Lilian Schott

Dossier technique bloc 4 RNCP

Introduction

Contexte du projet

TMT est un outil de gestion interne permettant d’automatiser la création des conventions de formation. Il centralise les données liées à une formation (clients, participants, formateurs, formations, séances et conventions) avant d’automatiser la génération, l'envoi et le stockage des documents, conservés dans un volume dédié et dans un Drive d'entreprise. Les privilèges sont répartis entre les formateurs, les administrateurs et le directeur de l'organisme de formation. Le projet dispose d'une version déployée sur un VPS OVH.

Technologies utilisées

Le projet a été conçu sur une architecture 3-tiers avec un front Angular, un back Symfony et une base de données PostgreSQL. Pour la CI/CD, on utilise GitLab CI : la pipeline est initialement composée de quatre stages (lint, build, test et deploy) et intègre également des étapes de sécurité (analyse SAST et détection de secrets via Gitleaks). Pour les besoins de ce projet, on ajoutera par la suite un stage dependencies. L'application est conteneurisée avec Docker et orchestrée avec docker-compose.

Environnement de production

L'application est déployée sur un VPS OVH sous Debian et orchestrée avec docker-compose. Elle est composée de trois services conteneurisés : le front Angular (servi via un conteneur exposé sur le port 4200), le back Symfony (exposé sur le port 8000, en mode APP_ENV=prod) et la base de données PostgreSQL 16. La communication entre services est gérée par Docker, le back étant configuré pour ne démarrer qu'une fois la base disponible (condition service_healthy). La persistance des données est assurée par des volumes Docker dédiés : les données PostgreSQL, les fichiers uploadés et le token d'accès au Google Drive d'entreprise sont conservés dans des volumes distincts pour garantir leur conservation entre les redéploiements des conteneurs. La configuration sensible (secret applicatif, identifiants de base de données, clés JWT, DSN du service de mail, dossier Drive) n'est jamais inscrite en dur : elle est injectée au conteneur via des variables d'environnement, elles-mêmes stockées de façon sécurisée côté serveur et dans les variables protégées de la pipeline.

Le déploiement est automatisé par le stage deploy de la pipeline GitLab, déclenché sur la branche main. Le job se connecte au serveur en SSH (clé privée stockée en variable CI masquée), récupère la dernière version du code, s'authentifie auprès du registre de conteneurs, puis récupère et redémarre les images à jour via docker compose pull et docker compose up -d. Un nettoyage des images obsolètes (docker image prune) est effectué à chaque déploiement afin de limiter l'occupation disque du serveur.

```yaml
db:
  image: postgres:16-alpine
  environment:
    - POSTGRES_DB=${POSTGRES_DB}
    - POSTGRES_USER=postgres
    - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
  volumes:
    - postgres_data:/var/lib/postgresql/data
  healthcheck:
    test: ["CMD-SHELL", "pg_isready -U postgres"]
    interval: 5s
    timeout: 5s
    retries: 5
```

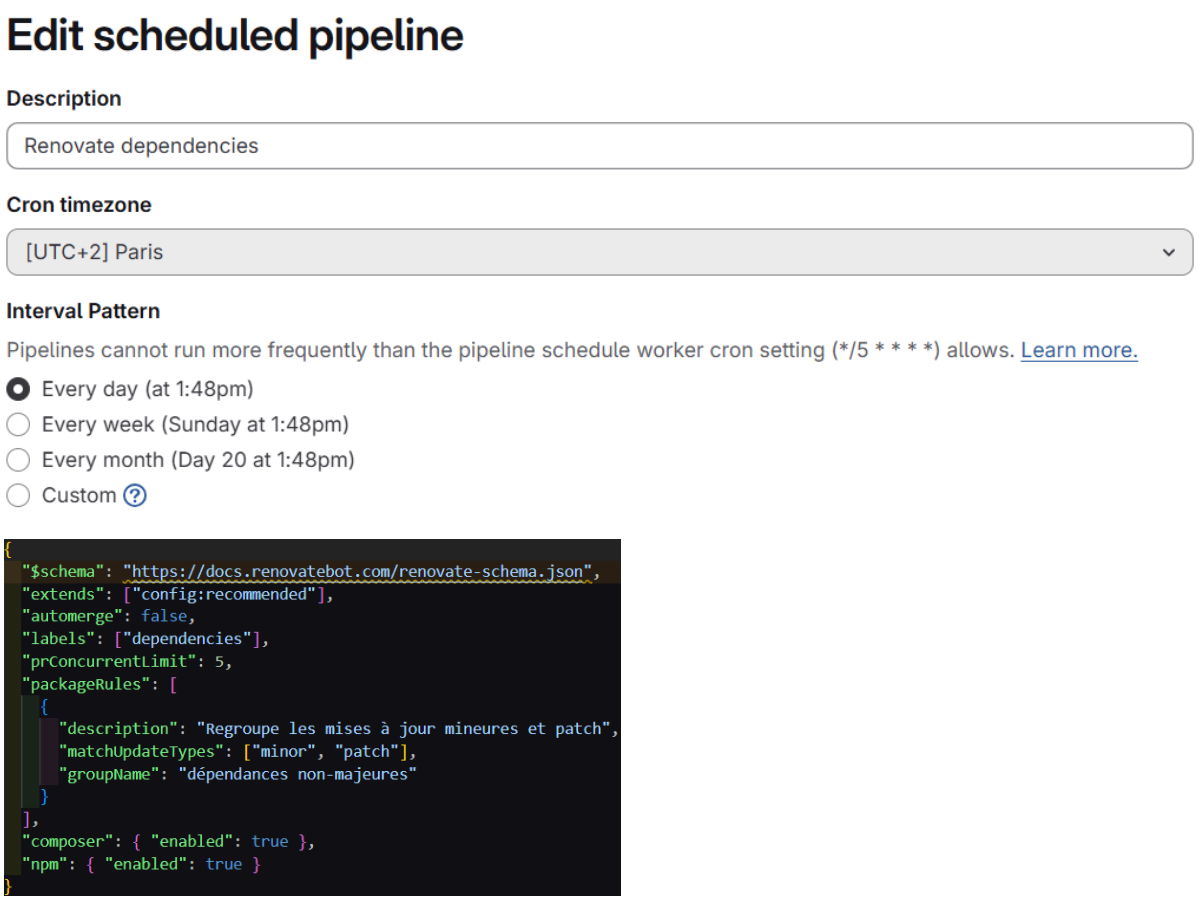
Monitoring

Le maintien du logiciel en condition opérationnelle repose sur deux volets complémentaires : la mise à jour régulière et sécurisée des dépendances et la supervision de l'application en production, permettant de détecter et de remonter les anomalies.

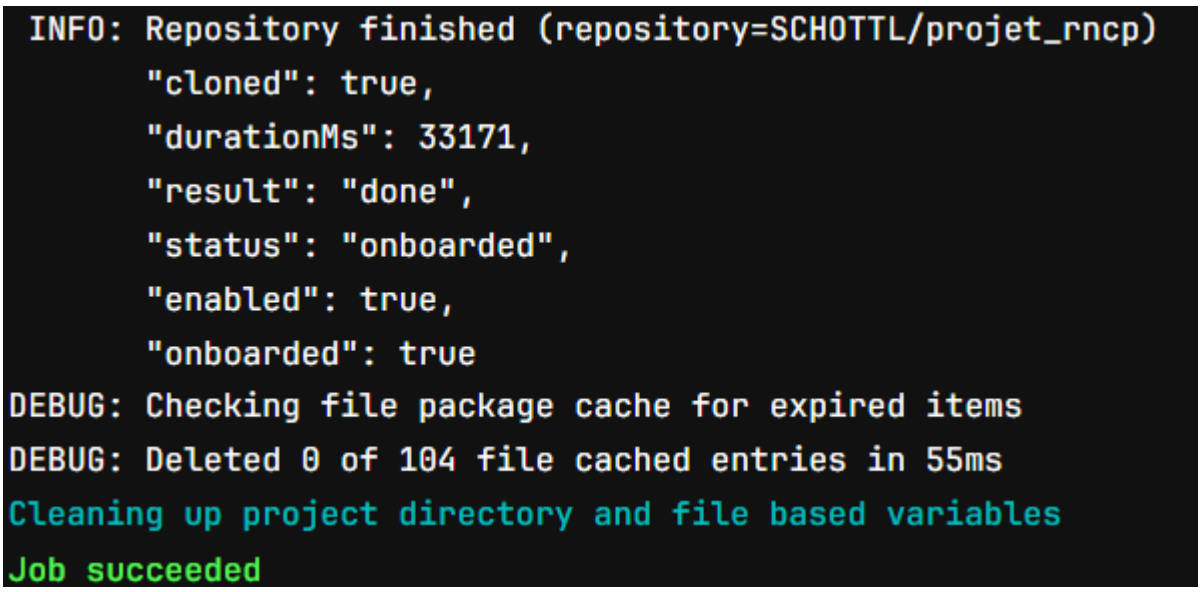
Mise à jour des dépendances

Initialement, les vulnérabilités sur les dépendances sont vérifiées à chaque commit par le job build (que ce soit pour le front ou pour le back). Cependant, il est crucial d’ajouter une vérification planifiée des dépendances afin de s’assurer qu’elles sont à jour sans avoir à faire un commit déclenchant la pipeline. Pour cela, on utilisera Renovate. La 1ère étape est de mettre à jour la pipeline en ajoutant un job exécutant l’image Docker Renovate. Ensuite, il faut créer un job schedule dans l’interface de GitLab. Dans ce cas, j’ai choisi de le paramétrer pour qu’il s’exécute automatiquement tous les jours (à noter qu’il peut également être exécuté à la main). L’audit des dépendances couvre l'intégralité du projet : les paquets

du back gérés par Composer, les paquets du front gérés par NPM, les images Docker utilisées par les conteneurs et les composants de la pipeline elle-même. Deux niveaux se complètent. À chaque commit, les jobs du stage build signalent les vulnérabilités connues sur les dépendances existantes : composer audit côté back, npm audit côté front. Quotidiennement, la scheduled pipeline exécute le job Renovate, qui détecte les nouvelles versions disponibles même sans commit. La détection est automatique, l'intégration reste manuelle : l'automerge est volontairement désactivé. Chaque montée de version proposée donne lieu à une merge request qui traverse l'intégralité de la pipeline (lint, build, tests unitaires front et back avec un coverage minimal imposé, analyse SAST et détection de secrets avec Gitleaks), la fusion demeurant une décision humaine, prise au vu du rapport de pipeline et du changelog de la dépendance. Les mises à jour mineures et correctives, à faible risque de rupture, sont regroupées dans une merge request unique. Les majeures, susceptibles d'introduire des ruptures de compatibilité, restent isolées et traitées une par une afin d'en circonscrire l'impact. En cas de régression détectée après déploiement, le retour arrière s'effectue par revert du commit fautif puis redéploiement via la pipeline .




```
{
  "$schema": "https://docs.renovatebot.com/renovate-schema.json",
  "extends": ["config:recommended"],
  "automerge": false,
  "labels": ["dependencies"],
  "prConcurrentLimit": 5,
  "packageRules": [
    {
      "description": "Regroupe les mises à jour mineures et patch",
      "matchUpdateTypes": ["minor", "patch"],
      "groupName": "dépendances non-majeures"
    }
  ],
  "composer": { "enabled": true },
  "npm": { "enabled": true }
}
```



Système de supervision et d’alerte

La supervision de l'application repose sur quatre dispositifs complémentaires : la journalisation applicative, la remontée centralisée des erreurs, la surveillance de la disponibilité de l'infrastructure et de l’application, et un ensemble de seuils et de modalités de signalement définissant le déclenchement des alertes. On y ajoute un error management conçu pour ne pas exposer d'informations sensibles en production. Le périmètre supervisé couvre les trois services conteneurisés (front Angular, back Symfony, base PostgreSQL) ainsi que les événements de sécurité applicatifs.

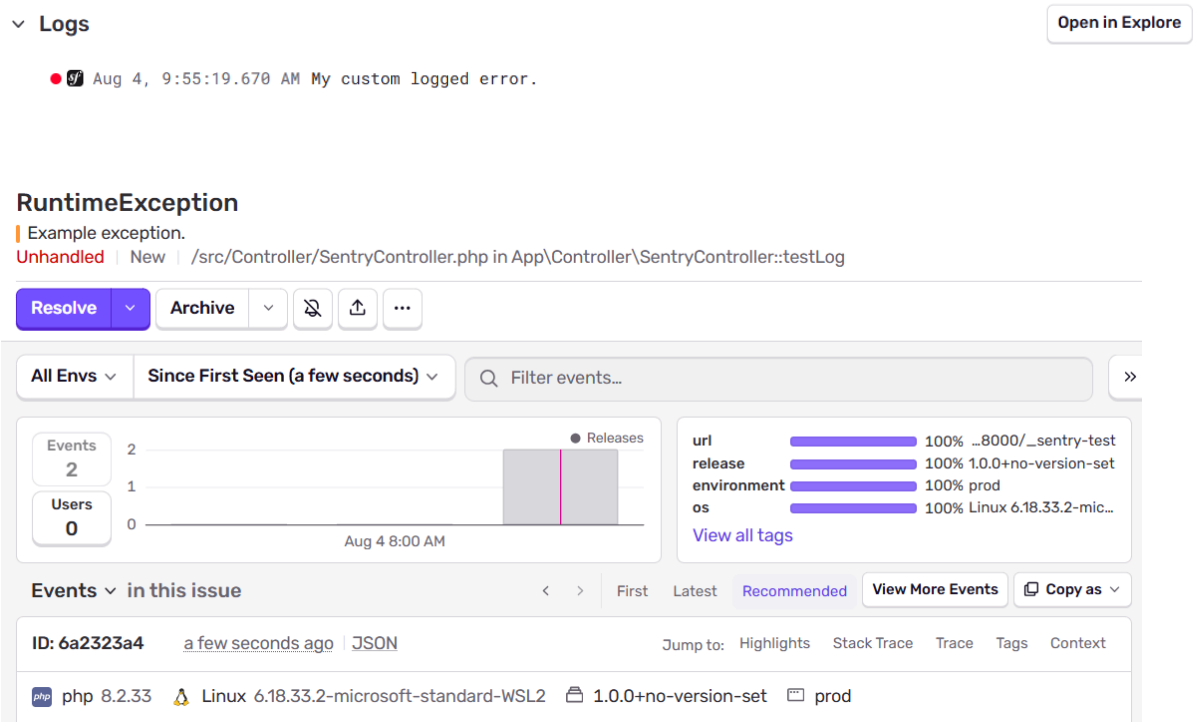
Journalisation applicative

La journalisation est assurée par Monolog. Un canal dédié security a été configuré pour isoler les événements de sécurité dans un fichier distinct (security.log), à partir du niveau warning. Cette séparation facilite le suivi et l'analyse des événements sensibles, en les distinguant du reste des journaux applicatifs.

```yaml
monolog:
    handlers:
        security:
            type: stream
            path: '%kernel.logs_dir%/security.log'
            level: warning
            channels: [security]
        sentry_logs:
            type: service
            id: Sentry\SentryBundle\Monolog\LogsHandler
```

Remontée centralisée des erreurs

L'application intègre Sentry (bundle sentry/sentry-symfony) afin de centraliser la remontée des erreurs. Sentry capture les exceptions non gérées ainsi que les erreurs journalisées par l'application, et les regroupe dans une interface de suivi permettant de les qualifier et de les prioriser. Le bon fonctionnement de cette remontée est vérifiable grâce à un endpoint de test dédié, qui déclenche volontairement une erreur journalisée puis une exception non capturée, permettant de confirmer que les deux types d'événements sont bien transmis.



Surveillance de la disponibilité

La disponibilité de la base de données est surveillée au niveau de l'infrastructure. Un healthcheck est configuré sur le conteneur PostgreSQL (commande pg_isready, exécutée à intervalle régulier) et le back n'est démarré qu'une fois la base déclarée disponible pour éviter les erreurs liées à un démarrage prématuré des services. La disponibilité applicative est vérifiée depuis l'extérieur du serveur par une sonde HTTP (UptimeRobot) qui interroge à intervalle régulier la page d'accueil du front et un endpoint du back. Deux échecs consécutifs déclenchent une notification vers l'adresse de maintenance. Cela permet de détecter une indisponibilité que la supervision interne ne verrait pas (panne du VPS, du reverse proxy ou du conteneur front). Le signalement suit une règle unique : toute nouvelle issue remontée par Sentry déclenche une notification par mail vers l'adresse de maintenance. Une issue avec une criticité majeure donne lieu à l'ouverture immédiate d'un ticket selon le gabarit de consignation décrit dans la section suivante, les événements de moindre criticité étant agrégés dans l'interface Sentry et traités lors des revues de maintenance. TMT étant un outil de gestion interne, utilisé en heures ouvrées par un nombre restreint d'utilisateurs, les critères retenus sont les suivants :

- Disponibilité : 99 % en heures ouvrées, la génération et l'envoi des conventions ne tolérant pas d'interruption prolongée en période de contractualisation.
- Temps de réponse : moins de 500 ms au 95e centile sur les endpoints de consultation, la génération de documents PDF faisant l'objet d'un seuil distinct plus permissif.
- Taux d'erreurs serveur : moins de 1 % des requêtes en réponse 5xx.
- Délai de prise en compte : moins d'un jour ouvré pour une anomalie de criticité majeure signalée par le support.
- Coverage : seuil minimal de 75 % côté back et 80% côté front imposé et vérifié automatiquement par la pipeline, tout correctif devant maintenir ce niveau.

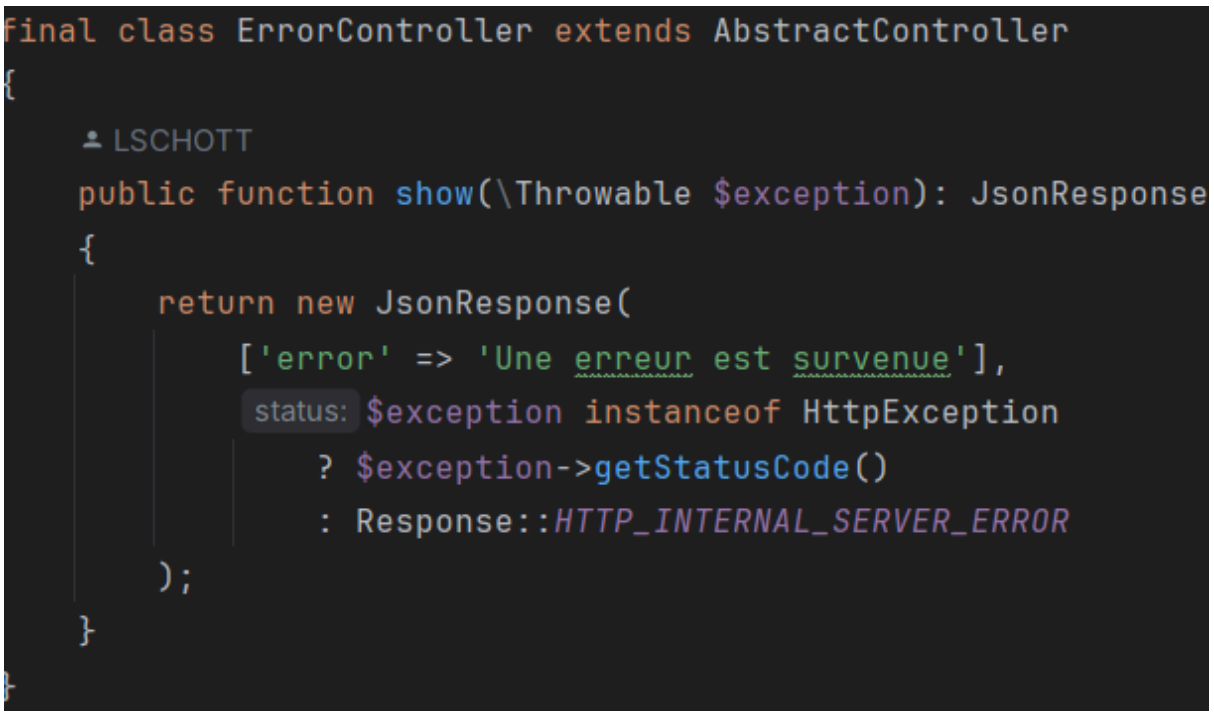
Sonde	**Périmètre**	**Finalité**	**Seuil de déclenchement**	**Signalement**
Healthcheck pg_isready	DB Postgres	Vérifier la disponibilité de la base de données	5 échecs de suite à 5s d’intervalle passent le conteneur en unhealthy	Conteneur unhealthy, démarrage du back bloqué

Exceptions non gérées par Sentry	Back Symfony	Détecter les erreurs 500	Toute nouvelle issue créée	Notification immédiate par email
Erreurs journalisées par Sentry	Applicatif	Remonter les erreurs métier captées par Monolog	Niveau ERROR et au-delà	Notification immédiate par email
Canal security Monolog	Auth	Tracer les événements sensibles	Niveau warning et au-delà	Consignation dans security.log
Sonde HTTP externe	Front Angular + back Symfony	Vérifier la disponibilité de l'application	2 échecs consécutifs à 5 min d'intervalle	Notification immédiate par email

La disponibilité et le temps de réponse font actuellement l'objet d'une détection par incident : une indisponibilité ou une dégradation entraîne une remontée d'erreur, mais l'évolution de ces indicateurs n'est pas tracée. Le passage d'une détection ponctuelle à une mesure continue constitue l'axe “Extension de la supervision” présenté plus loin, chiffré à 5 jours et sans coût de licence.

Gestion des erreurs en production

Enfin, un contrôleur d'erreur dédié intercepte les exceptions et renvoie à l'utilisateur une réponse JSON générique (« Une erreur est survenue »), tout en conservant le code HTTP approprié : le code de l'exception HTTP lorsqu'il est disponible, ou un code 500 par défaut. La stacktrace détaillée n'est ainsi jamais exposée au client, ce qui évite la divulgation d'informations techniques exploitables, tandis que le détail de l'erreur reste disponible côté serveur (journaux et Sentry) pour l'analyse. Cette gestion répond à la fois à un objectif de sécurité et de supervision.



Traitement des anomalies en production

Processus de collecte

Le traitement d'une anomalie repose d'abord sur sa détection puis sur sa consignation structurée. Les anomalies survenant en production sont identifiées à partir de plusieurs sources : les alertes remontées automatiquement par Sentry lors d'une exception non gérée, les journaux applicatifs produits par Monolog (notamment le canal security dédié aux événements sensibles), et les retours des utilisateurs. La gestion des erreurs a été conçue pour préserver ces informations sans les exposer : le contrôleur d'erreur de l'application renvoie à l'utilisateur un message générique tout en conservant le code HTTP approprié, tandis que la stacktrace complète reste journalisée côté serveur et transmise à Sentry. Ces éléments constituent la matière première de l'analyse. Chaque anomalie détectée est ensuite consignée dans l'outil de suivi du projet, selon un gabarit standardisé garantissant que toutes les informations nécessaires à sa reproduction et à son analyse sont réunies : contexte de survenue, environnement concerné, criticité, étapes de reproduction, comportement attendu et comportement observé, extrait de journal ou de stacktrace, analyse de la cause et correction appliquée. Cette consignation assure la traçabilité des incidents et permet de déterminer le correctif à mettre en place.

Traitement d’une anomalie détectée

Une fois l'anomalie consignée et analysée, le correctif est développé puis déployé en s'appuyant sur le processus d'intégration et de déploiement continu du projet. La correction fait l'objet d'un commit et suit l'ensemble des étapes de la pipeline (lint, build, tests) avant d'être déployée en production pour éviter d’introduire des régressions. La résolution est ensuite vérifiée, la disparition de l'anomalie étant confirmée par l'absence de nouvelle remontée dans Sentry. Par exemple, une anomalie de ce type a été traitée au cours du projet : la suppression d'un formateur rattaché à une formation échouait à cause d’une contrainte liée aux clés étrangères entre les 2 entités. L'anomalie a été signalée en production par un administrateur, consignée sous la référence BUG 001, puis analysée et corrigée selon le processus décrit ci-dessus. La fiche de consignation correspondante figure ci-après.

Fiche de consignation

Identifiant	BUG 001
Titre	Suppression d’un formateur impossible s’il est lié à une formation
Date de détection	21/07/2026
Détecté par	Retour utilisateur
Environnement de détection	Production
Environnement d’analyse	Développement
Composants concernés	Entités Symfony
Criticité	Majeure
Statut	Résolu

Lors de la suppression d'un formateur relié à au moins une formation, l'opération échoue. La base de données rejette la suppression en raison d'une contrainte de clé étrangère : le formateur est référencé par des enregistrements liés (formations). Cela empêche sa suppression tant que ces références existent.

Etapes de reproduction

1 : Créer un formateur
2 : Créer une formation avec ce formateur
3 : Tenter de supprimer le formateur depuis l'interface d'administration.
4 : La suppression échoue et une erreur est renvoyée.

Comportement attendu

La suppression d'un formateur doit s'effectuer sans laisser la base dans un état incohérent, et sans erreur pour l'utilisateur.

Comportement observé

L'opération est interrompue par une violation de contrainte d'intégrité référentielle (clé étrangère). Le formateur n'est pas supprimé.

Message d’erreur

La gestion des erreurs en production ne renvoie qu'un message générique afin de ne pas exposer d'information sensible. L'anomalie a été retestée en environnement de développement pour récupérer la stacktrace complète. Cette bascule permet d'obtenir le détail nécessaire à l'analyse sans dégrader le niveau de sécurité de l'environnement de production.

backend-1 | 127.0.0.1 - 21/Jul/2026:08:19:09 +0000 "DELETE /index.php" 500
db-1 | 2026-07-21 08:19:09 UTC [290] ERROR: update or delete on table "formateur" violates foreign key constraint "fk_404021bffb08edf6" on table "formation"
db-1 | 2026-07-21 08:19:09 UTC [290] DETAIL: Key (id)=(1) is still referenced from table "formation".
db-1 | 2026-07-21 08:19:09 UTC [290] STATEMENT: DELETE FROM formateur WHERE id = $1

Cause du bug

Les entités Formateur et Formation sont liées par une clé étrangère. Par défaut, PostgreSQL empêche la suppression d'une ligne encore référencée, afin de préserver l'intégrité des données. Le comportement de suppression n'avait pas été explicitement défini au niveau de l'entité.

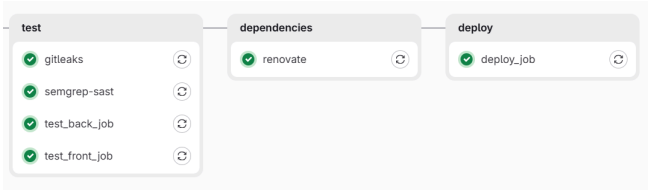
Correction appliquée

Pour corriger le problème, j’ai réfléchi à 3 solutions. La première consiste à conserver la contrainte en l'état et à interdire la suppression tant que des enregistrements liés existent, en imposant à l'utilisateur de détacher au préalable les formations concernées. C'est la solution la plus conservatrice, mais elle reporte sur l'utilisateur une manipulation fastidieuse et ne répond pas au besoin exprimé par le client. La 2ème est de neutraliser la référence en base (SET NULL). Je n’ai pas retenu cette solution car une formation sans formateur n’est pas cohérente et ne peut donc pas être enregistrée. L’ajout de la suppression en cascade a été retenu comme correctif de première intention car cela permet de résoudre le problème relevé par l’utilisateur et de préserver la cohérence référentielle de la base. Le comportement de suppression a donc été défini au niveau des entités concernées : la suppression d'un formateur entraîne celle des formations qui lui sont rattachées, et par propagation celle des conventions et des séances associées. Les entités Client, Formation et Convention ont été traitées de la même manière, l'anomalie étant susceptible de se reproduire sur chacune de ces relations. Cette correction reste une solution à court terme :

elle résout le blocage fonctionnel mais autorise la disparition d'enregistrements liés à des documents à valeur contractuelle malgré leur conservation sur le Drive. Une évolution vers une suppression logique est en conséquence proposée en axe d'amélioration.

Vérification

La nouvelle version a été poussée sur Git puis a suivi l'ensemble des étapes de la pipeline (lint, build, tests) avant d’être redéployée. Un nouveau test manuel a ensuite permis de vérifier que la suppression d'un formateur rattaché à une formation s'effectue désormais sans erreur, et aucune nouvelle remontée n'apparaît dans Sentry.



Maintenance du logiciel

Axes d’amélioration

Maintenant que la V1 est fonctionnelle, voici plusieurs axes d’amélioration pour la suite. Ces axes découlent de deux sources : les limites identifiées lors de la mise en place de la supervision, et les retours des utilisateurs recueillis par le support. C'est notamment le signalement à l'origine de BUG 001 qui a révélé le besoin d'un traitement plus fin de la suppression des données.

Ajout de la signature électronique

L'application génère et envoie les conventions de formation, mais leur signature s'effectue aujourd'hui en dehors de l'outil. L'intégration d'un service de signature électronique permettrait de gérer le cycle complet du document, de la génération à la signature, directement dans l'application : les gains attendus sont un renforcement de l'automatisation, la traçabilité des signatures et une valeur probante accrue des documents, ce qui en fait l'amélioration la plus visible pour l'utilisateur final. Son implémentation ne nécessite que l'intégration d'une API tierce, mais les services conformes reposent sur un abonnement payant : cette dépendance à un budget récurrent explique qu'elle n'ait pas été intégrée dans le cadre de ce projet.

Suppression logique

La correction du bug de suppression a été réalisée via une suppression en cascade : supprimer un formateur entraîne la suppression des formations qui lui sont rattachées. Or l'application gère des conventions à valeur contractuelle, dont la suppression automatique peut entraîner la perte de données à conserver malgré le fait que les conventions soient conservées dans le Drive quoi qu'il arrive. Le passage à une suppression logique préserverait l'historique et les documents associés, tout en retirant l'entité des vues actives. Cette solution permettrait de gagner en fiabilité et en conformité, sans dégrader l'expérience utilisateur. Pour cela, il suffit d'ajouter un indicateur d'archivage sur les entités concernées et d'adapter les requêtes de lecture pour filtrer les enregistrements archivés. Cette solution ne nécessite aucune dépendance externe.

Automatisation des mises à jour

La mise à jour des dépendances est aujourd'hui détectée par les audits en intégration continue et par Renovate qui propose les montées de version. Le processus peut être consolidé pour couvrir davantage de cas et fiabiliser l'application des correctifs de sécurité. L'objectif principal est de réduire la dette technique et la fenêtre d'exposition aux vulnérabilités connues avec un maintien de l'application à jour plus régulier. La mesure à mettre en place est l'ajustement de la configuration existante (fréquence, regroupement, règles d'automerge sur les mises à jour mineures après validation des tests). Cette évolution n'implique aucun coût supplémentaire.

Extension de la supervision

La supervision actuelle repose sur la journalisation (Monolog), la remontée d'exceptions (Sentry) et une sonde externe. Elle pourrait être étendue au suivi des indicateurs de performance et de disponibilité dans la durée. Cette amélioration permettra une détection plus précoce des dégradations (temps de réponse, taux d'erreur), une meilleure visibilité sur la santé de l'application en production et la capacité d'anticiper les incidents plutôt que de les subir. Cette mesure nécessite l'ajout de quelques métriques applicatives jusqu'à la mise en place d'un outillage de métrologie plus complet. Des solutions open source existent, sans coût de licence.

Axe	Charge de travail estimée	Coût récurrent	Gain	Priorité
Suppression logique	3j	Aucun	Préservation des documents à valeur contractuelle	Elevée
Extension de la supervision	5j	Aucun	Mesure continue des critères de performance	Elevée

Automatisation des mises à jour	1j	Aucun	Réduction de la fenêtre d'exposition	Moyenne
Signature électronique	8j	Abonnement tiers	Cycle documentaire complet, valeur probante	Moyenne

Journal des versions

Version	Date	Modification
0.1.0	16/03/2026	Ajout du CRUD sur les clients, formateurs, formations, séances, participants et conventions. Ajout de la génération de documents. Ajout de l'envoi de documents
0.2.0	21/03/2026	Ajout de l'authentification via JWT. Ajout des rôles utilisateurs. Ajout des mots de passe oubliés. Mise en place du hachage des mots de passe. Mise en place de la documentation OpenAPI avec Nelmio

0.3.0	09/05/2026	Orchestration de l'application avec docker-compose (services pour le front, le back et la base de données). Création d'un jeu de tests unitaires pour le front et le back.
0.4.0	12/05/2026	Ajout de règles de validation sur les entités. Mise en place du système de supervision et de journalisation. Implémentation du rate limiting avec un limiter spécial pour le login. Ajout d'en-têtes HTTP.
0.5.0	25/05/2026	Export des documents générés vers un Google Drive. Chaque convention a un dossier créé avec le nom du client, de la formation et la date d'entrée en formation.
0.6.0	01/06/2026	Ajout de la blocklist JWT pour invalider un token avant son expiration (déconnexion, révocation).

0.7.0	22/06/2026	Mise en place de la pipeline CI/CD (lint, test, build et deploy). Intégration de SAST et Gitleaks dans la pipeline. Ajout d'un coverage minimal requis pour les tests unitaires. Rotation d'APP_SECRET à la suite de sa détection dans l'historique du dépôt, et vérification de l'absence de nouvelle exposition via Gitleaks. Ajout du versioning sur l'API, la v1 comporte le CRUD, la génération de documents et l'envoi, la v2 ajoute l'export vers Drive.
0.7.1	21/07/2026	Correction du BUG 001 sur la suppression d'un formateur rattaché à une formation.
1.0.0	27/07/2026	Ajout de Renovate pour automatiser la mise à jour des dépendances.

Collaboration avec un support client

Partie prenante	Contribution
Administrateur de l’organisme de formation	Signalement, description du contexte métier et de l'impact
Support client	Recueil des éléments de reproduction, vérification du caractère systématique, exclusion d'une erreur d'usage, consignation du ticket et transmission
Equipe technique	Analyse de la cause, arbitrage entre options de correction, développement et déploiement via la pipeline, vérification et retour au support

Contexte du retour client

Un administrateur de l'organisme de formation contacte le support pour signaler un dysfonctionnement : lorsqu'il tente de supprimer un formateur, la suppression échoue et l'application affiche un message d'erreur générique avec une erreur 500, sans supprimer les données. Le besoin métier est légitime : l'organisme souhaite pouvoir retirer de l'outil les formateurs qui ne collaborent plus avec lui, afin de conserver des données à jour.

Qualification et remontée

Le support recueille les informations nécessaires à la reproduction : le compte concerné, l'action effectuée, la date et l’heure de l’incident, et la capture du message affiché. Il constate que le problème se reproduit systématiquement dès lors que le formateur à supprimer est rattaché à au moins une formation. Ne relevant pas d'une simple erreur d'utilisation, l'incident est consigné (fiche BUG 001) puis transmis à l'équipe technique pour analyse.

Solution apportée

Après analyse, on constate que l’erreur provient d'une contrainte d'intégrité Postgres : la base de données empêche la suppression d'un formateur encore référencé par des formations, les entités formations, séances et conventions sont également concernées par ce problème. La solution à appliquer est de définir le comportement de suppression au niveau des entités concernées, de façon à traiter les enregistrements liés. Dans ce cas, on applique la suppression en cascade. Le correctif est développé, testé, puis déployé via la pipeline CI/CD. Après déploiement, la suppression d'un formateur rattaché à des formations s'effectue sans erreur.

Conclusion

Pour conclure, j'ai mis en place un suivi automatisé des dépendances, avec une vérification à chaque commit et une analyse planifiée via Renovate, ainsi qu'un système de supervision reposant sur la journalisation, la remontée centralisée des erreurs vers Sentry et la surveillance de la disponibilité de la base de données et de l’application, chaque sonde étant associée à un seuil de déclenchement, à une modalité de signalement et à des critères de qualité et de performance définis pour le projet. Pour le traitement des anomalies, j'ai défini un processus de consignation structuré, illustré par une fiche de consignation pour répertorier les bugs puis les corriger, tout en m'appuyant sur la pipeline CI/CD pour valider et déployer les correctifs. La maintenance s'accompagne d'un journal des versions retraçant l'évolution du projet et d'un exemple de problème résolu en collaboration avec le support. Enfin, plusieurs axes d'amélioration ont été identifiés pour la suite, notamment l'intégration de la signature électronique des conventions et le passage à une suppression logique plutôt qu’une suppression en cascade des données afin de préserver les documents à valeur contractuelle. TMT dispose ainsi d'une base fiable et maintenable, tout en conservant des perspectives d'évolution.