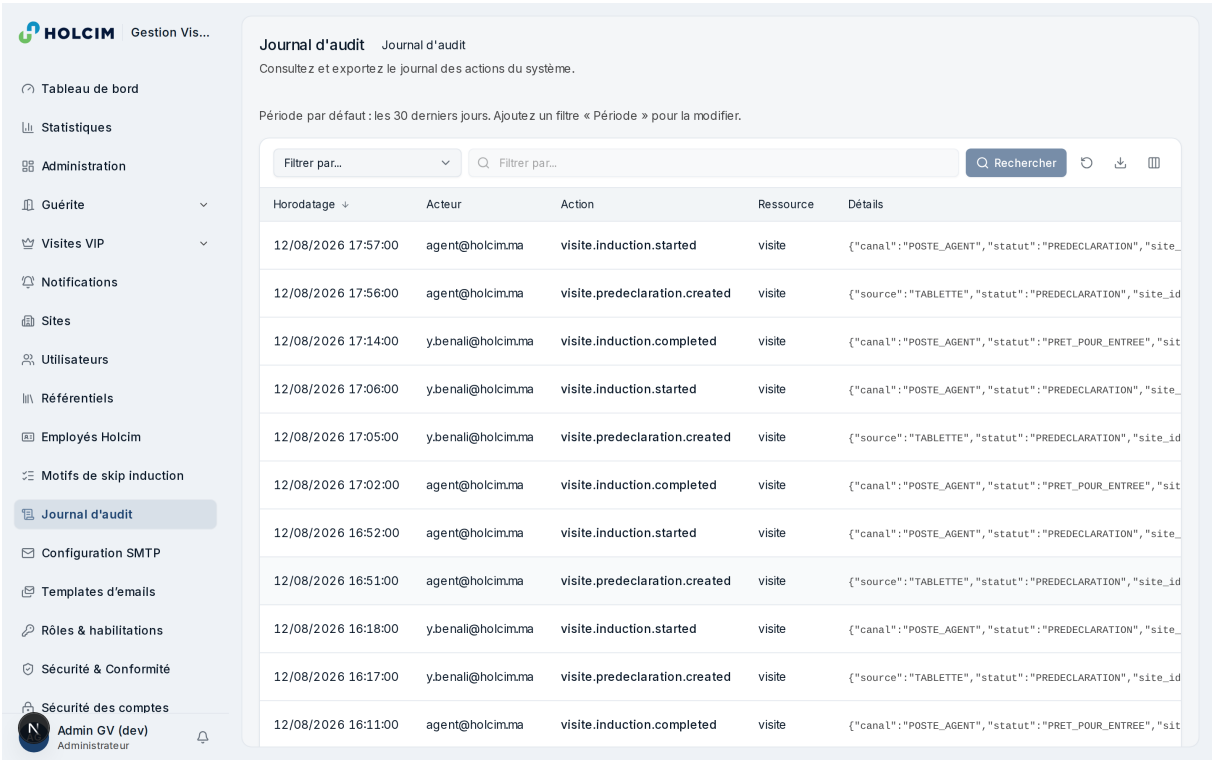


Figure 4.30 – Journal d'audit

4.6.8 Traçabilité et restitution

Chaque modification sensible produit une trace attribuée et horodatée. Le journal d'audit propose des filtres et un export, tandis que l'historique d'une visite restitue les changements successifs. Les tableaux de bord et statistiques appliquent le périmètre de l'utilisateur : vision d'un site pour un responsable local, vision consolidée pour la direction sûreté. Les rapports planifiés génèrent des exports CSV ou PDF et les adressent aux destinataires configurés. Le back-office dispose enfin d'une boîte de notifications avec redirection vers les objets concernés, d'une recherche globale accessible au clavier, d'une carte des sites et d'un thème clair ou sombre suivant la préférence du poste. Les points de santé du backend et les journaux structurés facilitent la supervision technique.

Le journal répond à la question qu'un registre papier laisse sans réponse : *qui a autorisé quoi, et quand*. Chaque ligne porte l'action, son auteur, l'objet concerné, l'horodatage et un identifiant de corrélation qui permet de retrouver la trace technique correspondante dans les journaux du serveur. Le mécanisme est celui décrit plus haut : publication dans la transaction métier, écriture différée. L'utilisateur n'attend jamais l'écriture du journal, et une opération annulée ne laisse pas de trace fantôme.

Au niveau d'un dossier, l'historique restitue la même information sous forme chronologique : chaque changement d'état de la machine décrite au chapitre précédent y figure

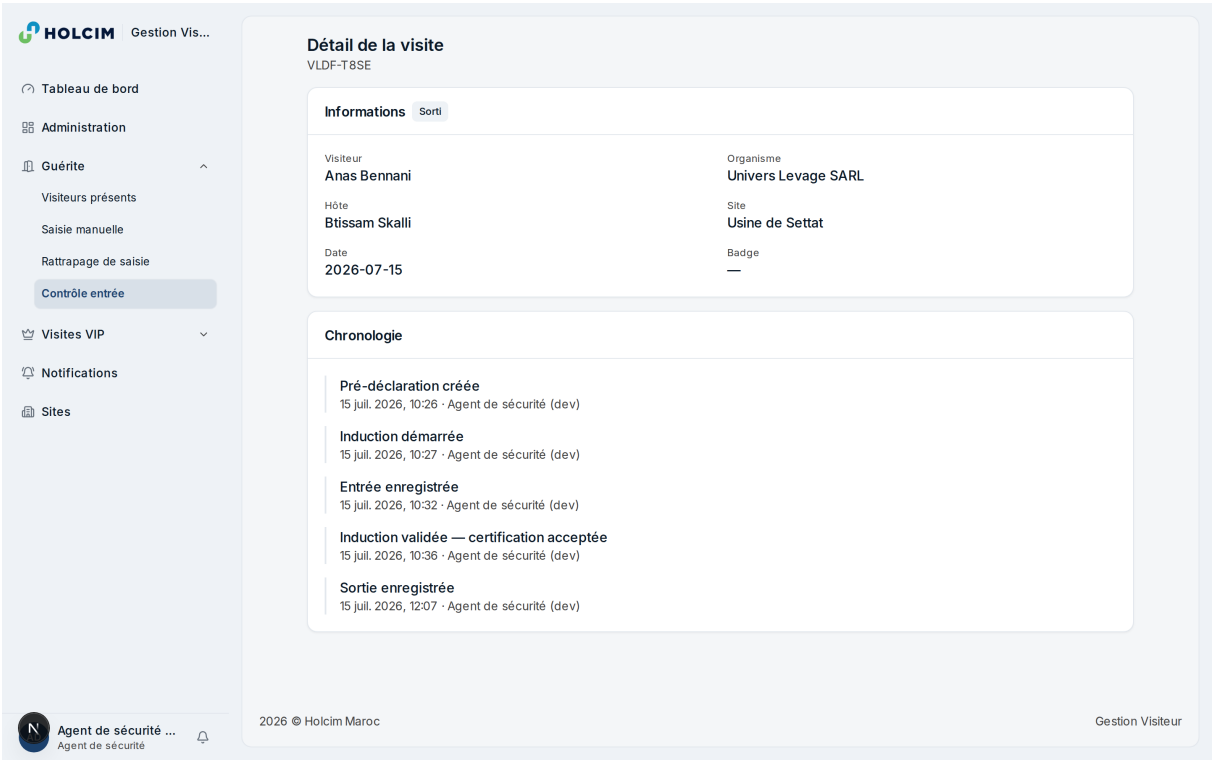


Figure 4.31 – Historique des transitions d'une visite

avec son auteur. C'est ce qui rend le cycle de vie non seulement contraint, mais aussi *explicable* après coup.

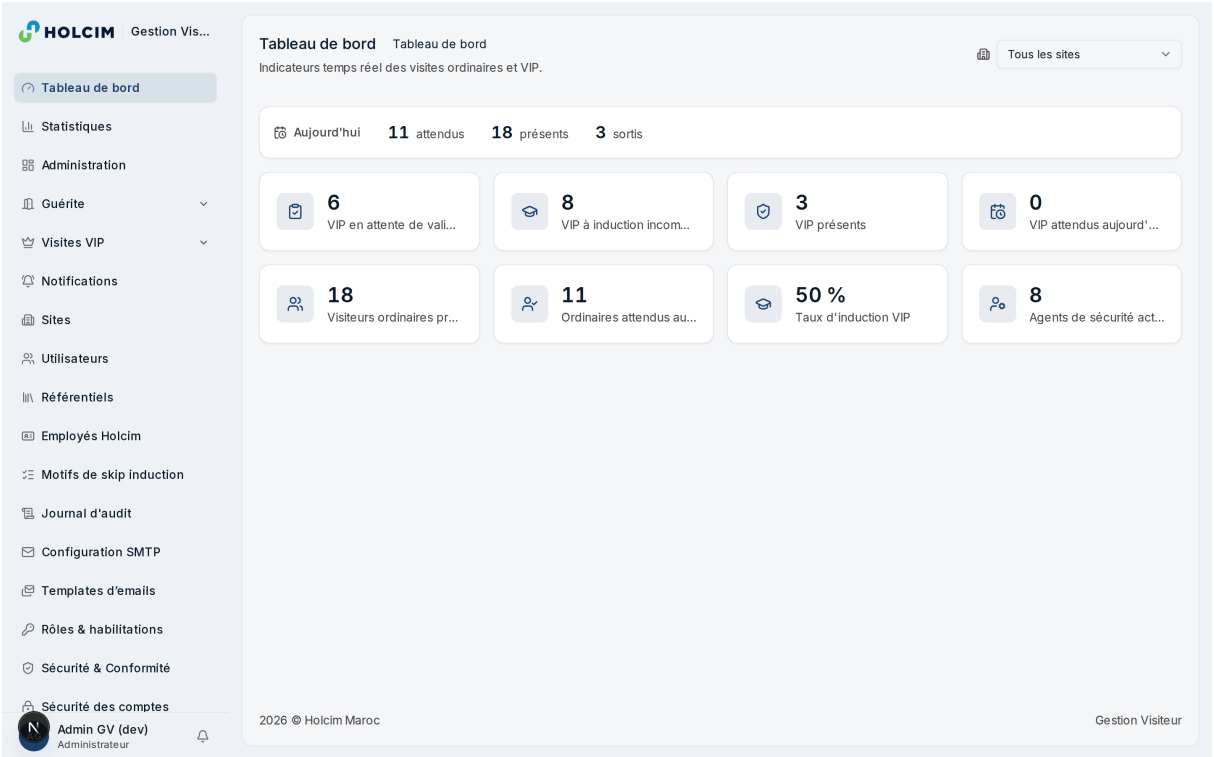


Figure 4.32 – Tableau de bord des indicateurs

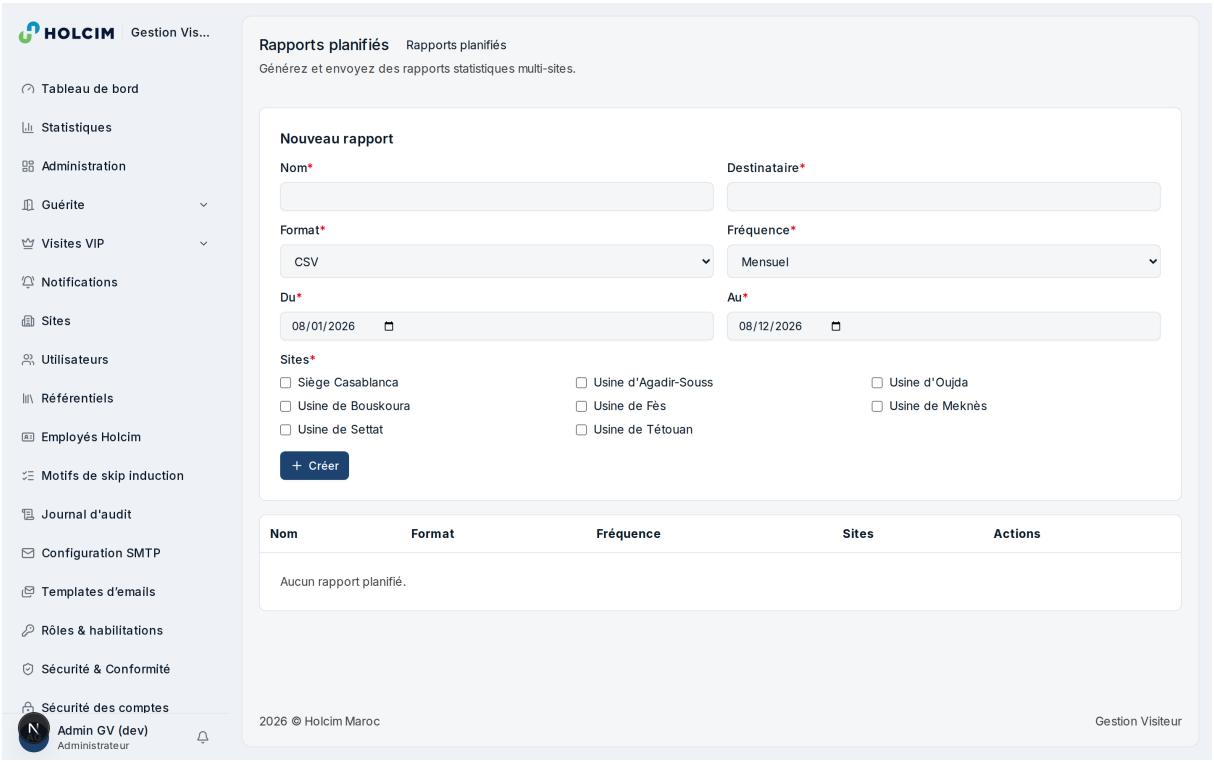


Figure 4.33 – Configuration et suivi des rapports planifiés

4.6.9 Un seul poste, deux rendus

Le back-office est conçu pour un écran de bureau ; la guérite, elle, travaille debout, parfois sur une tablette posée au comptoir. Le poste de guérite n'est pourtant pas une seconde application : c'est le même code, qui adopte deux rendus selon l'équipement du site. La fiche du site porte un paramètre *mode tablette* ; lorsqu'il est activé, les écrans de guérite passent en rendu tactile — cibles d'au moins cinquante-six pixels, espacement accru, typographie agrandie, action principale en pleine largeur au bas de l'écran. Lorsqu'il est inactif, la même interface reste dense, adaptée à un poste de bureau et à une douchette. L'agent ne change jamais d'écran ni de mode opératoire ; seule la taille des cibles change, et elle est décidée à l'administration, non à l'exécution.

Ce parti pris d'ergonomie tient à ce qu'est réellement le poste. L'agent de sécurité l'utilise debout, souvent d'une seule main, sans être un utilisateur avancé, avec un visiteur qui attend devant lui. Un écran qu'il faut lire pour comprendre ce qu'il faut faire est un écran qui ralentit le flux. L'interface répond donc d'abord à la question « que dois-je faire maintenant » : une action principale par écran, des tuiles d'accueil qui correspondent aux quatre situations du quotidien, et une couleur qui porte du sens plutôt que de la décoration — vert pour autorisé, orange pour un point à vérifier, rouge pour bloqué. La décision se lit avant le texte.

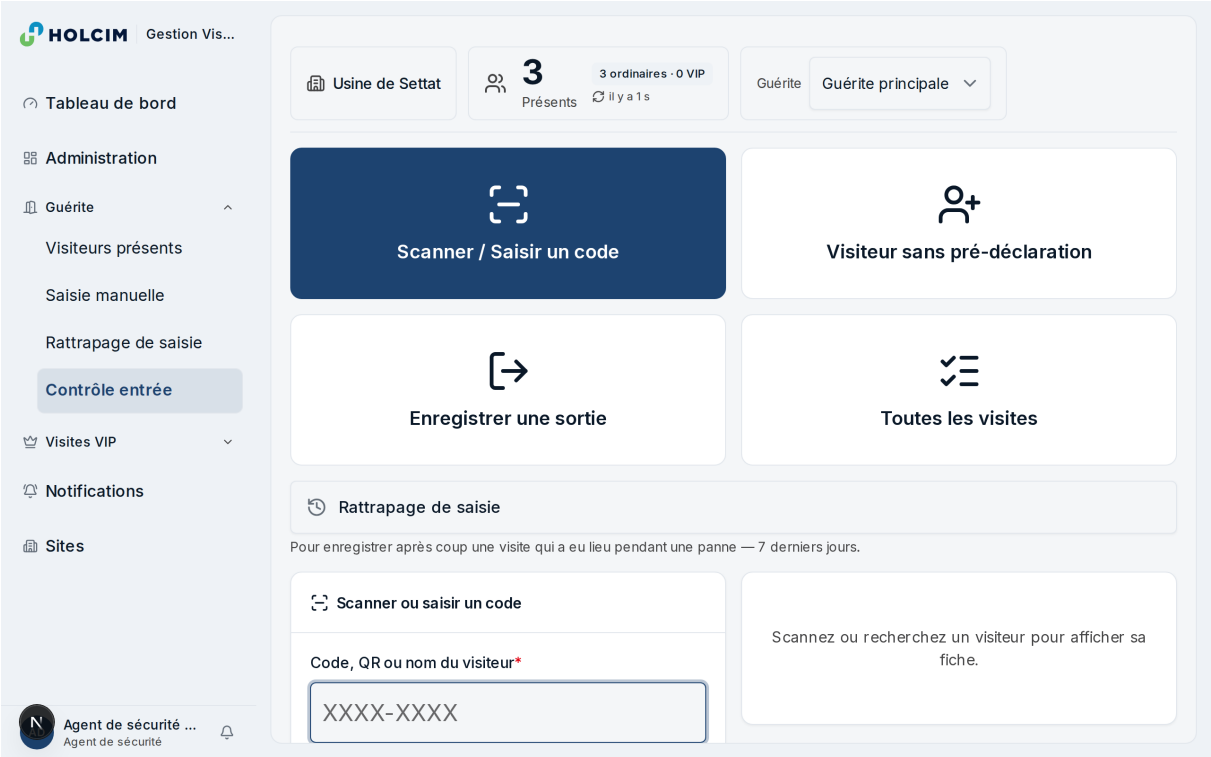


Figure 4.34 – Poste de guérite en rendu tactile, activé par le paramètre « mode tablette » de la fiche site

La saisie d'un visiteur non déclaré se fait sur ce même poste. Le visiteur qui se présente sans s'être déclaré est traité d'un bout à l'autre sur l'appareil de l'agent, dans la même session : l'agent saisit l'identité, le type de visiteur, l'organisme, l'objet de la visite et l'éventuel véhicule en étapes successives, puis, si le site exige une induction, il tend l'appareil au visiteur. Le poste passe en mode plein écran dédié le temps que le visiteur consulte le contenu de sécurité et certifie l'avoir compris, puis rend la main à l'agent sur l'écran de récapitulatif, la visite étant désormais prête pour l'entrée. Un bouton d'annulation reste disponible à tout moment, et une inactivité prolongée ramène l'écran à l'étape précédente sans rien perdre de ce qui a été saisi.

Il n'existe donc ni appareil séparé, ni file d'attente, ni verrou de concurrence entre postes. Le gain est réel : dès lors qu'on suppose un second appareil en libre-service, il faut gérer une file, synchroniser deux écrans et arbitrer les accès concurrents — trois mécanismes qui n'existent que pour compenser la séparation, et autant d'états supplémentaires à tester. Le poste étant unique, il est simplement occupé le temps de l'induction ; si le site compte plusieurs guérites, un autre agent traite un autre visiteur sur son propre poste, sans ressource partagée. Le visiteur, lui, n'attend pas son tour devant une borne : l'agent lui tend l'appareil, et le parcours se poursuit.

4.7 Bilan des fonctionnalités livrées

Table 4.5 – Synthèse des livrables fonctionnels

Domaine	Résultat livré
Socle et sécurité	Séparation des populations, SSO, OTP, profils, permissions et cloisonnement par site
Visiteur ordinaire	Pré-déclaration, QR de site, induction, passeport, entrée, sortie et historique
Visites VIP	Déclaration guidée, validation, induction, quiz, certificat PDF/QR et guérite VIP
Administration	Sites, guérites, comptes, référentiels, annuaire, import, exports, SMTP et modèles de courriels
Opérations	Présents, saisie manuelle, rattrapage, rendu tactile, refus d'accès et évacuation
Restitution	Tableaux de bord, statistiques, carte, rapports PDF/CSV et exports d'audit
Exploitation	Audit, notifications, supervision, images Docker, pipeline GitLab et chart Helm

Au 30 août 2026, les cinq lots fonctionnels sont intégrés et la plateforme s'exécute de bout en bout, l'ensemble des parcours décrits dans ce chapitre ayant été déroulé dans un navigateur contre un serveur réel, puis rejoué sur l'environnement de recette de l'entreprise.

4.8 Difficultés rencontrées et solutions apportées

4.8.1 Comprendre un périmètre métier étendu

La première difficulté a été d'assimiler seul un domaine réunissant des acteurs, des règles de sécurité et des parcours très différents. Le backlog a été découpé en lots cohérents, puis en récits suffisamment petits pour être validés chaque semaine. Les retours réguliers de l'encadrant ont permis de corriger rapidement les interprétations fonctionnelles.

4.8.2 Conduire seul un projet complet

Le même développeur devait intervenir sur le backend, les deux portails, la base, les tests et le déploiement. Une organisation uniforme du dépôt, des contrats OpenAPI générateurs de clients et des composants réutilisables ont limité la dispersion. Claude Code a été utilisé comme outil d'assistance pour accélérer les tâches répétitives ; ses propositions étaient systématiquement relues et vérifiées.

4.8.3 Respecter le délai sans sacrifier la qualité

La contrainte de temps a été traitée par des sprints hebdomadaires, une priorisation par dépendances et l'automatisation des contrôles. Les tests unitaires, d'intégration, d'architecture et d'interface étaient exécutés en continu, de sorte qu'une régression était détectée au plus près de sa création.

4.8.4 L'évaluation sécurisée du quiz

Une première version envoyait au navigateur la liste des questions *et* l'indication de la bonne réponse, l'évaluation étant faite localement. Il suffisait d'ouvrir les outils de développement du navigateur pour lire les réponses. La projection envoyée au client a été modifiée pour retirer cette indication, et l'évaluation a été déplacée côté serveur.

4.8.5 Éviter le double envoi des rapports planifiés

Le mécanisme d'envoi périodique, exécuté toutes les heures, ne comportait aucun verrou. Déployé en plusieurs instances, chacune traitait les mêmes échéances et les destinataires recevaient le rapport en double. Un verrou consultatif, pris en base de données pour la durée de la transaction, garantit désormais qu'une seule instance traite un tour donné.

4.8.6 Maîtriser l'ordre des opérations et la transaction

Toujours sur les rapports, l'envoi des courriels avait lieu dans la même transaction que le déplacement du curseur d'échéance. Une panne du serveur de messagerie annulait donc les deux, et le tour suivant renvoyait des courriels déjà partis. Le curseur est désormais déplacé et validé *avant* tout envoi, et chaque rapport est traité dans sa propre transaction, de sorte qu'une panne sur l'un n'affecte pas les autres.

Conclusion

La réalisation étant décrite, le chapitre suivant traite de la vérification et du déploiement.

CHAPITRE 5

Tests, qualité et déploiement

Introduction

> *Ce chapitre expose la stratégie de vérification adoptée, les mécanismes de contrôle automatique de l'architecture, et la chaîne de déploiement.*

5.1 Stratégie de tests

La vérification s'organise en cinq niveaux, du plus rapide au plus proche de l'usage réel.

Une exécution complète a été réalisée le 30 août 2026 sur l'environnement local décrit au chapitre précédent. Elle a validé les tests Java, les tests des deux portails, le typage TypeScript et le parcours OTP réel à travers le navigateur et le serveur de messagerie. Le scénario E2E ne simule pas le code : il déclenche l'envoi, lit le courriel reçu, saisit le code puis contrôle la session obtenue.

Table 5.1 – Niveaux de test et périmètre couvert

Niveau	Outil	Ce qu'il vérifie
Test unitaire	JUnit, Vitest	Une règle métier isolée, sans base ni réseau
Test de composant	Testing Library	Un écran réagit correctement aux actions de l'utilisateur
Test d'intégration	Testcontainers	Le code fonctionne contre une vraie base de données
Test d'architecture	ArchUnit	Les frontières entre modules et les conventions sont respectées
Test de bout en bout	Playwright	Un parcours complet fonctionne dans un vrai navigateur

5.1.1 Le test d'architecture, un niveau à part

Un niveau mérite d'être détaillé car il est moins courant. Les tests d'architecture ne vérifient aucun comportement : ils inspectent le code lui-même et refusent la compilation lorsqu'une convention structurelle est violée. Treize règles sont ainsi vérifiées automatiquement, parmi lesquelles :

- aucun module ne référence l'intérieur d'un autre module ;
- chaque point d'entrée de l'API déclare une permission requise, ou porte une justification explicite s'il est public ;
- chaque table déclare sa position vis-à-vis du cloisonnement ;
- aucun horodatage persistant n'est stocké sans fuseau ;
- chaque contrôleur implémente l'interface engendrée à partir du contrat d'API.

L'intérêt de ce mécanisme est qu'il transforme une convention — qu'un relecteur peut oublier de vérifier — en un échec de compilation, immédiat et sans ambiguïté.

5.2 Intégration continue

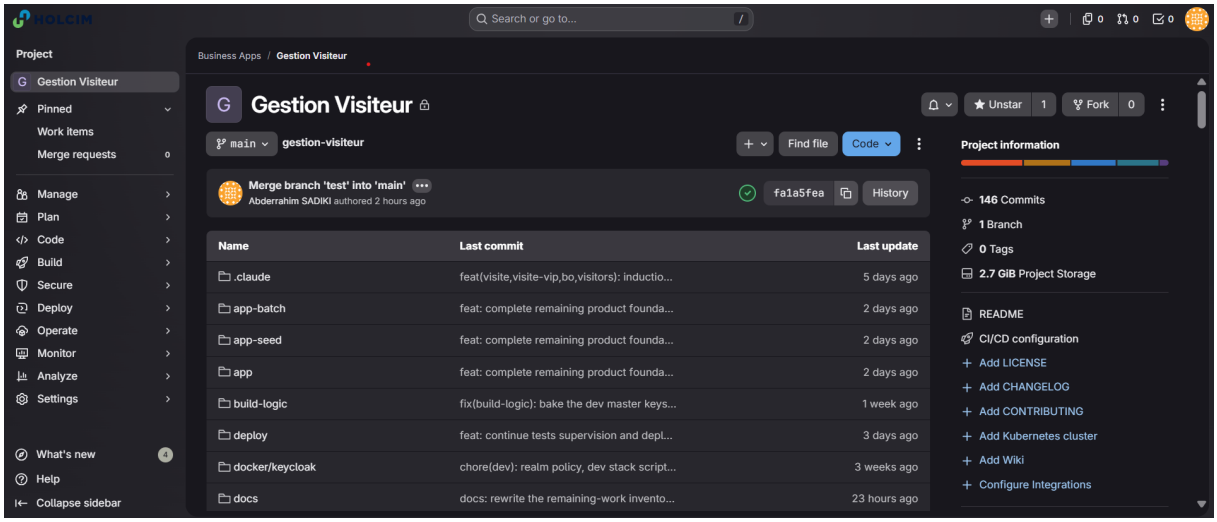


Figure 5.1 – Dépôt du projet et historique des contributions

Le code est versionné sur le GitLab d'entreprise. La figure 5.1 montre le dépôt tel qu'il se présente à la livraison : **161 validations** sur la durée du stage, soit une moyenne

de quatre par jour ouvré. Ce rythme matérialise le caractère incrémental de la démarche — un dépôt ne comportant que quelques validations massives trahirait l'inverse. L'arborescence y est visible et correspond à celle décrite plus haut ; on y distingue notamment le répertoire `.claude`, qui porte le dispositif de règles et de procédures décrit en section 1.4.2.

Chaque modification déposée sur le dépôt déclenche automatiquement une chaîne GitLab CI composée de quatre étapes : test, construction, production des images et publication du paquet de déploiement. Les pipelines s'exécutent sur les branches, les tags et les demandes de fusion.

Table 5.2 – Étapes du pipeline GitLab CI

Étape	Jobs principaux	Contrôles et livrables
Test	Backend, BO, Visitors	JUnit, intégration Testcontainers, typecheck TypeScript, Vitest et Playwright conditionnel
Build	Backend, BO, Visitors	JAR Spring Boot et bundles autonomes Next.js
Image	Backend, BO, Visitors	Images OCI publiées avec le SHA du commit et le nom de branche
Package	Helm	Validation, empaquetage et publication du chart dans un registre OCI

Les jobs Java utilisent une image Gradle avec JDK 21 et mettent en cache les dépendances Gradle. Les jobs frontend utilisent Node.js 22, activent pnpm 9 et partagent un cache fondé sur le fichier de verrouillage. Les rapports JUnit, rapports Playwright et couvertures sont conservés comme artefacts afin de faciliter le diagnostic d'un échec.

La figure 5.2 montre une exécution réelle : douze tâches réparties sur les quatre étapes, toutes au vert, en trente-trois minutes. Elle se lit de gauche à droite comme une chaîne de filtres. La modification est d'abord *vérifiée* — tests du serveur, tests d'intégration, tests des deux portails —, puis *compilée*, puis *transformée en images* déployables, et enfin *empaquetée* pour Kubernetes. En cas d'échec, GitLab interrompt la chaîne au premier contrôle non conforme : une version défectueuse n'atteint jamais l'étape de publication. C'est ce qui distingue une chaîne d'intégration continue d'un simple script de compilation.

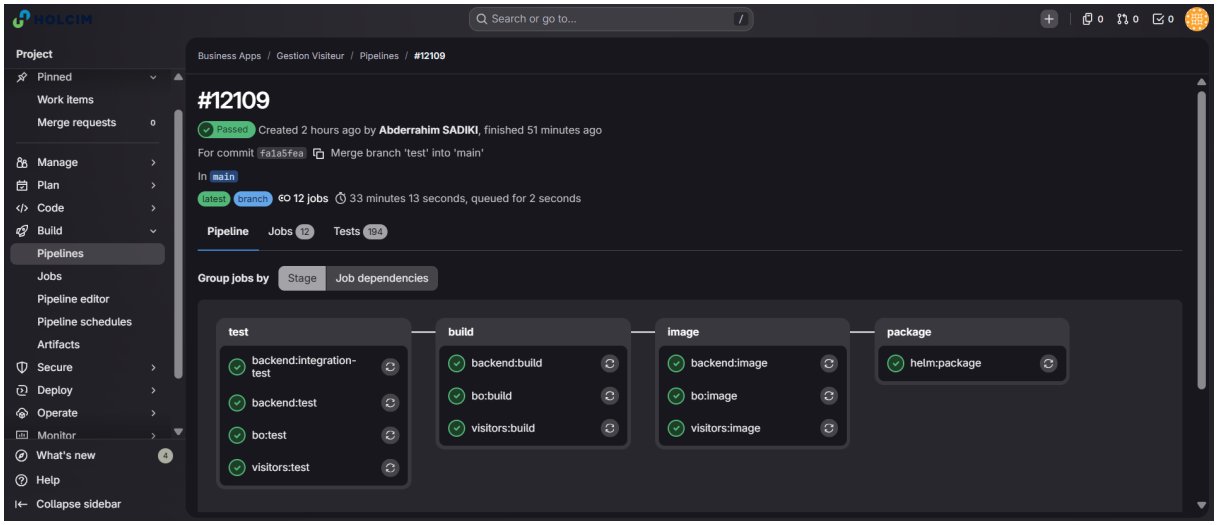


Figure 5.2 – Pipeline GitLab réussi avec les étapes test, build, image et package

5.2.1 Construction et publication des images

L'image backend est produite par Jib après la génération du JAR. Elle est publiée sous deux références : le SHA immuable du commit et le nom de la branche. Les images des portails sont construites par Kaniko, ce qui évite de donner un démon Docker privilégié aux jobs GitLab. Le cache de couches accélère les reconstructions et chaque portail possède son propre dépôt d'images.

Harbor, le registre du groupe. Les images produites ne sont pas conservées sur la machine qui les fabrique : elles sont poussées vers **Harbor**, le registre de conteneurs de l'entreprise. Un registre est à une image ce qu'un dépôt Git est à du code — un entrepôt versionné, adressable et contrôlé. Les trois images du projet y occupent chacune leur emplacement sous `staging/visitor/gv-backend`, `. . ./gv-bo` et `. . ./gv-visitors`.

Le choix d'un registre interne plutôt que d'un service public découle de la même contrainte que le reste : les images embarquent le code de l'entreprise et ne doivent pas quitter son infrastructure. Harbor sert également de dépôt aux *charts* Helm, publiés au format OCI — le même standard que les images, ce qui évite d'exploiter deux systèmes de distribution distincts.

Les identifiants du registre sont fournis par les variables protégées de GitLab CI. Ils ne sont ni écrits dans le dépôt ni intégrés aux images. Le chart Helm est contrôlé par `helm lint`, empaqueté avec une version liée au numéro du pipeline, puis publié.

Le registre est la sortie concrète du pipeline : chaque exécution réussie y dépose trois images étiquetées par l'empreinte du commit. C'est cette étiquette qui relie une image déployée à la ligne de code exacte dont elle provient — sans elle, savoir ce qui tourne réellement en production relève de la reconstitution.

5.3 Déploiement

5.3.1 Images Docker multi-stage

L'application serveur et les deux portails sont empaquetés en images distinctes. Chaque image ne contient que ce qui est nécessaire à l'exécution : les outils de compilation et les dépendances de développement en sont exclus, ce qui réduit à la fois la taille et la surface d'attaque.

Le Dockerfile du backend emploie une première image Eclipse Temurin JDK 21 pour compiler le JAR, puis une image JRE 21 pour l'exécution. L'application est lancée avec une limite de mémoire proportionnelle à la mémoire du conteneur et expose le port 8080.

Les deux Dockerfiles frontend utilisent Node.js 22 Alpine. La première étape installe les dépendances avec pnpm et construit l'application Next.js. L'étape finale copie uniquement la sortie `standalone`, les ressources statiques et le dossier public. Le portail visiteur expose le port 3100 et le back-office le port 3101.

Deux chemins de construction coexistent, et il importe de ne pas les confondre. En **local**, les trois images se fabriquent avec les `Dockerfile` du dépôt, ce qui permet de reproduire à la main ce que fera la chaîne d'intégration. En **intégration continue**, l'image du serveur est produite par **Jib** — qui construit une image sans démon Docker, directement depuis Gradle — tandis que les deux portails sont construits par **Kaniko**, pour la même raison : éviter d'accorder aux tâches d'intégration les privilèges qu'exige un démon Docker. **Les images effectivement déployées sont celles produites par la chaîne d'intégration**, jamais celles construites en local, qui ne servent qu'à la vérification.

```
docker build -f app/Dockerfile -t gv-backend:test .
docker build -f portals/bo/Dockerfile -t gv-bo:test .
docker build -f portals/visitors/Dockerfile -t gv-visitors:test .
```

Listing 5.1 – Construction locale des trois images

Les trois composants donnent trois images distinctes, dont les tailles sont instructives : environ 760 Mo pour le serveur, qui embarque le moteur d'exécution Java, contre 385 et 391 Mo pour les portails, qui ne conservent que la sortie autonome de Next.js. Chacune peut être reconstruite et livrée indépendamment tout en appartenant à la même version fonctionnelle, identifiée par son étiquette.

5.3.2 Ce que font Kubernetes et Helm

Ces deux outils étant au cœur du déploiement, il faut dire précisément ce que chacun apporte — ils sont souvent cités ensemble alors qu'ils répondent à deux questions différentes.

Kubernetes exécute. Une image de conteneur est un paquet inerte ; quelqu'un doit décider sur quelle machine l'exécuter, la surveiller, la redémarrer si elle s'arrête, la remplacer lors d'une mise à jour et répartir le trafic entre ses exemplaires. C'est le rôle de Kubernetes. On ne lui donne pas d'ordres successifs mais une *description de l'état souhaité* — « je veux deux exemplaires du serveur, joignables à cette adresse, avec cette configuration » — et il travaille en continu à faire correspondre la réalité à cette description. Si un conteneur meurt à trois heures du matin, il en redémarre un sans intervention.

Trois notions suffisent ici. Un **pod** est l'unité d'exécution : un conteneur qui tourne. Un **déploiement** décrit combien d'exemplaires d'un pod doivent exister et comment les remplacer sans interruption. Un **service** donne à ces exemplaires une adresse stable à l'intérieur du cluster, indépendante de leur nombre et de leur emplacement.

Les sondes, ou comment Kubernetes sait qu'une application va bien. Trois vérifications sont déclarées sur le serveur applicatif. Ces trois sondes appliquent une discipline de robustesse classique [4]. La sonde de *démarrage* indique quand l'application a fini de s'initialiser — le temps d'appliquer les migrations de schéma, ce qui peut prendre plusieurs dizaines de secondes au premier lancement. La sonde de *disponibilité* indique si l'application est en état de recevoir du trafic ; tant qu'elle répond négativement, aucune requête ne lui est envoyée. La sonde de *vivacité* détecte une application bloquée, qui répond encore au réseau mais ne travaille plus, et provoque son redémarrage. Sans cette distinction, une application lente à démarrer serait tuée en boucle avant d'avoir servi une seule requête.

Helm décrit. Décrire l'état souhaité représente plusieurs centaines de lignes de fichiers de configuration, presque identiques d'un environnement à l'autre à quelques valeurs près — le nom des images, le nombre d'exemplaires, l'adresse de la base. Helm est le gestionnaire de paquets qui résout ce problème : un **chart** est un modèle paramétrable, et un fichier de *valeurs* fournit ce qui change d'un environnement à l'autre.

L'intérêt pratique est double. La recette et la production partagent *exactement* la même description, ce qui élimine la classe d'incidents où un réglage existe d'un côté et pas de l'autre. Et le chart étant versionné et publié comme une image, on peut revenir à une version antérieure du déploiement aussi simplement qu'à une version antérieure du code.

5.3.3 Le chart du projet

Le chart `deploy/helm/gv` décrit trois charges applicatives : le backend Spring Boot, le portail back-office et le portail visiteur. Le backend reste interne au cluster ; les portails relaient les appels `/api` côté serveur. PostgreSQL, le stockage S3/MinIO, le relais SMTP et les fournisseurs d'identité sont des services externes provisionnés par l'équipe d'exploitation.

La configuration non sensible passe par des ConfigMaps. Les mots de passe et clés sont lus depuis un Secret Kubernetes existant ou matérialisés depuis Vault par External Secrets Operator. Le chart refuse de produire un manifeste lorsqu'aucune source de secrets n'est définie et ne contient aucune valeur secrète en clair.

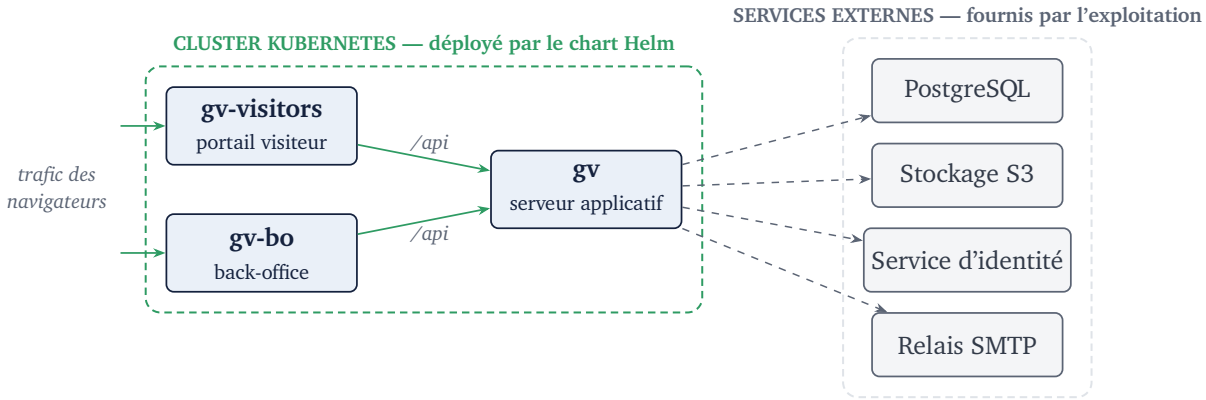


Figure 5.3 – Déploiement : trois charges dans le cluster, services d'infrastructure à l'extérieur

Le diagramme distingue ce que le projet déploie de ce qu'il consomme. À l'intérieur du cluster figurent les trois charges applicatives — le backend et les deux portails — chacune dans son conteneur, le backend n'étant joignable que depuis l'intérieur : le trafic des navigateurs entre par les portails, qui relaient les appels `/api` côté serveur. À l'extérieur figurent les services fournis par l'exploitation : la base PostgreSQL, le stockage compatible S3, le relais SMTP et le fournisseur d'identité. Cette frontière est celle de la responsabilité : le chart décrit les trois premiers composants et se contente de référencer les seconds par leurs coordonnées et leurs secrets.

Les trois sondes décrites plus haut sont déclarées sur le serveur, et un **conteneur d'initialisation** attend que PostgreSQL réponde avant que l'application ne démarre. Sans lui, le serveur tenterait d'appliquer les migrations contre une base absente, échouerait, serait redémarré par Kubernetes, échouerait de nouveau — la boucle d'échec classique d'un premier déploiement.

Le chart expose en outre, par fichier de valeurs et sans modification du modèle, les réglages qui varient d'un environnement à l'autre : les ressources processeur et mémoire allouées, le nombre d'exemplaires, leur ajustement automatique selon la charge, le nombre minimal d'exemplaires à maintenir pendant une opération de maintenance, les règles de filtrage réseau entre composants, le chiffrement des échanges et la supervision. Tous sont désactivés en recette et destinés à être activés en production, ce qui est la raison d'être de la séparation décrite ci-après.

```
helm upgrade --install gv ./deploy/helm/gv \
  -n gv-test \
  -f deploy/helm/gv/values-test.yaml
```

Listing 5.2 – Installation dans l'environnement de recette

5.3.4 Séparation entre local, recette et production

Docker Compose est réservé au développement et fournit localement les dépendances nécessaires. En recette, Helm déploie uniquement les trois composants applicatifs et référence les services gérés par l'exploitation. La production reprend le même chart avec un fichier de valeurs différent, des images épinglées par digest, des secrets issus de Vault, du TLS et les mécanismes de haute disponibilité activés. Cette séparation garantit que la même version applicative est testée puis promue sans reconstruire son contenu.

5.4 Bilan de la vérification

Table 5.3 – Bilan des tests automatisés

Suite	Nombre	Résultat
Tests unitaires serveur	1 054, sur 133 classes	Aucun échec
dont règles d'architecture ArchUnit : 13, toutes respectées		
Tests d'intégration (Testcontainers)	157, sur 28 classes	Aucun échec
Tests portail back-office (Vitest)	660, sur 59 fichiers	Aucun échec
Tests portail visiteur (Vitest)	236, sur 13 fichiers	Aucun échec
Parcours de bout en bout (Playwright)	4	Aucun échec
Total	**2 111**	**Aucun échec**

Comment ce total est obtenu. Le total additionne les quatre suites exécutables et les parcours de bout en bout : $1\,054 + 157 + 660 + 236 + 4 = 2\,111$ vérifications automatisées. Les 13 règles d'architecture ne forment pas une ligne supplémentaire : elles s'exécutent comme des tests JUnit et sont donc déjà comprises dans les 1 054 tests unitaires serveur. Aucun test n'est compté deux fois.

Ces chiffres proviennent de l'exécution du 30 août 2026 sur l'environnement local, en repartant d'un cache de tests vidé afin qu'aucune suite ne soit déclarée conforme sans avoir été réellement rejouée. La distinction entre les deux lignes serveur est importante : les 1 054 premiers tests s'exécutent sans infrastructure, tandis que les 157 tests d'intégration démarrent une véritable base PostgreSQL dans un conteneur éphémère et vérifient donc le comportement réel des requêtes, des migrations et du cloisonnement par site.

Les commandes correspondantes sont exécutées en repartant d'un cache de tests vidé. Cette précaution n'est pas cosmétique : Gradle réutilise le résultat d'une suite dont rien n'a changé et affiche alors un succès quasi instantané, qui donne l'illusion d'une exécution. Vider le cache au préalable est la seule façon de garantir que les 1 054 tests ont bien été rejoués.

```
./gradlew cleanTest :app:test test    # 1054 tests dont 13 regles ArchUnit
./gradlew integrationTest             # 157 tests sur PostgreSQL conteneurise
pnpm -C portals/bo test               # 660 tests
pnpm -C portals/visitors test         # 236 tests
```

Listing 5.3 – Commandes de vérification exécutées

5.4.1 Scénarios de sécurité vérifiés

La quantité de tests ne suffit pas à qualifier la sécurité. Les scénarios suivants ciblent donc les invariants qui auraient l'impact le plus élevé en cas de régression : refus d'une transition d'état illégale, lecture et écriture hors périmètre site, absence du marqueur de bonne réponse dans le quiz public, expiration et réutilisation d'un OTP, blocage après plusieurs échecs, limitation des demandes, contrôle des permissions et génération du certificat uniquement après réussite. Le parcours de livraison a ajouté une vérification navigateur réelle : OTP, pré-déclaration, induction, émission du passeport, accès VIP, quiz 2/2 et génération des deux QR.

Cette vérification démontre le fonctionnement de l'application dans l'environnement local. Elle ne vaut pas test d'intrusion : l'homologation reste conditionnée aux contrôles d'infrastructure associés au tableau 3.3.

Conclusion

Les mécanismes de vérification décrits permettent de détecter la majorité des régressions avant leur mise en service. Ils ne remplacent toutefois pas la vérification manuelle des parcours dans un navigateur, qui a été effectuée lors des démonstrations hebdomadaires et de la stabilisation finale. Les suites automatisées, les constructions et le chart de déploiement étaient conformes au moment de la livraison.

Conclusion générale et perspectives

Bilan du travail réalisé

Ce projet est parti d'un constat simple : le registre papier tenu en guérite ne permet ni de savoir qui se trouve sur un site, ni de garantir que les personnes présentes ont été formées aux consignes de sécurité. Il aboutit à une plateforme couvrant le cycle de vie complet d'une visite, de la pré-déclaration à distance jusqu'à la sortie du site.

Sur le plan fonctionnel, la solution livrée permet au visiteur de se déclarer à distance ou sur place par code QR, de suivre une induction sécurité évaluée et sanctionnée par un certificat vérifiable, et de se présenter en guérite avec un passeport numérique. Elle permet à l'agent de sécurité d'enregistrer les entrées et les sorties, de consulter à tout instant la liste des présents, et de déclencher une procédure d'évacuation. Elle outille le parcours protocolaire, avec sa validation hiérarchique et son induction dédiée. Elle offre enfin à l'administration la maîtrise des sites, des comptes, des habilitations et de la configuration, ainsi que la traçabilité et la restitution statistique.

L'ampleur du périmètre couvert se mesure. La plateforme livrée compte **douze modules** — sept métier et cinq techniques — pour **deux portails web** distincts, une quarantaine de tables et soixante-trois migrations de schéma appliquées dans l'ordre. **2 111 tests automatisés** s'exécutent sans un seul échec, dont 157 contre une véritable base de données, et treize règles d'architecture refusent la compilation d'un code qui enfreindrait les frontières entre modules. La chaîne de livraison est complète : **trois images conteneurisées**, un pipeline d'intégration continue en quatre étapes et un chart de déploiement Kubernetes. Vingt-trois décisions d'architecture consignent les choix et leurs raisons. L'ensemble a été conçu, développé, testé et déployé par une seule personne en huit semaines.

Table 5.4 – Le projet en chiffres

Périmètre fonctionnel	Réalisation technique
9 epics, 56 récits utilisateurs	12 modules applicatifs
7 profils d'acteurs	2 portails web · 1 API
7 cimenteries et le siège	∼40 tables · 63 migrations
2 parcours de visite distincts	2 111 tests · 0 échec
2 langues sur le portail public	13 règles d'architecture
7 diagrammes de cas d'utilisation	23 décisions d'architecture

Sur le plan technique, trois décisions structurent le résultat. Le **monolithe modulaire** concilie la simplicité d'exploitation d'un déploiement unique et la discipline de séparation d'un système distribué, les frontières entre modules étant vérifiées à la compilation. La **machine à états** rend le cycle de vie d'une visite explicite et vérifiable, là où des indicateurs indépendants auraient autorisé des états incohérents. Le **cloisonnement au niveau de la persistance** garantit qu'un site ne voit pas les données d'un autre, sans dépendre de la vigilance de chaque requête.

Apports personnels

Ce stage m'a permis de conduire de manière autonome un projet complet, depuis l'analyse des besoins transmis par l'encadrant jusqu'au déploiement en recette. J'ai développé le backend, les deux portails, les migrations, les tests, les images Docker, le pipeline et le chart Helm. Cette responsabilité de bout en bout m'a appris à raisonner en termes de produit et non uniquement en termes de code.

Sur le plan technique, j'ai consolidé ma maîtrise de Java, Spring Boot, TypeScript, React et Next.js. J'ai découvert les exigences d'une architecture modulaire, la conception d'autorisations multi-sites, la sécurisation d'un parcours public et l'automatisation de la qualité. Docker, Kubernetes, Helm et GitLab CI m'ont aussi permis de comprendre le passage d'une application fonctionnant sur un poste à une solution reproductible et déployable.

Enfin, ce projet m'a fait pratiquer une manière de développer qui est en train de s'installer dans le métier, et dont je retiens trois enseignements.

Le premier est que la valeur se déplace vers la **formulation et la vérification**. Un agent produit en quelques minutes ce qui prenait une journée, mais il produit du code

plausible : savoir décrire précisément l'attendu, puis démontrer que le résultat est juste, devient la compétence déterminante. J'ai appris à ne jamais accepter un compte rendu sans avoir lu la différence de code et relancé les tests moi-même — plusieurs fois, ce réflexe a révélé qu'une fonctionnalité annoncée terminée ne l'était pas.

Le deuxième est qu'un tel outil **amplifie la qualité de l'ingénierie qui l'entoure**, dans les deux sens. Sur ce dépôt, les conventions écrites, les frontières vérifiées à la compilation et les tests nombreux transforment la vitesse en avancement réel. Sur un projet dépourvu de ces garde-fous, la même vitesse aurait surtout accéléré l'accumulation de dette technique. Concevoir le dispositif de contrôle fait donc partie du travail, au même titre que le code.

Le troisième est que la **responsabilité ne se délègue pas**. L'architecture, l'interprétation du besoin, les arbitrages et la validation fonctionnelle sont restés de mon fait. C'est cette répartition — l'outil pour la production, l'ingénieur pour les décisions et la preuve — qui m'a permis de livrer seul, en huit semaines, un périmètre qui aurait autrement demandé une équipe.

Les difficultés rencontrées

Quatre obstacles ont demandé plus qu'une mise en œuvre : ils ont imposé de revenir sur une décision.

Deux parcours qui se ressemblent sans être les mêmes. La visite ordinaire et la visite protocolaire partagent le vocabulaire — un visiteur, une induction, une entrée, une sortie — et la tentation de les servir avec les mêmes tables était forte. Le cahier des charges l'interdit : validation hiérarchique, questionnaire noté et certificat n'existent que du côté protocolaire, et les mutualiser aurait fait porter à chaque écran ordinaire des conditions qui ne le concernent pas. La séparation en deux modules distincts coûte de la duplication apparente ; elle achète la certitude qu'une évolution d'un parcours ne peut pas casser l'autre, et la frontière est vérifiée à la compilation plutôt que laissée à la discipline.

Ne pas répondre à une question qu'on n'avait pas le droit de poser. Le cloisonnement par site paraissait résolu dès lors que chaque requête filtrait sur le périmètre de l'utilisateur. Il restait une fuite discrète : répondre « accès interdit » sur une visite d'un autre site révèle que cette visite existe. En interrogeant des identifiants au hasard, un agent aurait pu cartographier l'activité des autres sites sans jamais lire une donnée. La règle retenue — répondre « introuvable » — va à l'encontre de la formulation littérale de plusieurs récits utilisateurs, ce qui a exigé de le documenter explicitement pour que le choix ne soit pas pris plus tard pour une erreur.

Certifier ce qui a réellement été vu. Une case à cocher « j'ai pris connaissance des consignes » n'a aucune valeur si rien n'atteste la consultation. Le verrou de durée minimale a donc été posé côté serveur autant que côté écran, et la correction du questionnaire protocolaire s'effectue exclusivement sur le serveur : la projection des réponses envoyée au navigateur retire le marqueur de bonne réponse. Sans cette précaution, il suffirait d'ouvrir les outils de développement du navigateur pour lire les réponses attendues, et le certificat ne prouverait plus rien.

Vérifier plutôt que croire. La difficulté la plus constante n'a pas été technique. Un agent rend compte de ce qu'il a eu l'intention de faire, et ce compte rendu est convaincant même lorsqu'il est faux. Plusieurs fois, une fonctionnalité annoncée terminée ne l'était pas, ou l'était autrement. Le réflexe qui s'est imposé — lire la différence de code, relancer les tests soi-même, ouvrir l'écran — a coûté du temps mais évité d'inscrire dans le rapport des affirmations que la démonstration aurait démenties.

Ce que la plateforme change pour Holcim

Le bénéfice se mesure d'abord en sécurité. En cas d'évacuation, la question « qui se trouve actuellement sur le site » recevait jusqu'ici une réponse reconstituée à partir d'un registre papier, à la main, dans l'urgence. Elle reçoit désormais une réponse immédiate, nominative et imprimable, transmise simultanément aux responsables sûreté. Aucune autre fonction du produit n'a ce poids.

Le deuxième apport est la traçabilité. Chaque autorisation d'entrée, chaque refus, chaque modification de droits laisse une trace horodatée et attribuée, consultable et exportable. Un registre papier ne dit pas qui a laissé entrer qui, ni sur quel fondement ; le journal d'audit le dit, et la question devient vérifiable a posteriori.

Le troisième est le temps rendu à l'agent de sécurité. La pré-déclaration déporte la saisie vers le visiteur, qui la fait chez lui et à son rythme. Au portail, il ne reste qu'un code à scanner et un verdict à lire — ce qui, aux heures d'affluence, déplace le poste de garde d'une activité de recopie vers une activité de contrôle.

Reste une condition, et elle n'est pas technique : la valeur du dispositif tient à la qualité des consignes de sécurité administrées par site. Le logiciel garantit qu'elles ont été présentées et certifiées ; il ne garantit pas qu'elles soient à jour. C'est un travail métier, que l'écran d'administration rend possible mais ne remplace pas.

Perspectives d’évolution

La base livrée se prête à des prolongements qui en étendent la portée sans en remettre en cause la structure. Cinq se dégagent naturellement.

1. **Synchronisation avec l’annuaire d’entreprise.** La reprise automatique du référentiel des employés éviterait la saisie manuelle des hôtes.
2. **Intégration au contrôle d’accès physique.** Le rapprochement avec les tourniquets permettrait de rendre l’ouverture effective conditionnelle à l’autorisation logique.
3. **Application mobile pour l’agent.** Une application native apporterait le fonctionnement hors ligne, utile lorsque la couverture réseau est défaillante en guérite.
4. **Reconnaissance automatique des pièces d’identité.** La lecture optique réduirait le temps de saisie à l’arrivée.
5. **Supervision et alerte.** L’ajout d’une remontée centralisée des erreurs et de seuils d’alerte faciliterait l’exploitation.

Bibliographie et webographie

[1] Evans, E. *Domain-Driven Design : Tackling Complexity in the Heart of Software*. Addison-Wesley, 2003.

[2] Fowler, M. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

[3] Newman, S. *Building Microservices : Designing Fine-Grained Systems*. 2e éd., O'Reilly Media, 2021.

[4] Nygard, M. T. *Release It ! Design and Deploy Production-Ready Software*. 2e éd., Pragmatic Bookshelf, 2018.

[5] Royaume du Maroc. *Loi n° 09-08 relative à la protection des personnes physiques à l'égard du traitement des données à caractère personnel*. Bulletin officiel, 2009.

[6] Commission nationale de contrôle de la protection des données à caractère personnel (CNDP). *Formalités*. https://www.cndp.ma/formalites/

[7] Parlement européen et Conseil. *Règlement (UE) 2016/679 relatif à la protection des personnes physiques à l'égard du traitement des données à caractère personnel* (RGPD), 2016.

[8] Nottingham, M., Wilde, E. et Dalal, S. *RFC 9457 — Problem Details for HTTP APIs*. IETF, juillet 2023. https://www.rfc-editor.org/rfc/rfc9457.html

[9] Jones, M., Bradley, J. et Sakimura, N. *RFC 7519 — JSON Web Token (JWT)*. IETF, mai 2015.

[10] Hardt, D. *RFC 6749 — The OAuth 2.0 Authorization Framework*. IETF, octobre 2012.

[11] Grassi, P. *et al. NIST SP 800-63B — Digital Identity Guidelines : Authentication and Lifecycle Management*. NIST. https://pages.nist.gov/800-63-4/sp800-63b.html

[12] OWASP. *Application Security Verification Standard* (ASVS). https://owasp.org/www-project-application-security-verification-standard/

[13] OWASP. *REST Security Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html

[14] W3C. *Web Content Accessibility Guidelines (WCAG) 2.2*. Recommandation, octobre 2023. https://www.w3.org/TR/WCAG22/

[15] The PostgreSQL Global Development Group. *Row Security Policies*, documentation PostgreSQL 16. https://www.postgresql.org/docs/current/ddl-rowsecurity.html

[16] VMware. *Spring Modulith Reference Documentation — Verification*. https://docs.spring.io/spring-modulith/reference/verification.html

[17] VMware. *Spring Boot Reference Documentation*. https://docs.spring.io/spring-boot/

[18] Vercel. *Next.js Documentation*. https://nextjs.org/docs

[19] OpenAPI Initiative. *OpenAPI Specification*, version 3.1. https://spec.openapis.org/oas/latest.html

[20] The Helm Authors. *Helm Documentation*. https://helm.sh/docs/

[21] The Kubernetes Authors. *Kubernetes Documentation*. https://kubernetes.io/docs/home/

[22] Anthropic. *Claude Code Documentation*. https://code.claude.com/docs/en/overview

[23] Holcim Maroc. *Site institutionnel — présentation du groupe, activités et actionnariat*. Consulté en août 2026. https://www.holcim.ma/

[24] Holcim Maroc. *Nos activités — outil industriel : cimenteries, centres de broyage, centrales à béton*. Consulté en août 2026. https://www.holcim.ma/nos-activites

[25] Holcim Maroc. *La nouvelle cimenterie Agadir-Souss est opérationnelle*. Communiqué, consulté en août 2026. https://www.holcim.ma/la-nouvelle-cimenterie-agadir-souss-est-operationnelle

ANNEXE A

Annexes

A.1 Extrait du contrat d'API

```yaml
paths:
  /api/v1/visites:
    post:
      summary: Enregistrer une pre-declaration de visite
      operationId: creerVisite
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/CreerVisiteRequest'
      responses:
        '201':
          description: Visite creee
        '400':
          description: Requete invalide
          content:
            application/problem+json: {}
        '404':
          description: Ressource introuvable ou hors perimetre
```

Listing A.1 – Extrait de la spécification OpenAPI

A.2 Format des erreurs

Toutes les erreurs renvoyées par l'API suivent le format normalisé RFC 9457 [8], afin que les portails n'aient qu'une seule forme à traiter. Le champ `correlation_id` permet de retrouver la trace correspondante dans les journaux du serveur.

```json
{
  "type": "https://gv.holcim.ma/errors/visite/visite-introuvable",
  "title": "Not Found",
  "status": 404,
  "detail": "visite.errors.visiteIntrouvable",
```

```
  "correlation_id": "01923e4c-7f21-7a3b-9e55-2c1d4f8a6b03"
}
```

Listing A.2 – Exemple de réponse d'erreur

Une précision sur le champ `detail`. L'implémentation y place la clé de traduction plutôt qu'une phrase. Ce choix est délibéré : le serveur ignore la langue de l'utilisateur, alors que le portail dispose d'un catalogue par locale et traduit à la frontière d'affichage. La RFC 9457 destine `detail` à une explication lisible par un humain ; l'écart est donc assumé et documenté, plutôt que laissé implicite. Il est sans conséquence pour les clients, toutes les erreurs transitant par un point unique qui garantit la forme du corps de réponse.

A.3 Exemple de migration de schéma

```
CREATE TABLE visite (
    id              UUID PRIMARY KEY,
    site_id         UUID NOT NULL REFERENCES ref_site(id),
    statut          VARCHAR(32) NOT NULL,
    date_visite     DATE NOT NULL,
    motif_id        UUID REFERENCES ref_motif_visite(id),
    created_at      TIMESTAMPTZ NOT NULL,
    updated_at      TIMESTAMPTZ NOT NULL
);

CREATE INDEX idx_visite_site_statut ON visite (site_id, statut);
```

Listing A.3 – Migration créant la table des visites

A.4 Une règle d'architecture exécutable

Le corps de ce rapport affirme que treize règles refusent la compilation d'un code qui enfreindrait les frontières décidées. En voici une, reproduite telle qu'elle s'exécute. Elle porte sur le cloisonnement par site : chaque table doit déclarer explicitement si ses lignes appartiennent à un site ou sont globales.

```
@ArchTest
static final ArchRule entities_are_tenant_scoped_xor_tenant_neutral =
    classes()
        .that()
        .areAnnotatedWith(Entity.class)
        .should(beTenantScopedXorTenantNeutral())
        .because(
```

```
8               "Site scope is a per-entity decision that must be taken "
9             + "explicitly (ADR-0001). An unannotated @Entity silently "
10            + "behaves as global.");
```

Listing A.4 – Règle imposant une décision explicite de portée à chaque entité

L'intérêt n'est pas la règle elle-même mais ce qu'elle rend impossible. Une table oubliée ne *ressemble* pas à une table globale : elle *devient* globale, silencieusement, et la fuite ne se voit ni en relecture ni dans un test métier qui n'exercerait qu'un seul site. La règle transforme cet oubli en échec de compilation, avec le message ci-dessus et le nom de la classe fautive. La décision est ainsi prise une fois, au moment où la table est introduite, par la personne qui connaît la réponse.

Une règle jumelle traite le seul angle mort de ce mécanisme : les filtres de Hibernate ne réécrivent pas le SQL natif. Toute requête native visant une table cloisonnée doit donc porter un marqueur de dérogation motivé, faute de quoi la compilation échoue également.

A.5 Extrait de la chaîne d'intégration continue

Le pipeline s'exécute à chaque envoi de code, en quatre étapes successives — `test`, `build`, `image`, `package`. Une étape ne démarre que si la précédente est entièrement verte, si bien qu'aucune image ne peut être publiée à partir d'un code dont les tests échouent.

```
stages: [test, build, image, package]

backend:test:
  stage: test
  script:
    - ./gradlew :app:test      # règles d'architecture et frontières
    - ./gradlew test           # tests unitaires de tous les modules
  artifacts:
    reports:
      junit: app/build/test-results/test/TEST-*.xml

backend:integration-test:
  stage: test
  services:
    - name: docker:27-dind     # base PostgreSQL réelle, jetable
  script:
    - ./gradlew integrationTest
```

Listing A.5 – Étape de test du pipeline GitLab CI

Le second travail mérite un mot : il démarre une véritable base PostgreSQL dans un conteneur jetable, le temps de la vérification. Les migrations, les requêtes et le cloisonnement par site sont donc éprouvés contre le moteur réel, et non contre une base en mémoire dont le comportement diffère précisément là où les erreurs se logent.

A.6 Glossaire métier

Terme	Définition
Guérite	Poste de contrôle situé à l'entrée d'un site, tenu par un agent de sécurité
Induction	Formation obligatoire aux consignes de sécurité, préalable à l'accès au site
Passeport visiteur	Justificatif numérique porteur d'un code QR, remis au visiteur après sa déclaration
Périmètre	Ensemble des sites auxquels un utilisateur a accès
Pré-déclaration	Déclaration d'une visite effectuée à distance, avant l'arrivée sur site
Rattrapage	Régularisation d'une visite dont l'entrée n'a pas été saisie sur le moment
Visite spontanée	Visite déclarée sur place, sans annonce préalable
Visite VIP	Visite protocolaire soumise à validation hiérarchique