Ecole d'ingénierie Digital et
d'Intelligence Artificielle (EIDIA)
Cybersecurity Engineering Cycle (Ac2023)

Développement Web Avancé

Système Intelligent de Gestion et Notification des Absences

Réalisé par :
Amr Ouafi
Reda Lahssaini
Akram Loutouat
Sara Tazi

Enseignant :
Pr. Ahmed Amamou

Année académique 2025–2026

Table des matières

Chapitre 1

Introduction

1.1 Contexte et Analyse de l'Existant

La gestion administrative des absences constitue une problématique organisationnelle majeure au sein des établissements universitaires. Actuellement, ce processus repose essentiellement sur des flux de données fragmentés et non standardisés.

La difficulté principale réside dans **l'hétérogénéité des sources d'information**. Les services de scolarité doivent agréger des données provenant de canaux variés tels que les exports du système Massar, les relevés enseignants ou les logs biométriques.

Cette absence d'uniformité structurelle (formats de fichiers, encodages et nomenclatures disparates) contraint l'administration à effectuer un traitement manuel systématique. Cette charge de travail, nécessaire à l'harmonisation des données, engendre inévitablement des risques d'erreurs et des latences dans la communication avec les parents.

1.2 Équipe et Méthodologie de Travail

1.2.1 Composition de l'Équipe

La réussite de ce projet repose sur une répartition stratégique des tâches, exploitant les compétences complémentaires des quatre membres de l'équipe :

— **Amr (Backend Lead & Sécurité) :** Responsable de l'architecture de sécurité et de l'authentification. Ses missions incluent l'implémentation du cycle de connexion (Login/Logout), la gestion des permissions via Middleware (`auth_guard`), la sécurisation des sessions et la protection systématique des API contre les injections SQL via des requêtes préparées.

— **Akram (Backend Core Features) :** En charge du cœur fonctionnel de l'application. Il a développé le module CRUD pour les étudiants et les absences, le parser CSV sécurisé (vérification stricte des types MIME et du contenu), le moteur de notifications (Email/WhatsApp) et le système de journalisation pour l'audit.

— **Reda (Frontend & Visualisation de Data) :** Focalisé sur l'interactivité et la visualisation des données. Il a conçu les tableaux de bord dynamiques (filtrage, pagination), implémenté la validation des formulaires côté client et intégré les graphiques statistiques pour le suivi des absences.

— **Sara (Frontend UI & Tests) :** Responsable de l'expérience utilisateur et de la qualité. Elle a réalisé l'interface d'importation, le dashboard simplifié, assuré

l'harmonisation du design CSS et mené les campagnes de tests manuels pour valider les flux critiques.

1.2.2 Stratégie de Collaboration

Pour mener à bien ce développement, nous avons adopté une méthodologie agile adaptée à nos contraintes :

— **Gestion de Version (GitHub) :** Utilisation d'une stratégie à deux dépôts. Un dépôt pour le développement local permettant les itérations rapides, et un dépôt "Prod" synchronisé avec la Machine Virtuelle pour le déploiement.

— **Synchronisation :** Des réunions présentielles régulières ont permis de résoudre les blocages techniques et d'aligner l'intégration Backend/Frontend.

1.3 Vision du Produit et Solution

Notre réponse à cette problématique est une plateforme web centralisée, conçue pour **automatiser l'intégralité du flux de gestion**.

Le cœur de l'innovation réside dans notre **Algorithme d'Ingestion Intelligent**. Contrairement à un importateur classique qui échoue si une colonne change de nom, notre système utilise une analyse sémantique (basée sur la distance de Levenshtein) pour "comprendre" le fichier entrant et mapper automatiquement les colonnes (ex : associer "Last Name", "Nom de Famille" ou "Student" au champ `nom_etudiant` de la base).

1.4 Périmètre Fonctionnel

Le projet s'articule autour de trois piliers techniques :

1. **Ingestion Intelligente :** Support de fichiers CSV arbitraires avec détection automatique du délimiteur et mapping assisté par l'algorithme de levenshtein.
2. **Traitement Optimisé :** Stockage des absences avec une stratégie de partitionnement mensuel (*Sharding*) pour garantir la performance des requêtes SQL.
3. **Diffusion Multicanale :** Système de notification hybride capable d'envoyer des alertes via Email (SMTP) et WhatsApp (API Twilio Sandbox) lorsque des seuils d'absences sont atteints.

1.5 Identification des Acteurs

L'application gère trois niveaux d'accès distincts :

— **L'Administrateur :** Super-utilisateur ayant accès à la configuration système (SMTP, API Keys), la liste des utilisateurs que ça soit côté parent ou côté équipe administratif et aux logs d'audit.

— **L'Opérateur :** Gestionnaire quotidien responsable de l'import des fichiers et de la validation des justifications.

— **Le Parent :** Utilisateur externe accédant à un portail sécurisé simplifié via un "Lien Magique" qui lui permet de rentrer son numéro de carte d'identité comme une couche additionnelle de sécurité et ensuite il créer son mot de passe pour faciliter l'accès la prochaine fois qu'il souhaite consulter la situation de son enfant.

Chapitre 2

Architecture Cloud et Infrastructure

Ce chapitre détaille l'environnement technique mis en place pour assurer l'hébergement et le déploiement de l'application. Pour dépasser les limitations d'un environnement de développement local (`localhost`) et répondre à l'exigence d'avoir un serveur web accessible publiquement, nous avons migré vers une infrastructure Cloud professionnelle.

2.1 Transition vers un Environnement de Production

2.1.1 Le Besoin d'Externalisation

Le développement initial sur machine locale présente une limite majeure : l'inaccessibilité depuis l'extérieur. Or, pour tester les fonctionnalités critiques comme les notifications mobiles (WhatsApp) ou l'accès au portail par les parents depuis leurs smartphones, il était impératif de disposer d'un serveur exposé sur Internet avec une adresse IP publique.

2.1.2 Le Choix de Google Cloud Platform (GCP)

Nous avons retenu **Google Cloud Platform (Compute Engine)** pour héberger notre solution. Ce choix est motivé par la nécessité de simuler un environnement de production réel, où nous sommes responsables de la configuration complète du serveur (OS, Réseau, Sécurité).

2.1.3 Stratégie d'Optimisation des Coûts

Une contrainte forte du projet était l'absence de budget opérationnel. Nous avons donc adopté une stratégie d'optimisation stricte en exploitant le "Free Tier" de Google. Cela nous a permis de bénéficier d'une instance de calcul gratuite, nous obligeant en contrepartie à optimiser notre code pour qu'il tourne sur des ressources limitées.

2.2 Dimensionnement et Système d'Exploitation

2.2.1 Analyse de l'Instance (Sizing)

L'infrastructure repose sur une machine virtuelle (VM) de type `e2-micro`. Les spécifications techniques, extraites directement du serveur, sont les suivantes :

— **Processeur :** Intel(R) Xeon(R) CPU @ 2.20GHz (2 vCPU).

— **Mémoire Vive (RAM) :** 1 Go (969 MiB disponibles).
— **Stockage :** 30 Go (Standard Persistent Disk).

2.2.2 Le Choix de Debian 12

Contrairement à la distribution Ubuntu Server souvent utilisée par défaut, nous avons opté pour **Debian 12 (Bookworm)**. Ce choix est purement technique : avec seulement 1 Go de RAM disponible, chaque mégaoctet compte. Debian est réputé pour sa légèreté et sa stabilité.

Les mesures effectuées sur le serveur montrent que l'OS et les services (Apache, PHP, MySQL) consomment déjà près de 64% de la mémoire disponible (621 MiB utilisés). L'utilisation d'un système plus lourd aurait pu compromettre la stabilité de l'application lors des pics de charge.

2.3 Architecture Réseau et Accessibilité

Pour rendre l'application accessible aux utilisateurs finaux (Administration et Parents), nous avons mis en place une architecture réseau spécifique.

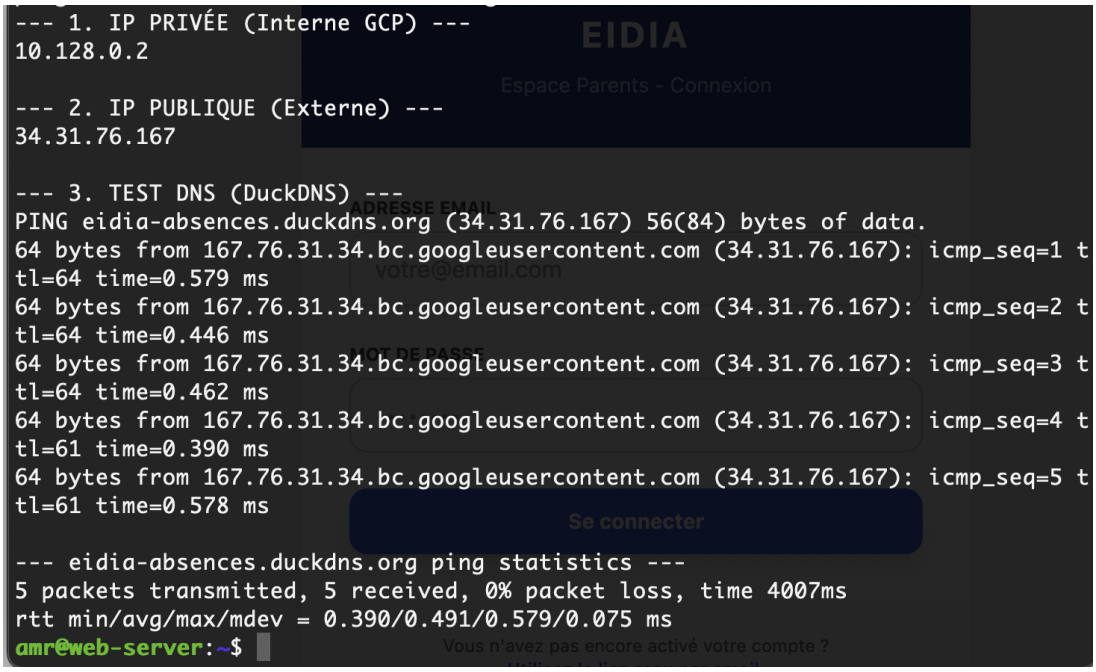

FIGURE 2.1 – Schéma de l'architecture réseau et flux de connexion

2.3.1 Gestion du DNS (DuckDNS)

Les adresses IP publiques des instances Cloud gratuites sont souvent dynamiques (elles changent au redémarrage). Pour pallier ce problème sans payer une IP statique, nous avons utilisé le service **DuckDNS**. Il permet d'associer un nom de domaine fixe (ex : `mon-projet.duckdns.org`) à l'IP de notre machine, garantissant un accès permanent.

2.3.2 Configuration du Serveur Web (Apache)

Le serveur web Apache a été configuré manuellement via les *VirtualHosts*. Cette configuration est essentielle pour la sécurité : elle dirige toutes les requêtes web vers le dossier `public/`, rendant le code source (Dossiers `src/`, `config/`) inaccessible depuis un navigateur.

2.4 Pipeline d'Intégration Continue (CI/CD)

Dans une optique d'industrialisation du projet, nous avons abandonné les transferts de fichiers manuels (FTP) au profit d'un déploiement automatisé.

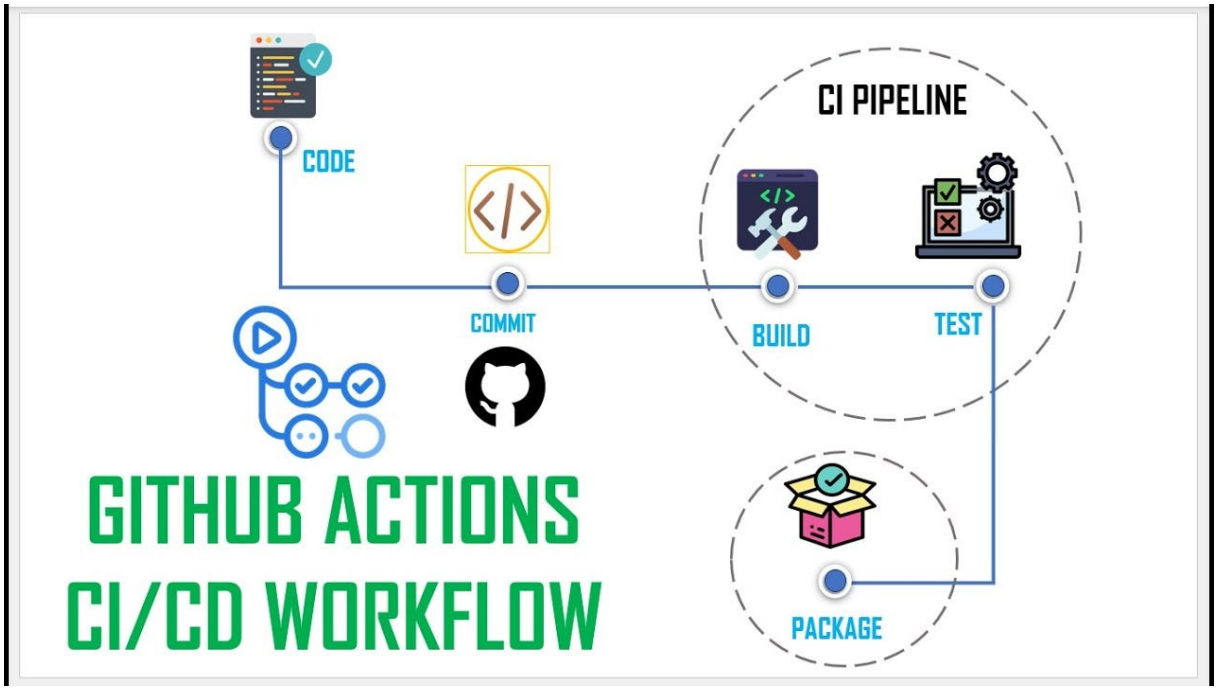

FIGURE 2.2 – Workflow de déploiement automatisé via GitHub Actions

2.4.1 Pourquoi l'automatisation ?

Le déploiement manuel est source d'erreurs (oubli de fichier, problème de permissions). L'intégration continue (CI/CD) permet de fiabiliser ce processus.

2.4.2 Implémentation avec GitHub Actions

Nous avons configuré un workflow GitHub qui s'active automatiquement à chaque modification (push) sur la branche principale (`main`). Le script effectue les actions suivantes :

1. Connexion sécurisée à la VM Google Cloud via SSH.
2. Récupération de la dernière version du code (`git pull`).
3. Ajustement automatique des permissions des fichiers pour le serveur web.

Cela garantit que la version en ligne est toujours synchronisée avec notre dépôt de code, sans intervention humaine risquée.

2.4.3 Gestion du DNS Dynamique (DDNS)

L'attribution d'un nom de domaine est indispensable pour le fonctionnement des certificats SSL et l'accessibilité utilisateur. Cependant, notre instance GCP ne dispose d'une IP publique statique ce qui facilite l'hébergement.

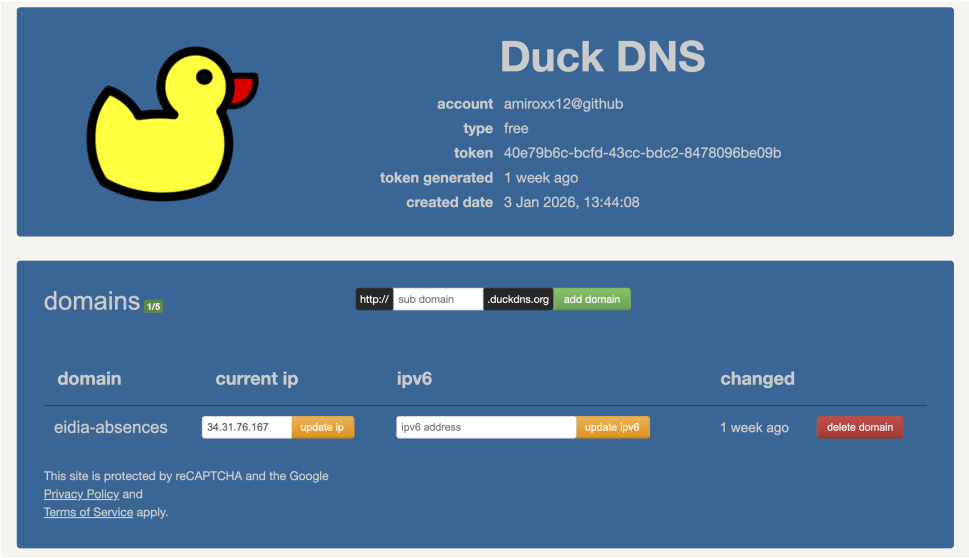

FIGURE 2.3 – Interface DuckDNS montrant l'association de l'IP Publique au domaine

Le Choix de DuckDNS vs BIND9

Pour résoudre ce problème de résolution de nom, deux stratégies s'offraient à nous :

1. **Configurer un serveur DNS autoritaire (BIND9) :** Cette solution académique consiste à héberger notre propre zone DNS. Bien que puissante, elle présente deux inconvénients majeurs pour notre contexte :
   — **Problème de la "Glue Record" :** Pour qu'un serveur DNS soit accessible, il doit lui-même avoir une IP fixe enregistrée chez le registrar, ce qui nous ramène au problème de coût initial.
2. **Utiliser un service de DNS Dynamique (DuckDNS) :** Nous avons opté pour cette solution pragmatique et surtout gratuite.

2.5 Sécurisation des Flux (HTTPS)

La manipulation de données sensibles (noms d'étudiants, numéros de téléphone des parents) impose un chiffrement strict des communications. L'utilisation du protocole HTTP standard (port 80) exposerait ces données à des attaques de type *Man-in-the-Middle* (MITM).

2.5.1 Implémentation SSL/TLS avec Let's Encrypt

Nous avons sécurisé le serveur web Apache en implémentant le protocole HTTPS via l'autorité de certification gratuite **Let's Encrypt**.

L'installation a été réalisée à l'aide de l'outil **Certbot**, qui a automatiquement :

1. Généré les clés cryptographiques (Publique/Privée).

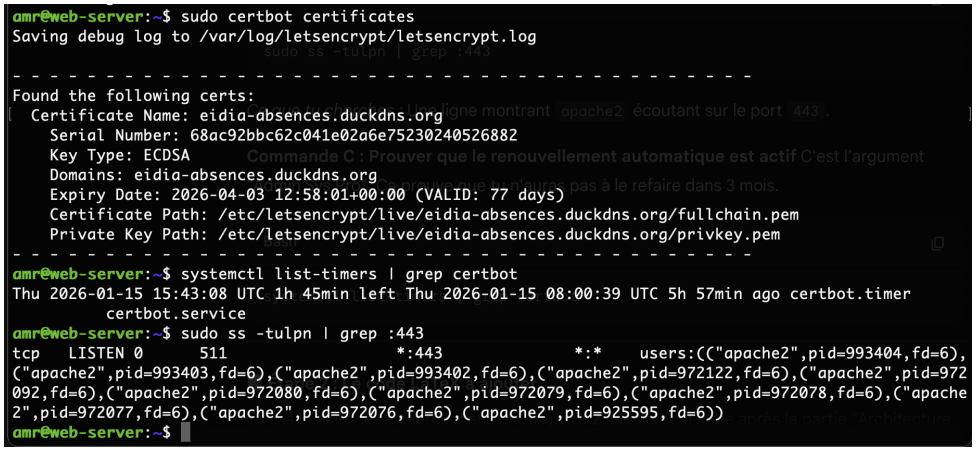

FIGURE 2.4 – État du certificat SSL Let's Encrypt sur le serveur (Preuve de validité)

2. Validé la propriété du domaine `duckdns.org` via un challenge ACME.
3. Configuré le VirtualHost Apache pour forcer la redirection du trafic HTTP (Port 80) vers HTTPS (Port 443).

Chapitre 3

Conception et Modélisation

La phase de conception est cruciale pour garantir la robustesse et l'évolutivité de l'application. Avant d'écrire la moindre ligne de code, nous avons structuré le système en suivant les standards UML (Unified Modeling Language) et en adoptant une architecture de base de données optimisée pour le volume de données attendu.

3.1 Analyse Fonctionnelle (Diagramme de Cas d'Utilisation)

L'analyse des besoins nous a permis d'identifier trois acteurs principaux interagissant avec le système, avec une hiérarchie de privilèges stricte.

3.1.1 Identification des Acteurs

— **L'Administrateur :** Il possède tous les droits système. C'est le seul acteur capable de modifier la configuration critique (Serveur SMTP, API Twilio), de consulter les journaux d'audit (Logs) et de gérer les comptes utilisateurs.

— **L'Opérateur :** C'est l'utilisateur "métier" quotidien. Il est responsable de l'importation des fichiers CSV, de la validation du mapping des colonnes et de l'envoi des campagnes de notification.

— **Le Parent :** Acteur externe au réseau administratif. Il accède au système de manière restreinte via un lien sécurisé pour consulter l'historique de son enfant et soumettre des justificatifs.

3.2 Architecture de Données et Optimisation (MLD)

La modélisation de la base de données ne s'est pas limitée à une simple traduction des besoins en tables. Nous avons intégré une stratégie d'optimisation avancée : le **Partitionnement Horizontal (Sharding)**.

3.2.1 Stratégie de Partitionnement Temporel (Sharding)

Une table unique `absences` contenant l'historique de plusieurs années deviendrait rapidement un goulot d'étranglement (plusieurs millions de lignes), ralentissant considérablement les requêtes du Dashboard.

Nous avons opté pour une création dynamique des tables par mois :

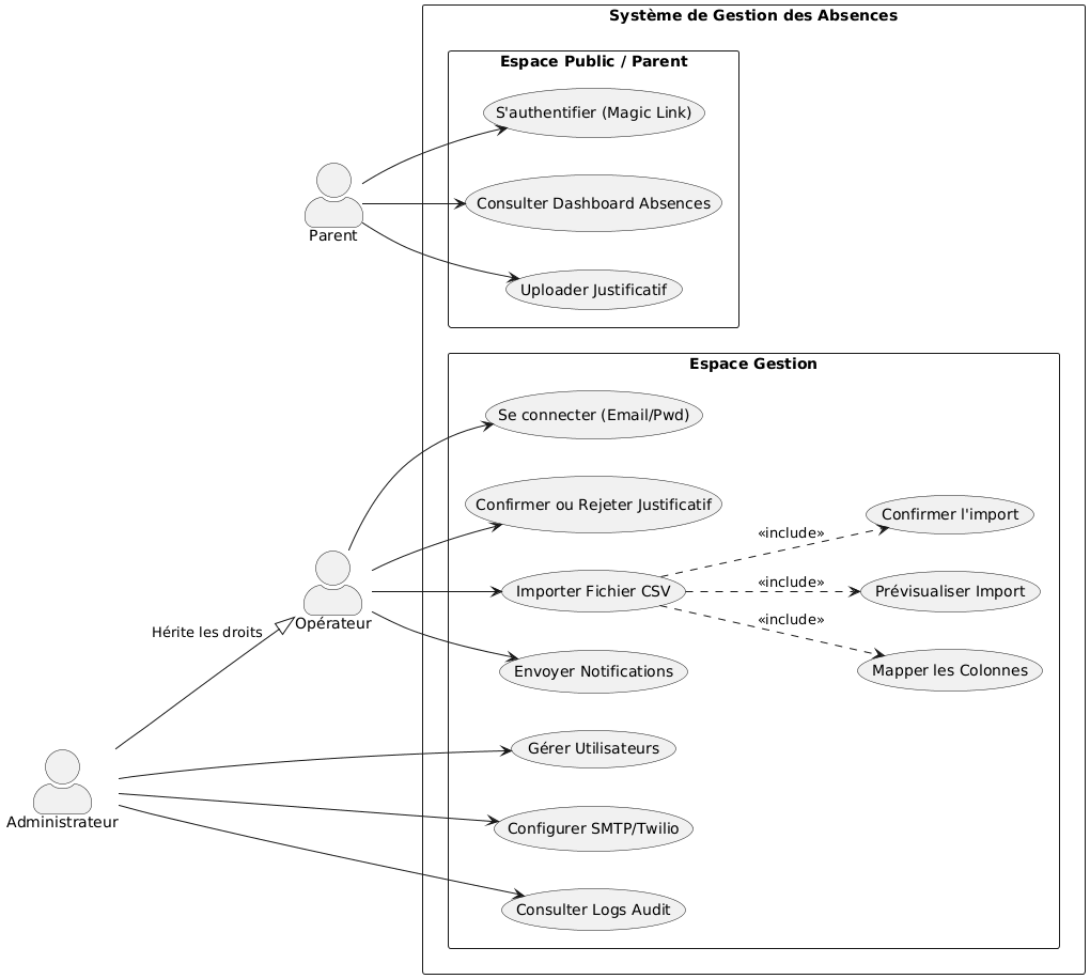

FIGURE 3.1 – Diagramme de Cas d'Utilisation Global

— `absences_01_2025`

— `absences_02_2025`

— ...

Avantages techniques :

1. **Performance de Lecture :** Le Dashboard du mois en cours ne scanne que la petite table du mois (quelques milliers de lignes) au lieu de l'historique complet.
2. **Archivage Facilité :** Il est possible de "détacher" ou d'archiver une vieille table mensuelle sans impacter le reste de la base de données.

3.2.2 Entités de Sécurité

Le modèle intègre également des tables dédiées à la sécurité :

— `parent_tokens` **:** Stocke les jetons d'accès temporaires (Magic Links). Chaque jeton est lié à une date d'expiration stricte (`expires_at`).

— `import_configurations` **:** Permet de sauvegarder les modèles de mapping JSON pour que l'IA n'ait pas à ré-analyser des fichiers connus.

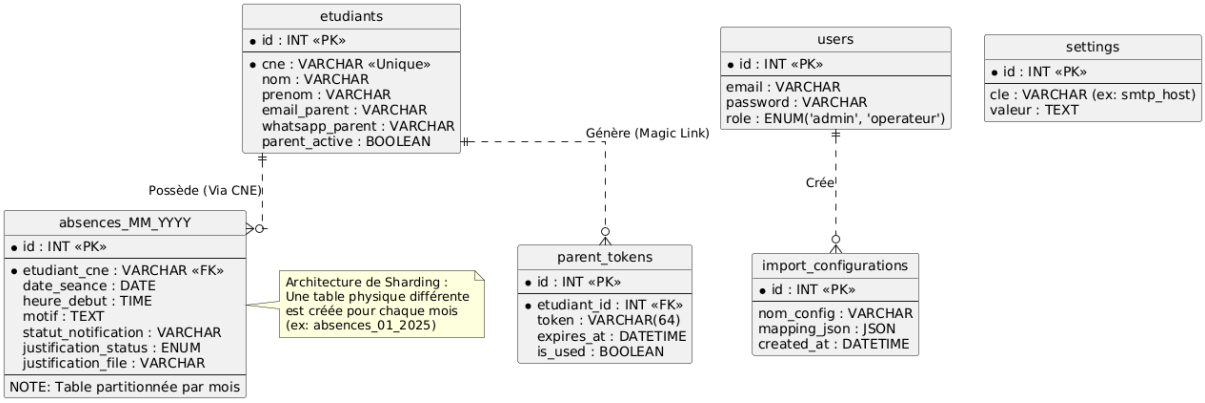

FIGURE 3.2 – Modèle Logique de Données (MLD) mettant en évidence le partitionnement mensuel

3.3 Conception Logicielle (Architecture MVC)

L'application repose sur le patron de conception **MVC (Modèle-Vue-Contrôleur)**, renforcé par une couche de **Services** pour isoler la logique métier complexe.

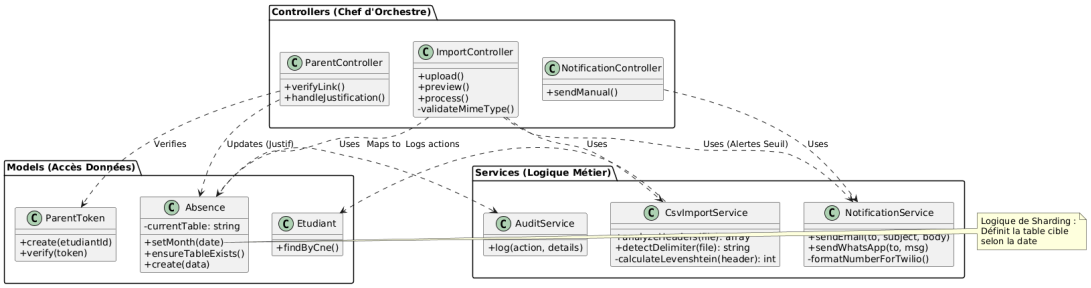

FIGURE 3.3 – Diagramme de Classes simplifié (Backend Architecture)

3.3.1 Séparation des Responsabilités

Le diagramme de classes illustre le découplage fort entre les composants :

- **Les Contrôleurs (ex : `ImportController`) :** Ils agissent comme chefs d'orchestre. Ils reçoivent la requête HTTP, valident les entrées (MIME types), mais ne contiennent aucune logique métier complexe.
- **Les Services (ex : `CsvImportService`) :** C'est le "cerveau" de l'application.
  - `CsvImportService` contient l'algorithme de distance de Levenshtein.
  - `NotificationService` encapsule la complexité des API externes (SMTP/Twilio).
- **Les Modèles (ex : `Absence`) :** Ils sont responsables de la persistance des données. C'est la classe `Absence` qui contient la logique intelligente permettant de router l'écriture vers la bonne table mensuelle (`absences_MM_YYYY`) selon la date de l'absence.

Chapitre 4

Implémentation Backend et Logique Métier

Ce chapitre constitue le cœur technique du rapport. Il détaille l'implémentation des mécanismes critiques qui assurent la sécurité, la performance et l'intelligence de l'application. L'architecture backend respecte strictement le modèle MVC (Modèle-Vue-Contrôleur), découplant ainsi la logique de présentation de la logique métier.

4.1 L'Orchestration Centrale (Le Routeur)

4.1.1 Concept : Le Front Controller

Contrairement aux architectures web anciennes où chaque fichier PHP correspondait à une URL (ex : `login.php`, `dashboard.php`), nous avons adopté l'approche du **Front Controller**. Le fichier `public/index.php` agit comme un point d'entrée unique pour l'intégralité du trafic HTTP. Cette centralisation offre deux avantages majeurs :

- **Initialisation globale :** Les sessions, l'autoloader et les configurations de sécurité sont chargés une seule fois avant tout traitement.
- **Sécurité par l'obscurité :** La structure réelle des dossiers (`src/Controllers/...`) est masquée à l'utilisateur final.

4.1.2 Mécanisme de Routage

Le routeur analyse l'URI de la requête et la compare à une table de correspondance (`$routes`). Si une route est trouvée, le contrôleur associé est instancié dynamiquement.

```
// 1. Analyse et nettoyage de l'URL
$requestUri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);

// 2. Table de routage (Map : URL -> [Classe, Méthode])
$routes = [
    '/' => ['App\Controllers\ParentController', 'showLogin'],
    '/import/process' => ['App\Controllers\ImportController', 'process'],
    '/absences/notify' => ['App\Controllers\NotificationController', 'sendManual'],
];

// 3. Dispatching dynamique
if (array_key_exists($path, $routes)) {
```

```
    $controllerName = $routes[$path][0];
    $methodName = $routes[$path][1];

    // Appel dynamique : new Controller()->method()
    (new $controllerName())->$methodName();
} else {
    // 4. Sécurité Fail-safe : Redirection silencieuse vers l'accueil
    // Emp che le "Path Disclosure" via les erreurs 404 par défaut
    header('Location: ' . BASE_URL . '/');
    exit;
}
```

Listing 4.1 – Logique de dispatching dans index.php

4.2 Authentification et Gestion des Accès

La gestion des identités est scindée en deux modules distincts pour répondre aux besoins spécifiques des administrateurs (sécurité maximale) et des parents (facilité d'accès).

4.2.1 Authentification Standard et ACL (Access Control List)

Pour le personnel administratif, l'authentification repose sur le service `AuthService`.

— **Vérification :** Les mots de passe sont hachés via l'algorithme `Bcrypt` (fonction `password_verify`).

— **Middleware Logique :** Au lieu d'utiliser un middleware externe complexe, chaque contrôleur critique intègre une vérification de rôle au début de son exécution. Si l'utilisateur n'a pas le rôle `admin` ou `operateur`, l'exécution est stoppée immédiatement et l'utilisateur est redirigé.

4.2.2 Innovation : Authentification "Magic Links" (ParentToken)

Pour les parents, souvent moins à l'aise avec la gestion de comptes multiples, nous avons implémenté une authentification sans mot de passe initial ("Passwordless").

Cycle de vie du Jeton :

1. Le système génère un jeton cryptographique à haute entropie.
2. Ce jeton est stocké en base avec une date d'expiration stricte (+24h).
3. Un lien unique est envoyé par email. Au clic, le système vérifie la validité du jeton et connecte le parent.

L'implémentation technique utilise `random_bytes`, un générateur de nombres pseudo-aléatoires cryptographiquement sécurisé (CSPRNG), rendant les jetons impossibles à deviner par force brute.

```
public function create(int $etudiantId, string $emailParent): string {
    // Génération de 32 octets aléatoires convertis en 64 caractères Hex
    // random_bytes est "Cryptographically Secure" contrairement à rand
    ()
    $token = bin2hex(random_bytes(32));

    // Définition de la fen tre de validité
    $expiresAt = date('Y-m-d H:i:s', strtotime('+24 hours'));

```

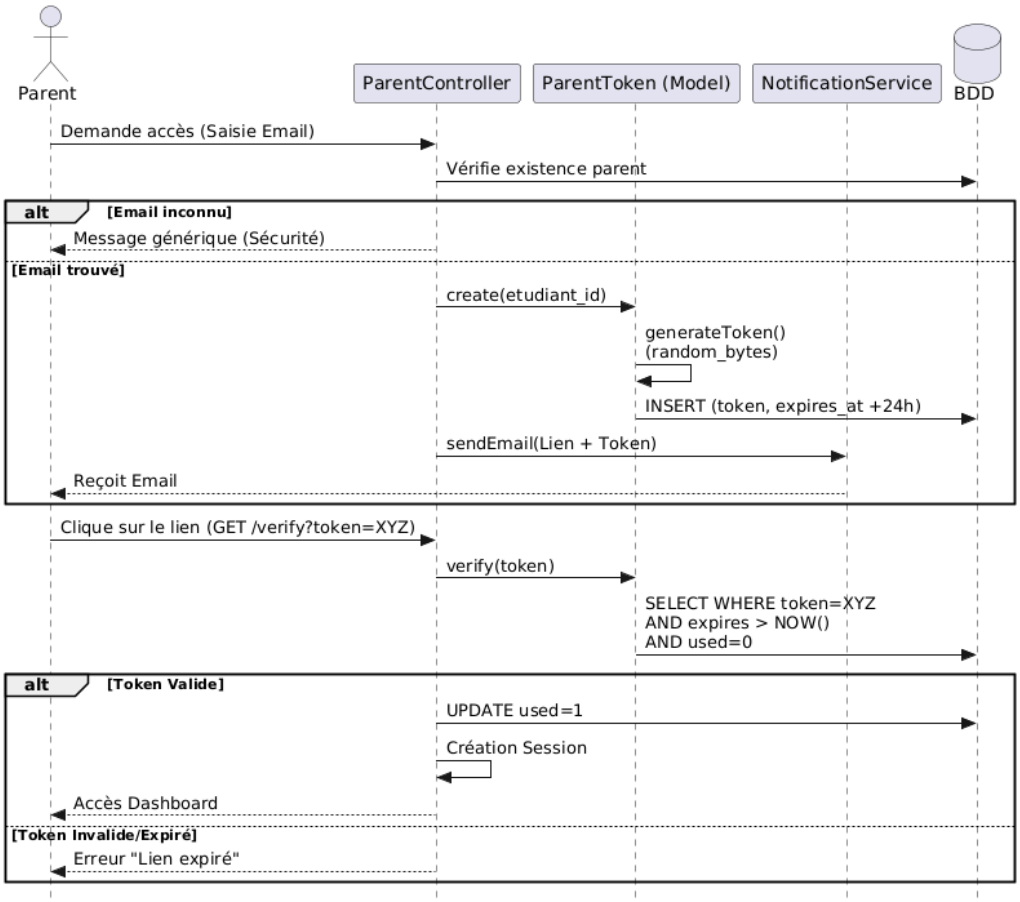

FIGURE 4.1 – Flux d'authentification par Magic Link sécurisé

```
9       // Persistance du token
10      $sql = "INSERT INTO parent_tokens (etudiant_id, token, expires_at)
        VALUES ...";
11      // ...
12      return $token;
13  }
```

Listing 4.2 – Génération cryptographique (ParentToken.php)

4.3 Gestion Avancée des Données (Les Modèles)

La couche Modèle ne se contente pas de refléter la base de données ; elle intègre des stratégies d'optimisation pour palier les ressources limitées du serveur Cloud (1 Go RAM).

4.3.1 Optimisation via le Pattern Singleton

L'ouverture d'une connexion TCP vers MySQL est une opération coûteuse en temps et en mémoire. Pour éviter d'ouvrir une nouvelle connexion à chaque fois qu'un Modèle est instancié (ce qui saturerait rapidement le serveur), nous avons implémenté le **Design Pattern Singleton** dans `DatabaseService`. Cela garantit qu'une unique instance de connexion `PDO` est partagée pour toute la durée de la requête HTTP.

```
1  public static function getInstance() {
2      if (self::$instance === null) {
```

```
        self::$instance = new self(); // Unique instanciation
    }
    return self::$instance;
}
```

Listing 4.3 – Implémentation du Singleton BDD

4.3.2 Architecture de Sharding Temporel

Le modèle `Absence.php` intègre une logique de routage dynamique des données. Au lieu d'écrire dans une table monolithique `absences`, le système dirige les écritures vers des tables mensuelles (`absences_01_2025`, `absences_02_2025`). Cette approche, appelée **Sharding Horizontal**, permet de maintenir des performances de lecture constantes (O(1)) pour le Dashboard, quelle que soit l'ancienneté du système.

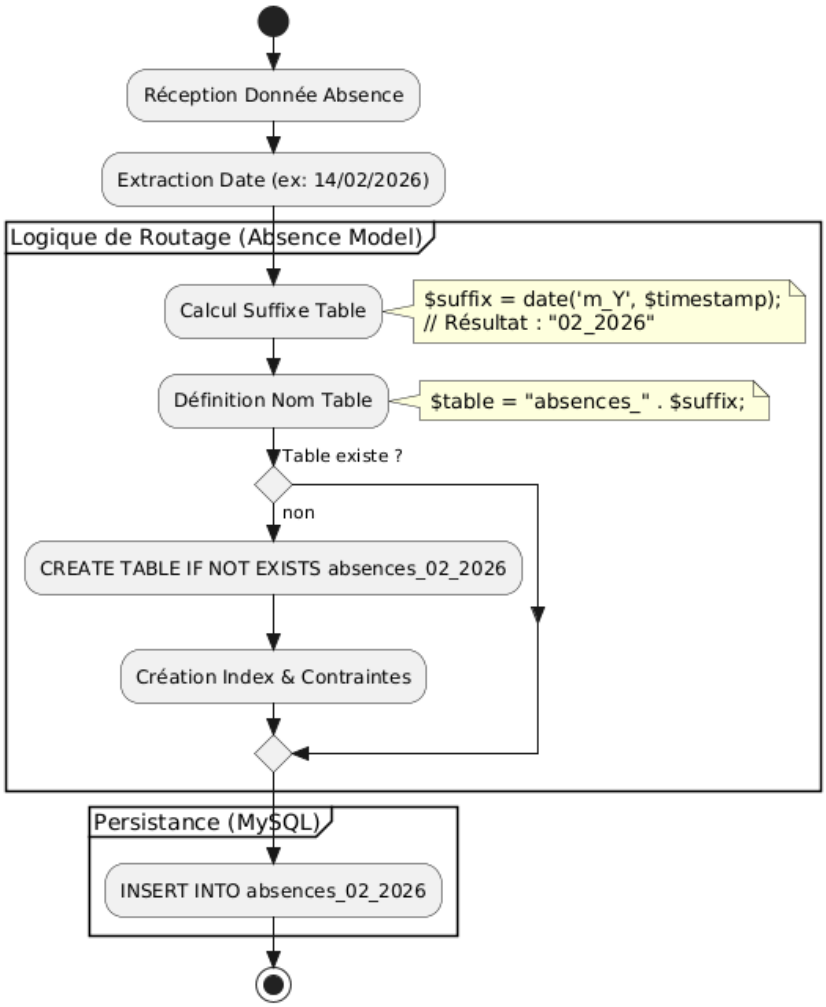

FIGURE 4.2 – Logique de routage dynamique vers les tables partitionnées

```
public function setMonth($dateString) {
    // La table cible change selon la date de l'absence
    $timestamp = strtotime(str_replace('/', '-', $dateString));
    $this->currentTable = "absences_" . date('m_Y', $timestamp);
}

```

```php
public function ensureTableExists() {
    // Création "Just-in-Time" de la table si c'est le 1er du mois
    $sql = "CREATE TABLE IF NOT EXISTS `" . $this->currentTable . "` (...)";
    $this->conn->exec($sql);
}
```

Listing 4.4 – Routage dynamique (Absence.php)

4.4 Moteur d'Importation : Sécurité et Algorithmique

Le module d'import CSV est le composant le plus complexe du système. Il doit concilier une flexibilité maximale (accepter n'importe quel fichier) avec une sécurité absolue.

4.4.1 Sécurité

L'upload de fichiers est le vecteur d'attaque n°1 sur les applications web. Dans `ImportController`, nous avons mis en place une défense en profondeur à trois niveaux ("Defense in Depth") :

1. **Contrôle de surface :** Vérification de l'extension du fichier.
2. **Signature binaire :** Vérification du *MIME Type* réel via `finfo_file` (pour rejeter un exécutable renommé en .csv).
3. **Inspection du contenu :** Scan intégral du fichier texte pour détecter des chaînes malveillantes comme `<?php` ou `system(`, prévenant ainsi les attaques de type Web Shell.

4.4.2 Algorithme de Mapping Sémantique (Fuzzy Matching)

Pour rendre le système tolérant aux variations de noms de colonnes, nous avons développé un algorithme heuristique dans `CsvImportService`. Il utilise la **Distance de Levenshtein** pour calculer un score de similarité entre l'en-tête du fichier CSV et notre dictionnaire interne.

Si le score dépasse un seuil de confiance (75%), le mapping est automatique. Sinon, une intervention manuelle est demandée.

```php
foreach ($variantes as $variante) {
    // Calcul de la distance d'édition (Levenshtein)
    $dist = levenshtein($headerNorm, $variante);

    // Normalisation en pourcentage de similarité
    $len = max(strlen($headerNorm), strlen($variante));
    $sim = ($len > 0) ? (1 - $dist / $len) * 100 : 0;

    // Seuil de tolérance fixé à 75%
    if ($sim > 75 && $sim > $bestScore) {
        $bestMatch = $champDb;
        $bestScore = round($sim);
    }
}
```

Listing 4.5 – Algorithme de matching (CsvImportService.php)

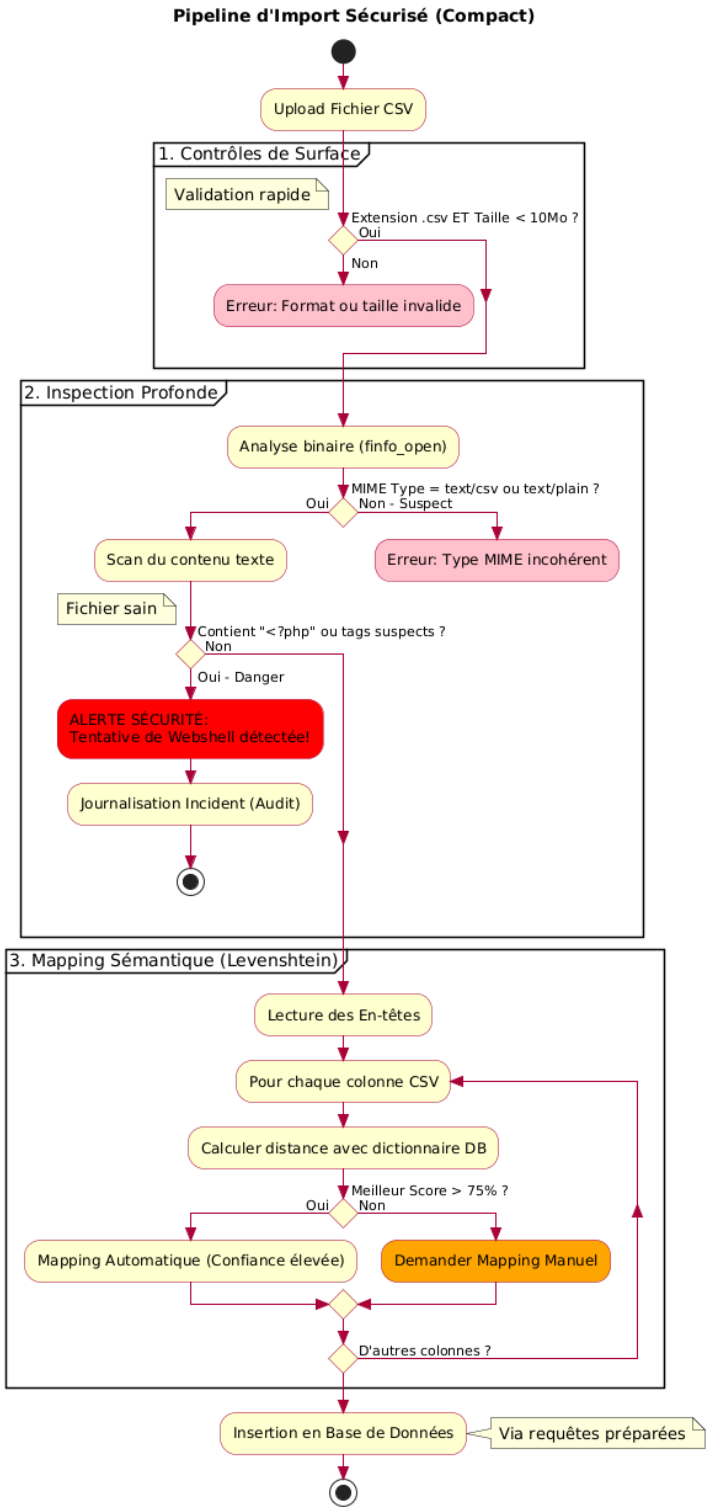

FIGURE 4.3 – Pipeline de validation sécurisée des fichiers

4.5 Communication et Traçabilité

4.5.1 Système de Notification Abstrait

Le `NotificationService` implémente le pattern **Façade**. Il masque la complexité technique des fournisseurs tiers au reste de l'application.

— **Canal Email :** Utilisation de la librairie `PHPMailer` connectée à un serveur SMTP authentifié TLS.

— **Canal WhatsApp :** Intégration de l'API REST Twilio. Le service inclut une logique de normalisation des numéros de téléphone (conversion automatique du format local `06...` vers le format international E.164 `+212...`) indispensable pour le routage SMS.

4.5.2 Audit et Journalisation (Logging)

Dans une optique de sécurité et de débogage, la traçabilité est assurée par l'`AuditService`. Contrairement à des logs fichiers volatils, nous enregistrons les événements critiques (Connexions, Imports échoués, Alertes envoyées) en base de données.

— **Pourquoi logger ?** Pour permettre une analyse post-incident (Forensics) et identifier les tentatives d'intrusion ou les erreurs récurrentes d'importation.

— **Données tracées :** Timestamp, ID Utilisateur, Type d'action, et Détails techniques (ex : "Import étudiant : 50 succès, 2 doublons").

Chapitre 5

Développement Frontend et Expérience Utilisateur

L'interface utilisateur (UI) a été conçue comme le point de convergence entre la complexité technique du Backend et les besoins opérationnels des utilisateurs. Nous avons adopté une approche centrée sur l'utilisateur, en distinguant strictement l'environnement de travail dense de l'administration (Desktop) et l'environnement de consultation épuré des parents (Mobile).

5.1 Interfaces d'Accès et Authentification

L'expérience utilisateur commence dès la page de connexion. Nous avons différencié les points d'entrée pour sécuriser les accès et adapter le parcours.

5.1.1 Portail Administratif

L'accès pour les Administrateurs et Opérateurs se fait via un formulaire standardisé. Le design est volontairement sobre pour inspirer confiance et professionnalisme.

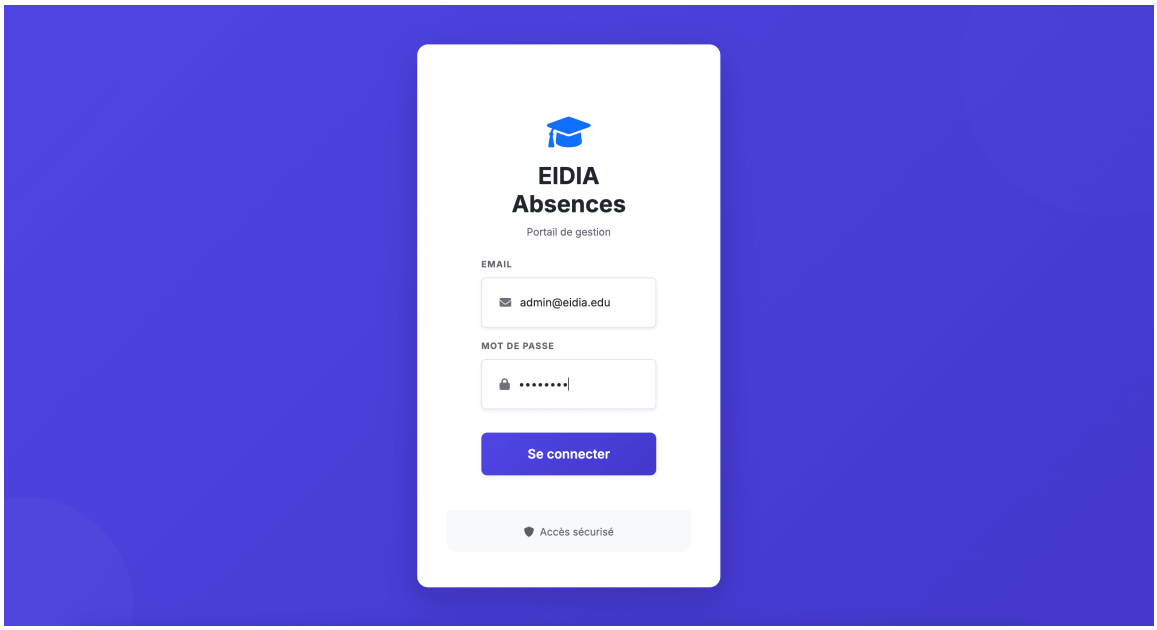

FIGURE 5.1 – Interface de connexion Administration (Email / Mot de passe)

5.1.2 Portail Parents

Pour les parents, l'interface de connexion a été simplifiée pour réduire la friction. Elle est conçue pour être responsive et s'afficher parfaitement sur les smartphones.

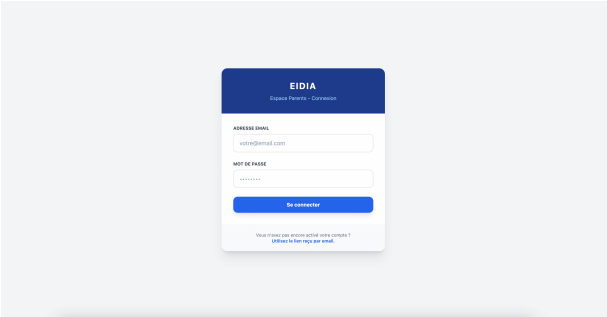

FIGURE 5.2 – Interface de connexion Parents (Mobile First)

5.2 Ergonomie des Tableaux de Bord (Dashboards)

Une fois connecté, l'utilisateur est dirigé vers un tableau de bord adapté à son rôle.

5.2.1 Dashboard Administrateur : Le Cockpit

L'administrateur dispose d'une vue d'ensemble sur le système. Son dashboard agrège des indicateurs clés de performance (KPIs) : nombre total d'étudiants, volume d'absences du mois, et état des services.

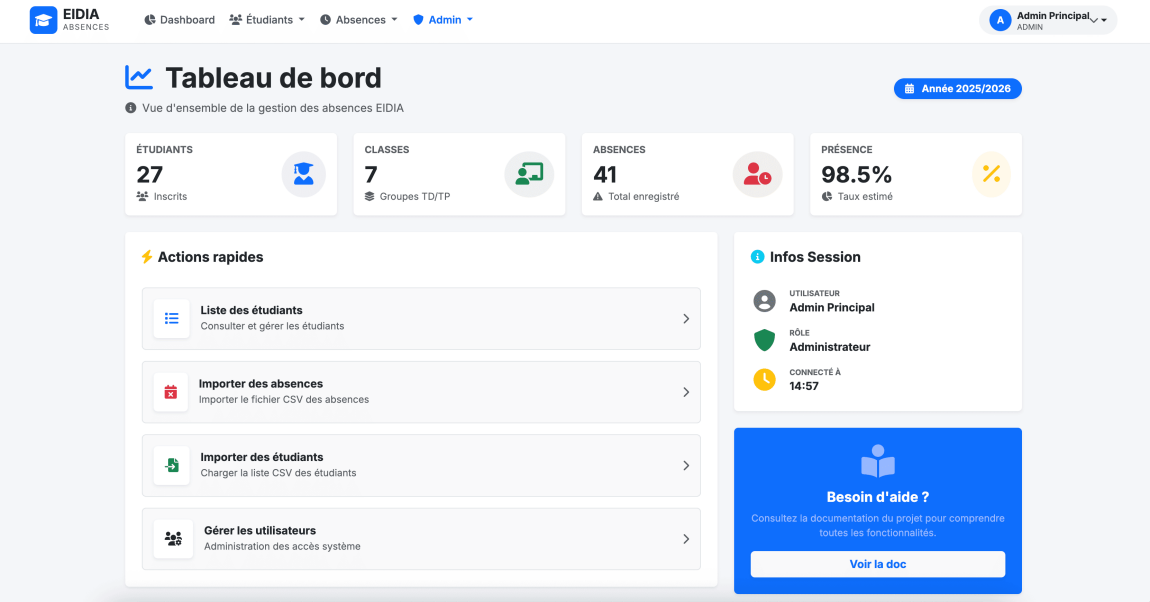

FIGURE 5.3 – Dashboard Administrateur : Vue synoptique et statistiques

5.2.2 Dashboard Parent : L'Information Immédiate

À l'opposé, le parent souhaite une réponse immédiate. Le design utilise des "Cartes" (*Cards*) avec des codes couleurs sémantiques (Vert = Présent, Rouge = Absent) et une typographie large pour une lisibilité maximale sur petit écran.

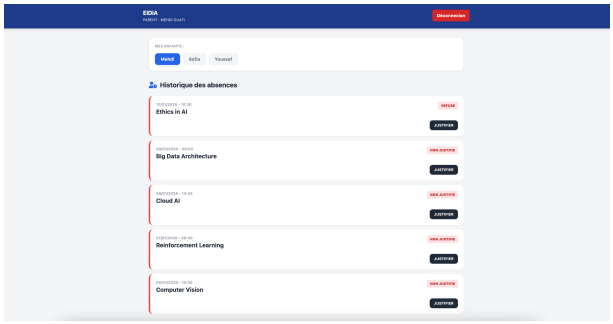

FIGURE 5.4 – Dashboard Parent : Vue mobile simplifiée par enfant

5.3 Le Workflow d'Importation (Wizard UI)

L'importation des fichiers CSV est une tâche complexe sujette aux erreurs. Pour guider l'opérateur, nous avons découpé ce processus en un assistant visuel ("Wizard") en trois étapes linéaires.

5.3.1 Étape 1 : Téléversement (Upload)

La première interface permet le glisser-déposer du fichier. Elle fournit un feedback immédiat sur la validité du format (MIME type) avant même l'envoi au serveur.

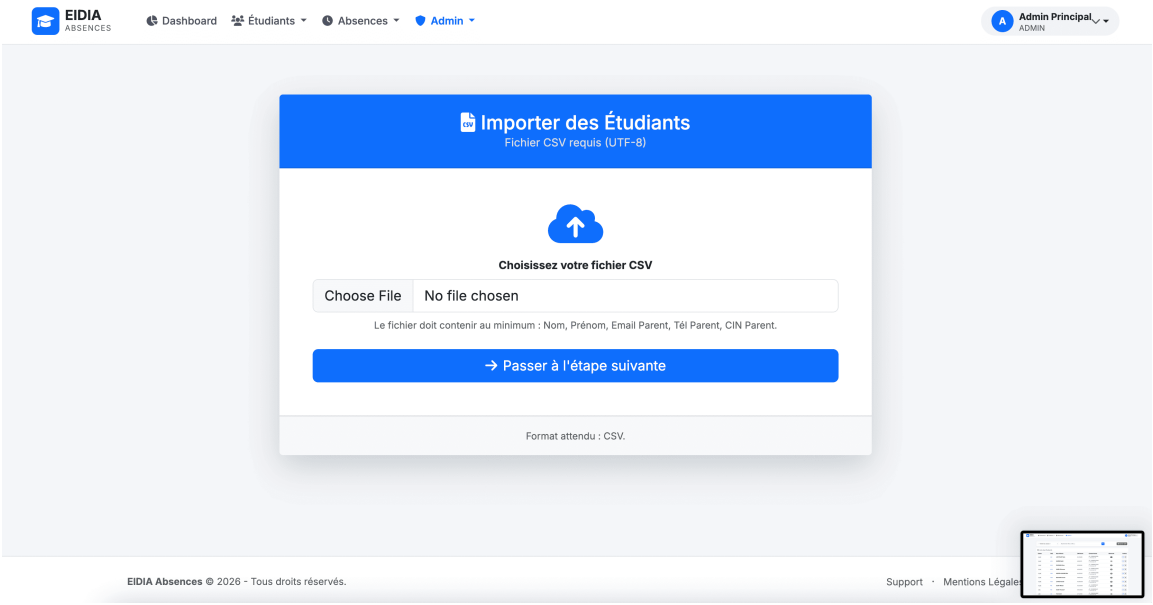

FIGURE 5.5 – Étape 1 : Zone de dépôt sécurisée et contrôle de format

5.3.2 Étape 2 : Mapping Assisté

C'est l'interface la plus sophistiquée. Elle affiche les colonnes détectées dans le CSV et propose automatiquement les correspondances avec la base de données.

5.3.3 Étape 3 : Validation (Preview)

Le principe "Trust but Verify" est appliqué ici. Un tableau présente les données telles qu'elles seront insérées, permettant de détecter les décalages de colonnes ou les erreurs d'encodage (UTF-8).

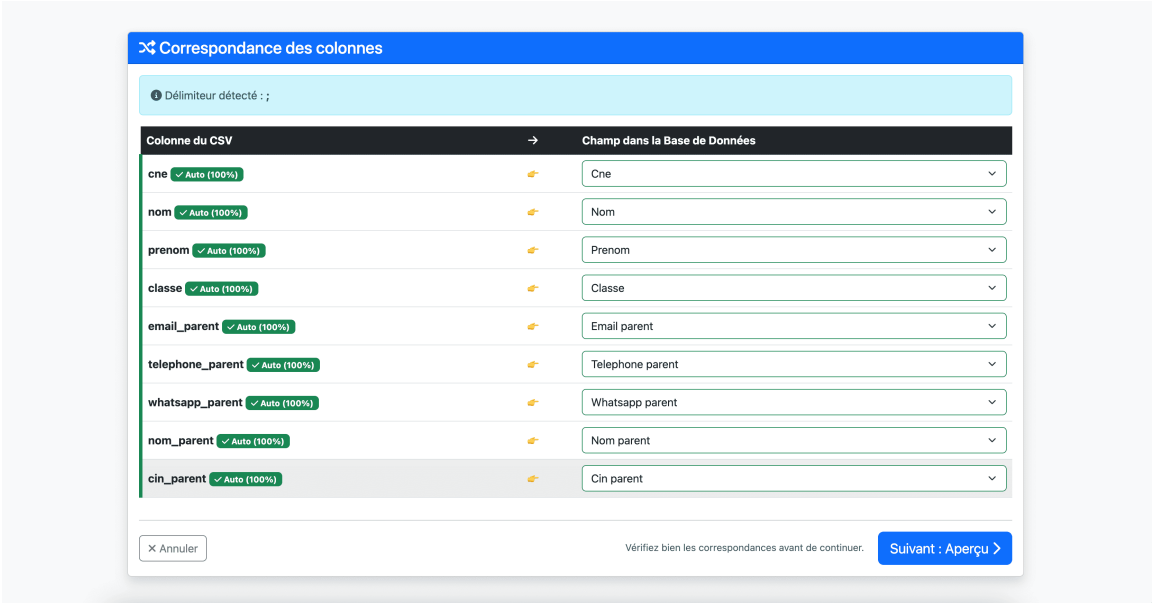

FIGURE 5.6 – Étape 2 : Interface de Mapping avec suggestions intelligentes

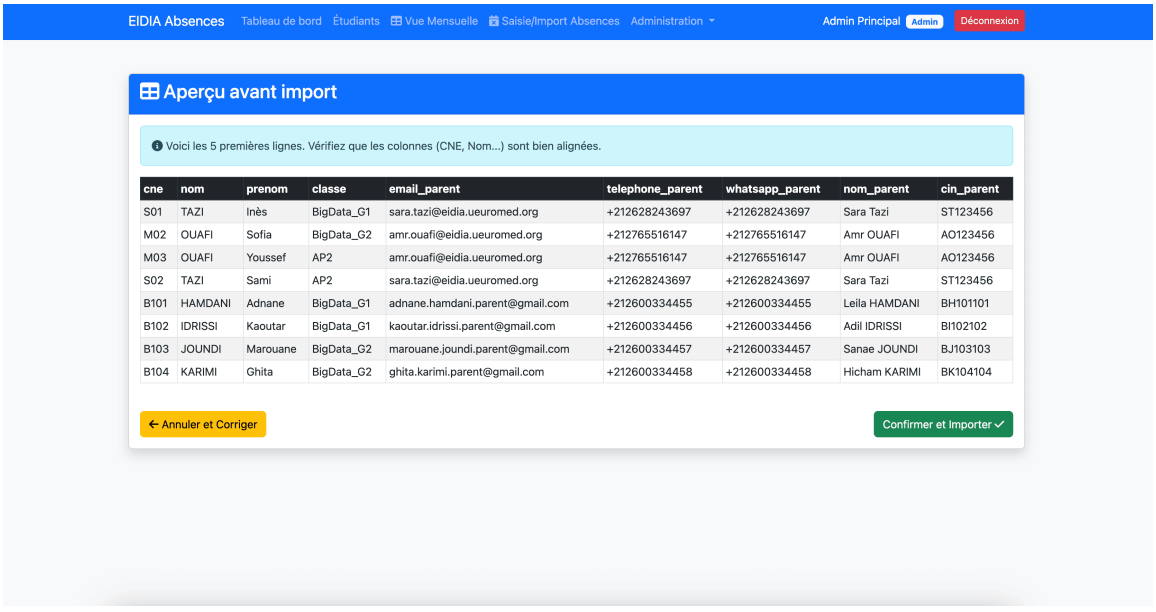

FIGURE 5.7 – Étape 3 : Prévisualisation des données avant commit final

5.4 Gestion des Données Académiques

La gestion quotidienne repose sur des listes interactives (*DataTables*) offrant des fonctionnalités avancées de recherche, de tri et de pagination.

5.4.1 Référentiel Étudiants

Cette vue permet de gérer la base élèves. Les actions CRUD sont accessibles via des boutons d'action contextuels sur chaque ligne.

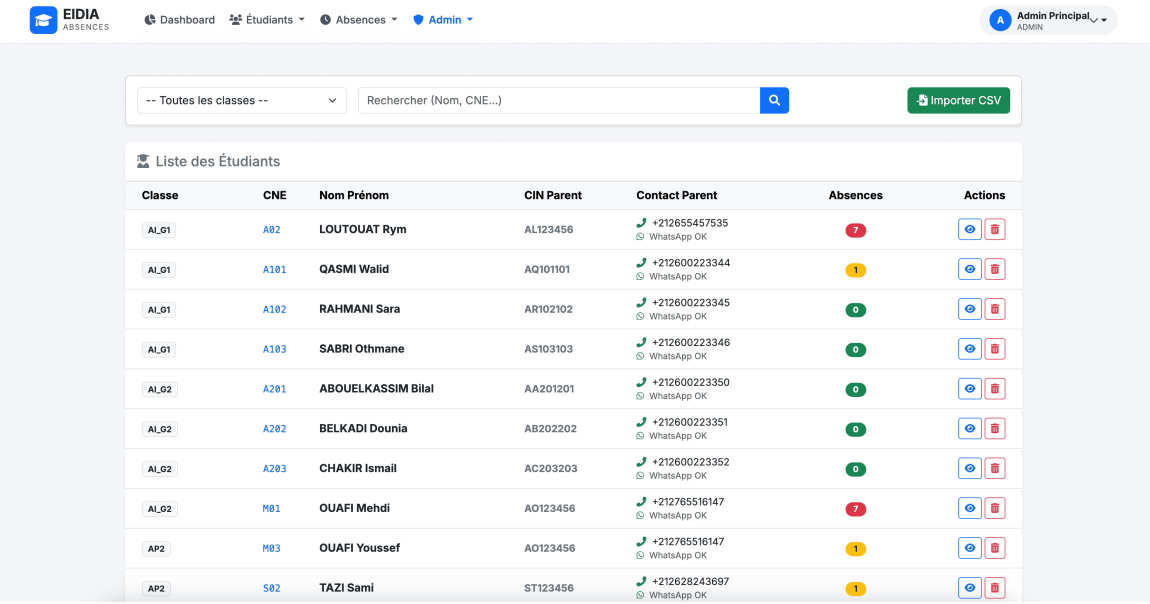

FIGURE 5.8 – Liste des étudiants avec recherche et filtres

5.4.2 Gestion des Parents

Une vue dédiée permet de lier les étudiants à leurs responsables légaux, et de vérifier l'état de leurs comptes.

5.4.3 Suivi des Absences

C'est l'outil de travail principal de l'opérateur. Le tableau intègre le *Sharding* temporel de manière transparente. Des badges de statut permettent d'identifier rapidement les actions à mener.

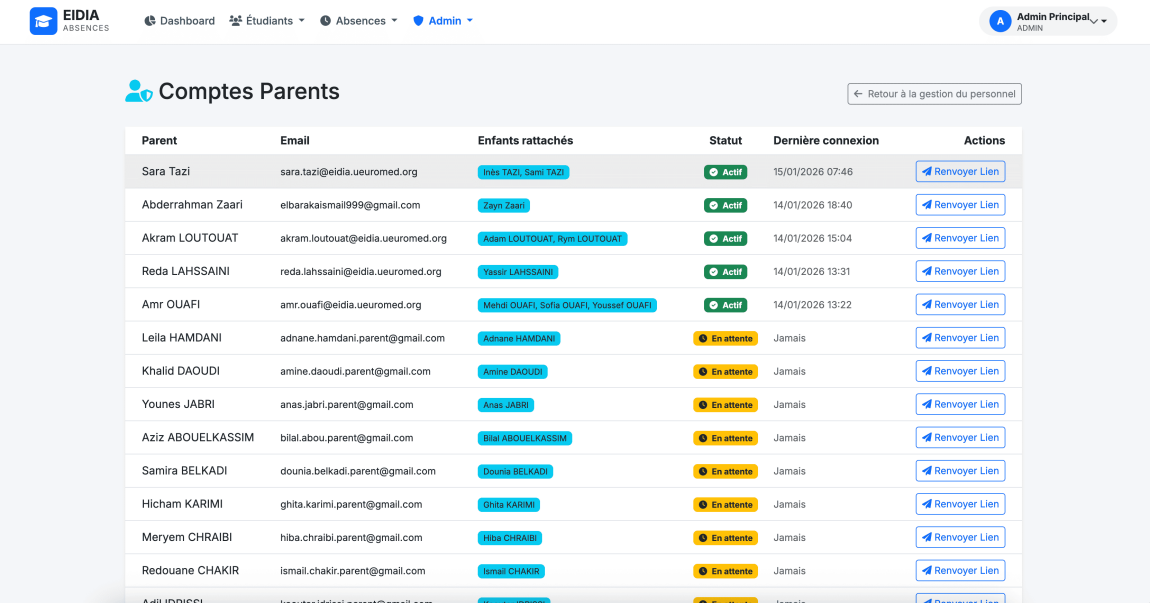

FIGURE 5.9 – Tableau de gestion des comptes parents

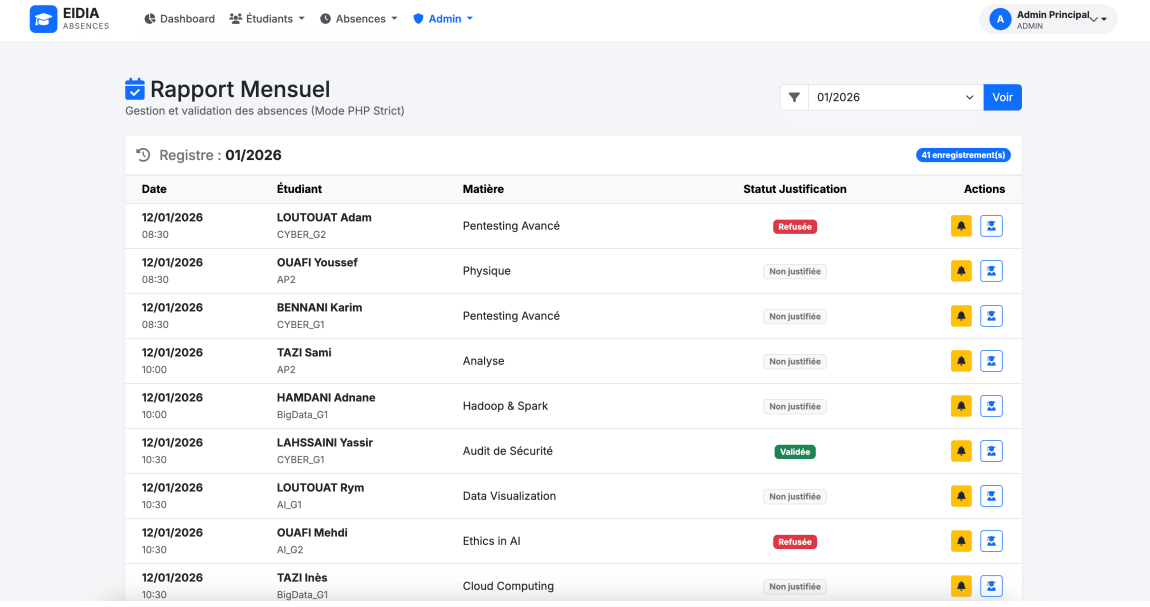

FIGURE 5.10 – Gestion mensuelle des absences et statuts de notification

5.5 Administration Système

Les interfaces de configuration sont isolées pour prévenir les modifications accidentelles.

5.5.1 Configuration Technique

Cette page permet à l'administrateur de modifier les paramètres critiques (Hôte SMTP, Clés API Twilio) à chaud.

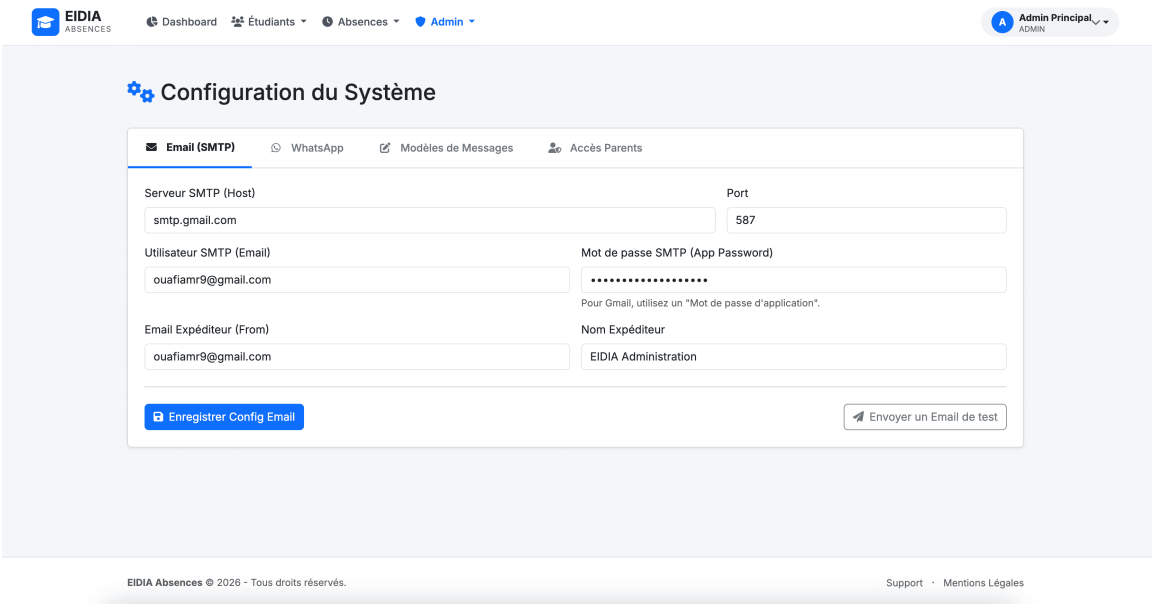

FIGURE 5.11 – Interface de Configuration Système (SMTP/API)

5.5.2 Gestion des Utilisateurs

L'interface de gestion des comptes permet à l'administrateur principal de créer des comptes opérateurs et de révoquer des accès en temps réel.

5.6 Justification Technologique : Pourquoi Bootstrap ?

L'utilisation du framework **Bootstrap 5** est un choix stratégique d'ingénierie logicielle motivé par trois facteurs :

— **Standardisation :** Bootstrap impose une grammaire visuelle commune. Cela évite la dette technique liée au CSS "spaghetti" et facilite la reprise du code par d'autres développeurs.

— **Accessibilité (a11y) :** Le framework intègre nativement les attributs ARIA et la gestion du focus clavier, garantissant une interface navigable pour tous.

— **Robustesse Cross-Browser :** Il normalise le rendu entre les différents navigateurs (Chrome, Safari, Firefox), assurant que le portail Parent fonctionne identiquement sur iOS et Android.

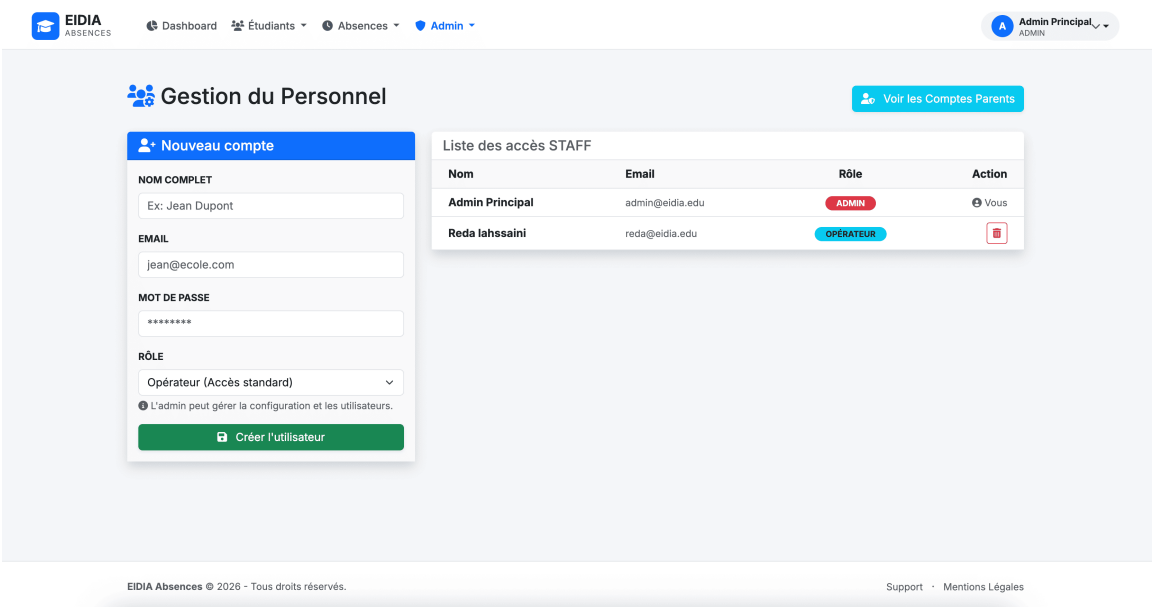

FIGURE 5.12 – Tableau de bord de gestion des utilisateurs internes

Chapitre 6

Sécurité et Robustesse (Security by Design)

La sécurité a été intégrée dès la phase de conception ("Security by Design") pour répondre strictement aux exigences du cahier des charges. Nous avons adopté une approche de "Défense en Profondeur", validant chaque entrée et protégeant chaque sortie.

6.1 Conformité aux Exigences de Sécurité

Le développement a été guidé par cinq contraintes critiques définies dans les spécifications techniques. Le tableau ci-dessous résume notre couverture de ces risques.

Catégorie	**Exigence Technique**
Uploads	Vérification MIME réel, Extension .csv, Taille $<$ 10Mo, Scan anti-injection
API Keys	Stockage sécurisé hors du code source
Rate Limit	Limitation à 100 envois/heure (Anti-Spam)
SQL	Requêtes préparées et validation des noms de tables dynamiques
Audit	Log complet des imports et envois (Conservation 2 ans)

TABLE 6.1 – Tableau de conformité des exigences de sécurité

6.2 Validation des Fichiers (Module Upload)

L'upload de fichiers étant le vecteur d'attaque le plus critique (risque de Webshell), nous avons implémenté une validation à trois niveaux dans `ImportController`.

6.2.1 1. Contrôle de Surface (Taille et Extension)

Avant tout traitement, nous vérifions les métadonnées basiques.

— **Taille :** Rejet immédiat si `filesize > 10 * 1024 * 1024` (10 Mo), protégeant le serveur contre le déni de service (DoS) par saturation disque.

— **Extension :** Seule l'extension `.csv` est acceptée (liste blanche stricte).

6.2.2 2. Analyse Binaire (MIME Type)

Une extension peut être falsifiée (ex : `virus.exe` renommé en `data.csv`). Nous utilisons `finfo_file` pour lire les "Magic Numbers" du fichier.

```
$finfo = finfo_open(FILEINFO_MIME_TYPE);
$mime = finfo_file($finfo, $path);
// Rejet si le MIME n'est pas text/csv, text/plain ou application/vnd.ms-excel
```

6.2.3 3. Inspection du Contenu (Anti-Injection)

Pour contrer les attaques polyglottes (code PHP caché dans un fichier texte valide), nous scannons le contenu.

```
$content = file_get_contents($path, false, null, 0, 2048);
if (strpos($content, '<?php') !== false) {
    AuditService::log('SECURITY_ALERT', "Tentative d'upload malveillant");
    unlink($path); // Destruction immédiate
    die("Fichier corrompu détecté.");
}
```

6.3 Sécurisation des Données et Secrets

6.3.1 Protection contre les Injections SQL

L'application interdit formellement la concaténation de variables dans les requêtes SQL.

- **Données :** Utilisation systématique de `PDO::prepare()` et `execute()`. Les données utilisateurs sont traitées comme des littéraux, neutralisant les commandes SQL injectées.
- **Structure (Sharding) :** Pour les tables dynamiques (`absences_MM_YYYY`), le nom de la table ne peut pas être préparé. Nous le sécurisons donc par une validation stricte via Regex :

```
// Validation du nom de table pour éviter "absences_01; DROP TABLE users;"
$cleanTable = preg_replace('/[^a-z0-9_]/', '', $tableName);
```

6.3.2 Gestion des Clés API

Conformément aux exigences, aucune clé API (Twilio, SMTP Password) n'est écrite en dur dans le code. Elles sont stockées soit dans un fichier de configuration externe non versionné, soit en base de données dans la table `configuration`, permettant une rotation des clés sans redéploiement.

6.4 Défense Active et Traçabilité

6.4.1 Rate Limiting (Protection Anti-Abus)

Pour éviter le spamming de SMS/Emails (coûteux) ou l'abus du système, un mécanisme de limitation de débit a été mis en place pour les notifications manuelles. Le système vérifie le nombre d'envois effectués par l'utilisateur sur la dernière heure glissante. Si le quota (100) est dépassé, l'action est bloquée.

6.4.2 Audit et Traçabilité (Logging)

Le service `AuditService` assure la non-répudiation des actions. Chaque événement critique est archivé avec :

- **L'Acteur :** ID utilisateur.
- **L'Action :** Type (IMPORT, NOTIFY, LOGIN_FAIL).
- **La Cible :** ID de l'étudiant ou nom du fichier concerné.
- **L'Horodatage :** `TIMESTAMP` précis.

Ces logs sont consultables via le dashboard Admin et permettent de retracer l'origine d'une erreur ou d'une action malveillante sur une période minimale de 2 ans.

Chapitre 7

Conclusion Générale

Ce projet de développement web avancé avait pour objectif de moderniser et sécuriser la gestion des absences universitaires. En partant d'une problématique concrète — la lourdeur du traitement manuel des fichiers hétérogènes — nous avons conçu et déployé une solution complète, robuste et automatisée.

7.1 Bilan de Conformité

Le succès de ce projet se mesure avant tout par sa conformité stricte avec le Cahier des Charges initial. Le tableau suivant synthétise l'adéquation entre les exigences demandées et les solutions techniques livrées.

Exigence du Cahier des Charges	**Solution Technique Implémentée**
Import CSV Intelligent Détection auto des colonnes et formats variés	Algorithme de **Fuzzy Matching** (Levenshtein) avec score de confiance > 75%
Gestion des Bases de Données Création automatique de tables par mois	**Sharding Temporel** dynamique (Tables `absences_MM_YYYY`)
Notifications Multicanales Envoi Email et WhatsApp aux parents	Service unifié intégrant **SMTP** (Email) et l'API **Twilio** (WhatsApp)
Sécurité des Fichiers Validation MIME, taille et contenu	Validation **"Paranoid Mode"** : Scan binaire + Analyse anti-Webshell
Expérience Utilisateur Distinction Admin / Parent	Interfaces distinctes avec Dashboard **Mobile First** pour les parents
Traçabilité Log des actions critiques	**AuditService** enregistrant chaque import, connexion et envoi en BDD

TABLE 7.1 – Matrice de conformité : Exigences vs Réalisations

7.2 Apports Techniques et Personnels

Au-delà de la réponse fonctionnelle, ce projet nous a permis de monter en compétence sur des architectures techniques avancées :

— **DevOps Réaliste :** Le déploiement sur une infrastructure Cloud (Google Compute Engine) avec un domaine réel (DuckDNS) et du HTTPS (Let's Encrypt) nous a sortis du confort du `localhost`.

— **Sécurité Offensive/Défensive :** La mise en place de protections contre les failles WebShell et SQL Injection nous a sensibilisés aux enjeux de la cybersécurité moderne.

— **Architecture Logicielle :** L'application stricte du pattern MVC et l'utilisation de Services dédiés garantissent un code maintenable et évolutif.