



****

**Stage effectué à :**

SQLI Maroc (Oujda)

**Soutenance le 28/06/2025 devant le Jury :**

M/MME ………………………………………….

M/MME ………………………………………….

M/MME ………………………………………….

**ECOLE DES HAUTES ETUDES D’INGENIERIE – OUJDA**

**Mémoire de Projet de Fin d’Etude**

Présenté en vue de l’obtention du

**DIPLÔME D’INGENIEUR D’ETAT**

FILIERE : Génie Informatique

**Année universitaire : 2024 - 2025**

**Encadré par :**

M Issam KHADIRI

M Mohamed RAHMOUNI

**Réalisé par :**

M Mohamed-Amine BENALI

Analyse approfondie en vue de la migration du projet Autossimo de Symfony 5.4 vers 6.4

**²**Une image contenant noir, obscurité

Le contenu généré par l’IA peut être incorrect.

# Dédicaces

**À mes très chers parents, aucun mot ne saurait exprimer toute ma reconnaissance envers vous. Merci pour vos efforts constants, vos sacrifices silencieux, vos prières bienveillantes et votre amour inconditionnel. Que Dieu vous accorde santé, bonheur et longue vie.**

**À mon cher frère, à mon oncle, à toute ma famille ainsi qu'à tous mes amis, je vous remercie sincèrement pour votre présence, votre soutien et votre affection tout au long de ce parcours.**

**À mes professeurs, je suis profondément reconnaissant pour votre engagement, votre rigueur et votre disponibilité. Vos enseignements ont nourri mon esprit et m’ont guidé sur le chemin de l’excellence.**

**À tous mes amis, merci pour votre soutien indéfectible, vos encouragements et les moments partagés. Votre présence a été précieuse et réconfortante.**

**Merci à tous ceux qui m’aiment et m’ont soutenu. Ce travail vous est dédié.**

# *Remerciements*

Je tiens à exprimer ma profonde gratitude et mes sincères remerciements à M. Issam Boukhari, Skill Manager au sein de SQLI-Oujda, pour m’avoir accueilli dans son équipe et pour les conditions favorables mises en place tout au long de mon stage.

Mes remerciements les plus distingués vont à M. Mohamed Rahmouni, mon encadrant technique, pour son accompagnement constant, sa disponibilité, ainsi que pour l’attention et le soutien précieux dont il a fait preuve durant toute la période du stage. Son expertise et ses conseils ont grandement contribué à la réalisation de ce travail.

Je remercie également M. Issam Khadiri, qui a assuré le double rôle d’encadrant pédagogique à l’École des Hautes Études d’Ingénierie et de collaborateur au sein de SQLI. Sa disponibilité, son implication, ainsi que ses recommandations éclairées ont été d’une aide inestimable tant sur le plan académique que professionnel.

Mes remerciements vont également à l’ensemble du personnel de SQLI-Oujda pour leur accueil chaleureux, leur bienveillance, et pour l’environnement de travail agréable et stimulant qu’ils ont su créer.

Enfin, j’adresse ma reconnaissance aux membres du jury, devant lesquels j’ai eu l’honneur de présenter ce travail, pour l’attention qu’ils y ont portée et pour l’évaluation qu’ils en ont faite.

# Tableau des abréviations

|  |  |
| --- | --- |
| Abréviation | Désignation |
| AD | AutoDistribution |
| API | Application Programming Interface |
| B2B | Business To Business |
| B2C | Business To Customer |
| BDD | Base De Données |
| DB | DataBase |
| ERP | Entreprise Resource Planning |
| GNU | GNU's Not Unix |
| SI | Système d’Information |
| HTML | Hyper Text Markup Language |
| JS | JavaScript |
| LTS | Long-Term Support |
| ORM | Object Relational Mapping |
| PHP | Hyper-Text Pre Processor |
| Redis | REmote DIctionary Server |
| WS | Web Service |

# *Résumé*

Ce rapport résume le travail réalisé dans le cadre de mon Projet de Fin d’Études (PFE) au sein de l’entreprise SQLI, portant sur l’étude et la préparation de la migration de l’application Autossimo, plateforme e-commerce B2B développée pour Autodistribution, actuellement basée sur Symfony 5.4 et PHP 7.3. L’objectif de ce projet est de préparer la transition vers Symfony 6.4, version LTS du framework, afin de répondre aux enjeux de sécurité, de performance, de compatibilité et de maintenabilité.

Pour cela, une approche progressive a été adoptée, comprenant trois étapes principales : la migration expérimentale d’un projet Symfony 5.4 minimal afin de valider le processus, l’intégration du fichier composer.json du projet Autossimo dans cet environnement pour analyser les dépendances réelles, puis l’analyse technique du code source afin d’identifier les dépréciations et anticiper les adaptations nécessaires.

Les résultats ont permis de détecter plusieurs dépendances obsolètes, packages abandonnés et interfaces dépréciées, nécessitant une adaptation technique pour assurer la compatibilité avec PHP 8.2 et Symfony 6.4. Ce travail m’a permis de mobiliser et renforcer des compétences en migration logicielle, audit de dépendances, Symfony, PHP, ainsi qu’en gestion technique et analyse structurée.

Bien que la mise en production ne fasse pas partie du périmètre de ce PFE, une feuille de route claire a été proposée pour encadrer les étapes restantes : traitement des interfaces dépréciées, remplacement des packages obsolètes, validation finale du code, et phase de tests de non-régression avant déploiement. Ce rapport constitue ainsi une base structurée pour conduire, dans des conditions maîtrisées, la modernisation technique de l’application Autossimo.

# *Abstract*

This report summarizes the work carried out as part of my end-of-study internship at SQLI, focused on the analysis and preparation of the migration of the Autossimo application—an e-commerce B2B platform developed for Autodistribution—from Symfony 5.4 and PHP 7.3 to Symfony 6.4, the latest long-term support (LTS) version. The objective of this project is to ensure the technical upgrade of the application to meet current requirements in terms of security, performance, compatibility, and maintainability.

A progressive approach was adopted, consisting of three main phases: an experimental migration of a minimal Symfony 5.4 project to validate the process; integration of Autossimo’s composer.json file to identify real dependencies; and an in-depth analysis of the source code to detect deprecated elements and anticipate necessary adjustments.

The results revealed several obsolete dependencies, abandoned packages, and deprecated interfaces requiring technical refactoring to ensure full compatibility with PHP 8.2 and Symfony 6.4. This project allowed me to apply and further develop key skills in software migration, dependency auditing, Symfony, PHP, and technical task management.

Although production deployment was not included in the project scope, a clear migration roadmap was defined to guide the next steps: addressing deprecated interfaces, replacing unsupported packages, conducting final validation, and performing regression testing. This report thus serves as a structured foundation for the future modernization of the Autossimo application in a secure and controlled environment.

# *ملخص*

يلخص هذا التقرير العمل المنجز في إطار مشروع التخرج (PFE) داخل شركة SQLI، والذي يهدف إلى دراسة وتحضير ترقية تطبيق Autossimo، وهو منصة تجارة إلكترونية بين الشركات (B2B) تم تطويرها لصالح Autodistribution، والمبني حاليًا باستخدام Symfony 5.4 و PHP 7.3. الهدف من هذا المشروع هو الانتقال إلى Symfony 6.4، النسخة طويلة الدعم (LTS)، لضمان الأمان، وتحسين الأداء، والحفاظ على قابلية الصيانة.

تم اتباع منهجية تدريجية تتكون من ثلاث مراحل رئيسية: أولًا، ترقية تجريبية لمشروع بسيط مبني على Symfony 5.4 للتحقق من خطوات الترقية؛ ثانيًا، إدماج ملف composer.json الخاص بتطبيق Autossimo لتحليل التبعيات؛ وثالثًا، تحليل تقني للكود المصدري لاكتشاف الميزات المهجورة وتقدير التعديلات المطلوبة.

أظهرت النتائج وجود العديد من الحزم المهجورة والتبعيات القديمة والواجهات البرمجية الموقوفة، مما يتطلب تعديلات تقنية لضمان التوافق مع PHP 8.2 وSymfony 6.4. وقد أتاح لي هذا المشروع تطوير مهارات متعددة في ترقية البرمجيات، تدقيق التبعيات، العمل باستخدام Symfony وPHP، وإدارة المهام التقنية.

رغم أن نشر التطبيق في بيئة الإنتاج لم يكن ضمن نطاق هذا المشروع، فقد تم إعداد خطة طريق واضحة توضح الخطوات التالية، مثل معالجة الواجهات المهجورة، واستبدال الحزم غير المدعومة، وتنفيذ اختبارات عدم الانحدار. يشكل هذا التقرير بذلك أساسًا منظمًا لترقية تطبيق Autossimo ضمن بيئة آمنة ومحكمة.

# Sommaire

[Dédicaces II](#_Toc201388191)

[*Remerciements* III](#_Toc201388192)

[Tableau des abréviations IV](#_Toc201388193)

[*Résumé* V](#_Toc201388194)

[*Abstract* VI](#_Toc201388195)

[*ملخص* VII](#_Toc201388196)

[Sommaire VIII](#_Toc201388197)

[Table des figures X](#_Toc201388198)

[*Introduction générale* 1](#_Toc201388199)

[CHAPITRE 1 : PRESENTATION DU PROEJT 2](#_Toc201388200)

[1.1. Introduction 3](#_Toc201388201)

[1.2. Présentation de l’entreprise d’accueil : 3](#_Toc201388202)

[1.2.1. Présentation du Groupe SQLI : 3](#_Toc201388203)

[1.3. Présentation du client Autodistribution: 5](#_Toc201388204)

[1.3.1. Présentation du projet « Autossimo v3 » 6](#_Toc201388209)

[1.3.2. Objectifs d’Autossimo 7](#_Toc201388210)

[1.4. Présentation du contexte du projet 7](#_Toc201388211)

[1.4.1. Etude de l’existant 7](#_Toc201388212)

[1.4.2. Identification des problématiques 8](#_Toc201388213)

[1.4.3. Solution proposée par le projet 9](#_Toc201388214)

[1.5. Planification et conduite du projet 9](#_Toc201388215)

[1.5.1. Méthodologie SCRUM 10](#_Toc201388216)

[1.5.2. Tableau Kandan 13](#_Toc201388217)

[1.5.3. Les outils utilisés 14](#_Toc201388218)

[1.6. Conclusion 15](#_Toc201388219)

[CHAPITRE 2 : ANALYSE ET ETUDE 16](#_Toc201388220)

[2.1. Introduction 17](#_Toc201388222)

[2.2. Embarquement 17](#_Toc201388223)

[2.2.1. Formation 17](#_Toc201388224)

[2.2.2. Présentation de projet 17](#_Toc201388225)

[2.2.3. Documents : Traveaux précédents 18](#_Toc201388226)

[2.3. L’environnement de développement 18](#_Toc201388230)

[2.3.1. Langages de programmation 19](#_Toc201388231)

[2.3.2. Bibliothèques et Frameworks 19](#_Toc201388232)

[2.3.3. Outils de développement 21](#_Toc201388233)

[2.3.4. Système de Gestion de base de données 23](#_Toc201388234)

[2.4. Le contrôle de version 23](#_Toc201388235)

[2.5. Le processus d’analyse 26](#_Toc201388236)

[2.5.1. Migration d’un projet Symfony 5.4 propre 26](#_Toc201388237)

[2.5.2. Migration de composer.json de projet Autossimo 36](#_Toc201388238)

[2.5.3. Analyse du code Autossimo 43](#_Toc201388239)

[2.6. Conclusion 63](#_Toc201388250)

[CHAPITRE 3 : STRATEGIE DE MIGRATION TECHNIQUE 64](#_Toc201388251)

[3.1. Introduction 65](#_Toc201388253)

[3.2. Types de migrations 65](#_Toc201388256)

[3.2.1. Migration monolithique (One-shot) 65](#_Toc201388257)

[3.2.2. Migration progressive / incrémentale 65](#_Toc201388258)

[3.2.3. Migration parallèle (Dual Maintenance) 66](#_Toc201388259)

[3.2.4. Pattern Strangler Fig (Étouffoir) 66](#_Toc201388260)

[3.2.5. Migration Blue-Green (ou basculement) 67](#_Toc201388261)

[3.2.6. Migration basée sur des Feature Toggles 67](#_Toc201388262)

[3.3. Le choix de migration 67](#_Toc201388263)

[3.4. Feuille de route 68](#_Toc201388264)

[3.4.1. Traitement des interfaces dépréciées 68](#_Toc201388270)

[3.4.2. Spike d’analyse des dépréciations 69](#_Toc201388271)

[3.4.3. Migration des packages obsolètes 69](#_Toc201388272)

[3.4.4. Résolution des problèmes post-migration 70](#_Toc201388273)

[3.4.5. TNR manuel de l’application 70](#_Toc201388274)

[3.5. Conclusion 70](#_Toc201388275)

[Conclusion générale 71](#_Toc201388276)

[WEBOGRAPHIE 73](#_Toc201388277)

# Table des figures

[Figure 1: Logo de SQLI 3](#_Toc201388105)

[Figure 2 : Répartition des agences 4](#_Toc201388106)

[Figure 3 : Partenaires de SQLI 4](#_Toc201388107)

[Figure 4: Organigramme de SQLI Maroc 5](#_Toc201388108)

[Figure 5: Logo Autodistribution 5](#_Toc201388109)

[Figure 6 : Logo Autossimo 6](#_Toc201388110)

[Figure 7 : Canaux de distribution 7](#_Toc201388111)

[Figure 8: Illustration simple d'architecture d'Autossimo 7](#_Toc201388112)

[Figure 9: Calendrier des sorties de Symfony (1) 8](#_Toc201388113)

[Figure 10: Calendrier des sorties de Symfony (2) 9](#_Toc201388114)

[Figure 11: Diagramme qui représent les équipes 10](#_Toc201388115)

[Figure 12: 3 piliers de Scrum 10](#_Toc201388116)

[Figure 13: Processus de Scrum 11](#_Toc201388117)

[Figure 14: Example de Burndown Chart au début de Sprint 13](#_Toc201388118)

[Figure 15: Example de Burndown Chart au milieu de Sprint 13](#_Toc201388119)

[Figure 16: Exemple de Tableau de Kanban de Autossimo 13](#_Toc201388120)

[Figure 17: Logo de Jira 14](#_Toc201388121)

[Figure 18: Logo de Microsoft Teams 14](#_Toc201388122)

[Figure 19: Logo de Slack 15](#_Toc201388123)

[Figure 20: Logo de Confluence 15](#_Toc201388124)

[Figure 21: Aperçu des projets de Autossimo v3 18](#_Toc201388125)

[Figure 22: Logo de PHP 19](#_Toc201388126)

[Figure 23: Logo de JS 19](#_Toc201388127)

[Figure 24: Logo de Symfony 19](#_Toc201388128)

[Figure 25: Logo de PHPUnit 20](#_Toc201388129)

[Figure 26: Logo de Rector 20](#_Toc201388130)

[Figure 27: Logo de Git 21](#_Toc201388131)

[Figure 28: Logo de Gitlab 21](#_Toc201388132)

[Figure 29: Logo de Composer 21](#_Toc201388133)

[Figure 30: Logo de Docker 22](#_Toc201388134)

[Figure 31: Logo de VSCode 22](#_Toc201388135)

[Figure 32: Logo GNU Make 22](#_Toc201388136)

[Figure 33: Logo de Oracle Database 23](#_Toc201388137)

[Figure 34: Logo de Redis 23](#_Toc201388138)

[Figure 35: Example d'un changement d'une version majeure 24](#_Toc201388139)

[Figure 36: Example d'un changement d'une version mineure 24](#_Toc201388140)

[Figure 37: Example d'un changement d'une version de correctif 24](#_Toc201388141)

[Figure 38: Oragnigramme du control de version 25](#_Toc201388142)

[Figure 39: Le symbole Caret (^) 25](#_Toc201388143)

[Figure 40: Le symbole Tilde (~) 25](#_Toc201388144)

[Figure 41: Structure du dossier .docker 26](#_Toc201388145)

[Figure 42: Comparaison en utilisant Rector contre sans l'utilisation de Rector 27](#_Toc201388146)

[Figure 43: Installation du Rector (1) 27](#_Toc201388147)

[Figure 44: Initialisation de "rector.php" 27](#_Toc201388148)

[Figure 45: Configuration de Rector 28](#_Toc201388149)

[Figure 46: Configuration des chemins de Rector 28](#_Toc201388150)

[Figure 47: Configuration des sets de Rector 29](#_Toc201388151)

[Figure 48: Configuration de PHP 8 sur Rector 29](#_Toc201388152)

[Figure 49: commit de "Rector resolu le conflit de RouteCollectionBuilder" 29](#_Toc201388153)

[Figure 50: Aperçu des dépréciations dans Symfony profiler 30](#_Toc201388154)

[Figure 51: Liste des commits concernant la résolution manuelle des dépréciations de configuration 31](#_Toc201388155)

[Figure 52: commit de "manuel: l'ajout de enable\_authenticator\_manager et remplacement de 'anonymous: lazy' to 'lazy: true' in config/packages/security.yaml" 31](#_Toc201388156)

[Figure 53: commit de "manuel: l'ajout de l'extension php-intl" 31](#_Toc201388157)

[Figure 54: commit de "manuel: l'ajout de 'storage\_factory\_id: session.storage.factory.native' au fichier config/packages/framework.yaml" 32](#_Toc201388158)

[Figure 55: commit de "manuel: l'ajout du ligne 'logging: true' au niveau de doctrine.dbal in config/packages/doctrine.yaml" 32](#_Toc201388159)

[Figure 56: commit de "Mise à jourée vers php 8" 32](#_Toc201388160)

[Figure 57: Liste des recettes qui nécessite une mise à jour 33](#_Toc201388161)

[Figure 58: Mise à jour de Symfony/console 33](#_Toc201388162)

[Figure 59: Liste des commits liés à la mise à jour des recettes 34](#_Toc201388163)

[Figure 60: Commit concernant l’ajout de symfony/runtime 34](#_Toc201388164)

[Figure 61: Commit de modification de composer.json pour la mise à jour de Symfony 35](#_Toc201388165)

[Figure 62: Exécution de "symfony update symfony/\*" 35](#_Toc201388166)

[Figure 63: Page d'accueil de Symfony 6.4 35](#_Toc201388167)

[Figure 64: commit de « composer install fonctionne bien » 36](#_Toc201388168)

[Figure 65: Détails de commit "Composer install fonctionne bien" 36](#_Toc201388169)

[Figure 66: Aperçu des dépréciations au niveau de Symfony Profiler 37](#_Toc201388170)

[Figure 67: Commits de mise à jour des recettes et l'ajout de PHP Intl extension 37](#_Toc201388171)

[Figure 68: Dépréciations après la mise à jour des recettes 37](#_Toc201388172)

[Figure 69: L'ajout de l'image de PHP 8.2 dans Docker 38](#_Toc201388173)

[Figure 70: Modification de la contrainte PHP dans `composer.json` 38](#_Toc201388174)

[Figure 71 : Erreur rencontrée lors de `composer update` 38](#_Toc201388175)

[Figure 72 : Détail du commit de mise à jour des packages 39](#_Toc201388176)

[Figure 73: Confirmation du passage à PHP 8.2 39](#_Toc201388177)

[Figure 74: Intégration de Rector dans les dépendances du projet 39](#_Toc201388178)

[Figure 75: Configuration de Rector 40](#_Toc201388179)

[Figure 76: Résultat de l’exécution de Rector 40](#_Toc201388180)

[Figure 77: Symfony profiler 41](#_Toc201388181)

[Figure 78: Erreur de composer update symfony/\* 41](#_Toc201388182)

[Figure 79: Erreur persistante malgré l’option -W 42](#_Toc201388183)

[Figure 80: Resultat de la commande "composer outdated" 43](#_Toc201388184)

[Figure 81: Code pour récupérer la session 44](#_Toc201388185)

[Figure 82: Dépréciations au niveau de PHPUnit 51](#_Toc201388186)

[Figure 83: L'exécution de la commande "docker compose up php -d --build" 54](#_Toc201388187)

[Figure 84: L'exécution de la commande "composer update" 55](#_Toc201388188)

[Figure 85: Erreur pendant la mise à jour de Symfony 59](#_Toc201388189)

[Figure 86: Sortie de la commande "compose recipes:update" 62](#_Toc201388190)

# *Introduction générale*

L’évolution accélérée des technologies numériques a profondément modifié les modes opératoires et les attentes des acteurs économiques à travers le monde. Dans ce contexte en constante mutation, les entreprises doivent intégrer des solutions informatiques robustes et évolutives pour rester compétitives, anticiper les besoins des utilisateurs et optimiser leurs processus métier. La transformation digitale constitue ainsi un levier stratégique essentiel, permettant d’assurer la continuité du service, d’améliorer l’expérience utilisateur et de garantir la pérennité des systèmes d’information.

Dans le domaine de l’après-vente automobile, cette dynamique se traduit par l’adoption croissante de plateformes digitales dédiées, capables de faciliter les échanges entre professionnels, d’accroître l’efficacité logistique et de renforcer la satisfaction client. Autodistribution, leader français dans la distribution indépendante de pièces détachées automobiles et poids lourds, s’inscrit pleinement dans cette démarche à travers sa solution e-commerce B2B Autossimo, conçue pour répondre aux exigences spécifiques de ses partenaires professionnels.

En collaboration avec SQLI, expert reconnu dans l’accompagnement à la transformation digitale des organisations, la plateforme Autossimo fait l’objet d’un processus continu de modernisation technologique. C’est dans ce cadre que s’inscrit le présent projet de fin d’études, mené au sein de l’agence SQLI-Oujda, et dont l’objectif principal est d’analyser en profondeur les enjeux et les modalités de la migration du projet Autossimo de Symfony 5.4 vers Symfony 6.4.

Cette étude vise à identifier les impacts techniques liés à cette mise à niveau, à évaluer les risques associés, et à proposer des bonnes pratiques de migration, tout en garantissant la stabilité fonctionnelle de l’application. Elle s’inscrit dans une démarche proactive visant à améliorer la performance, la sécurité et la maintenabilité du système, en adéquation avec les standards technologiques actuels.

# CHAPITRE 1 : PRESENTATION DU PROEJT

## Introduction

Ce premier chapitre vise à situer le cadre dans lequel s’est déroulé le stage. Il commence par une présentation de l’organisme d’accueil, SQLI GROUPE, ainsi que du client, le GROUPE AUTODISTRIBUTION. Ensuite, nous exposerons, le contexte de projet, les missions confiées, les objectifs fixés et la méthodologie de travail, avant de terminer par une vue d’ensemble sur la structure du présent rapport.

## Présentation de l’entreprise d’accueil :

### Présentation du Groupe SQLI :

Fondé en 1990, SQLI est un groupe européen qui accompagne les plus grandes marques internationales dans leur transformation digitale. SQLI met en œuvre et déploie des solutions digitales omnicanales (plateformes d’expérience, sites e-commerce, digital factories, digital workplaces, etc.) développant la notoriété, les offres et les ventes des entreprises, mais également leur performance interne.



Figure : Logo de SQLI

Elle s’appuie sur un réseau d’agences de proximité, lui garantissant une grande réactivité face aux attentes de ses clients. Grâce à son positionnement distinctif, elle est en mesure d’apporter des réponses globales aux enjeux de performance :

* Performance business : e-commerce, marketing digital & social, mobilité e- communication ...
* Performance de l’entreprise : entreprise collaborative, poste de travail, solutions métier, intégration de S.I ...

Ses +2400 collaborateurs sont répartis en France (Paris, Lyon, Toulouse, Bordeaux, Rouen et Nantes), en Suisse (Lausanne et Genève), au Luxembourg, en Belgique (Bruxelles), aux Pays-Bas et au Maroc (Oujda, Rabat et Casablanca).

Une image contenant capture d’écran, texte, carte, diagramme

Le contenu généré par l’IA peut être incorrect.

Figure  : Répartition des agences

Le groupe SQLI suit un rythme de développement élevé au niveau des projets, des ressources humaines, et chiffre d’affaires :

* 23 ans de projets et missions de haut niveau : SQLI assoit son développement sur une expertise technologique de pointe et sur sa politique intense de veille et R&D. En 2005, SQLI devient la 1ère SSII française à obtenir la certification CMMI niveau 3
* 26 partenaires, leaders des solutions technologiques : Le positionnement de SQLI à la conjonction du monde du digital et du système d’information de l’entreprise est sa valeur ajoutée. Le groupe a noué un réseau de partenaires pour adresser ses défis de façon la plus performante possible, et fournir le meilleur conseil, la meilleure solution



Figure : Partenaires de SQLI

#### SQLI Oujda :

Dans le cadre de sa stratégie de développement alliant taille et densité technologique, le Groupe SQLI s’est doté de pôles spécialisés, disposant chacun d’une expertise technologique et/ou métier spécifique. Pour accompagner le développement   
de son pôle dédié Open Source de Poitier, en septembre 2006, le Groupe SQLI ouvrait, en partenariat avec l’Université Mohammed Premier, le tout premier centre de R&D / offshore entièrement dédié aux technologies Open Source. Depuis, SQLI bénéficie, au sein même de l’Université, d’un fort potentiel de ressources de très haut niveau de qualité et de locaux entièrement équipés des technologies. En contrepartie, les experts du Groupe interviennent dans le cadre de formations spécifiques, telles que CMMI, et contribuent à l’enrichissement des cursus scolaires de modules complémentaires.

Concernant l’organisation du groupe SQLI Maroc, il s’agit d’une répartition selon l’organigramme suivant :

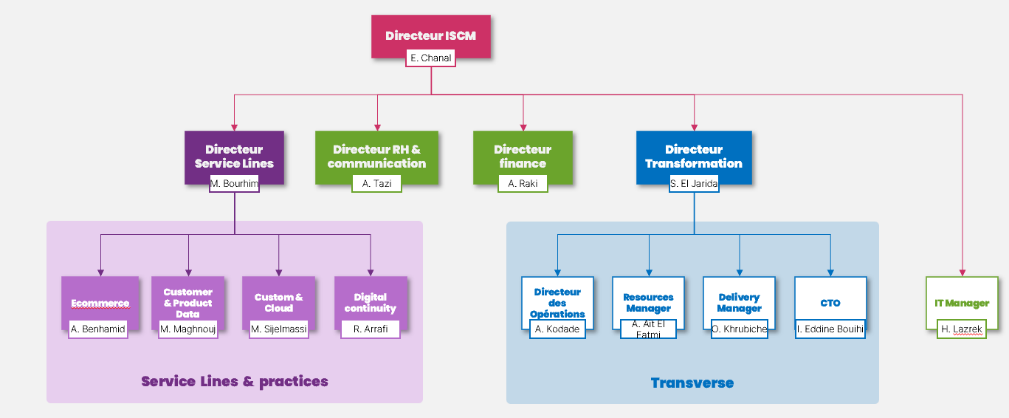


Figure : Organigramme de SQLI Maroc

## Présentation du client Autodistribution:



Figure : Logo Autodistribution

Crée en 1962, Autodistribution est membre d'AD International et fait partie d'Autodis Groupe, leader de la distribution indépendante de pièces détachées pour véhicules légers et lourds en Europe de l'Ouest.

Autodistribution est présent sur deux métiers : d'une part la distribution de pièces, de peintures, de pneumatiques d'équipements de garage et de services et d'autre part, la réparation multimarques. Il dispose d'une force de vente de plus de 650 commerciaux sur le terrain, d'un extranet de commande de pièces Autossimo pour les professionnels, d’un site « click & collect » Autodistribution.fr disponible 24h/24 et d'un service client de 200 personnes à l'écoute dans les centres d'appel régionaux.

Autossimo une plateforme e-commerce B2B permet aux 10 500 garagistes abonnés de commander quotidiennement des pièces détachées avec la garantie d’identifier le bon véhicule, la bonne pièce et de pouvoir être livré plusieurs fois par jour.



### Présentation du projet « Autossimo v3 »



Figure : Logo Autossimo

Pour la plupart des entreprises spécialisées dans la distribution de pièces détachées, la gestion de l'infrastructure informatique ne fait pas partie des compétences de base. Ces entreprises sont de véritables experts en distribution de produits et de pièces. Cela étant dit, le succès de nombreux distributeurs et revendeurs est directement lié à leur capacité à utiliser la technologie pour contrôler et équilibrer les stocks, réduire les coûts et minimiser les niveaux de stocks.

Les distributeurs qui ne disposent pas de la technologie appropriée signalent des stocks excédentaires ou des ruptures de stock fréquentes et coûteuses. Au fil du temps, ces inefficacités peuvent nuire aux bénéfices des entreprises et à la fidélité des clients finaux.

Cependant, ces deux éléments sont essentiels pour soutenir la croissance à long terme.

Le développement rapide de l'industrie des pièces automobiles rend une bonne gestion des stocks essentielle pour maintenir la compétitivité de l'industrie :

* **Trouver le juste équilibre entre les coûts de stockage minimum et les niveaux de service maximum :** les entreprises disposent de ressources limitées et doivent garantir des niveaux de stocks optimaux pour rester rentables.
* **Garantir une disponibilité maximale des pièces à forte demande :** les entreprises doivent réagir rapidement aux évolutions du marché pour répondre aux demandes des clients.
* **Automatisez le processus d'achat.**
* **Améliorer la visibilité et la responsabilité des fournisseurs :** le manque d'automatisation des systèmes ERP signifie que les délais de livraison des fournisseurs sont rarement pris en compte lors de la passation de commandes. La plupart des distributeurs ont des paramètres de commande automatiques pour acheter des pièces pendant une période spécifiée.

Une image contenant texte, capture d’écran, Police, diagramme

Le contenu généré par l’IA peut être incorrect.

Figure : Canaux de distribution

### Objectifs d’Autossimo

Portail complet et simple d’utilisation, il offre des fonctions adaptées aux besoins des réparateurs et carrossiers, en permettant l’accès :

* À un module de recherche par immatriculation, pour simplifier l’identification du véhicule.
* À un référentiel de pièces détachées en ligne multi-équipementiers et multimarque.
* Au carnet d’entretien et aux temps barémés constructeurs.
* À un module permettant de piloter les devis de réparation auto.
* À la visualisation des stocks distributeurs et aux commandes en ligne de pièces auto.
* À un service de multidiffusion des véhicules d’occasion (VO) et véhicules neufs (VN), en partenariat avec Caradisiac et La Centrale.

## Présentation du contexte du projet

### Etude de l’existant

Le projet Autossimo repose actuellement sur une architecture technique fondée sur PHP 7.3 et le framework Symfony dans sa version 5.4, qui constitue la dernière mise à jour LTS (Long Term Support) de cette branche. Cette solution utilise une base de données Oracle 19c, s’appuie sur Redis pour la gestion du cache, et intègre RabbitMQ comme système de transport asynchrone. L’ensemble est déployé au moyen de conteneurs Docker, assurant ainsi une certaine abstraction vis-à-vis de l’infrastructure.

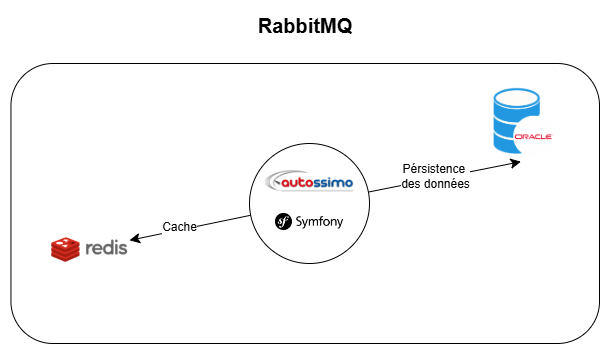


Figure : Illustration simple d'architecture d'Autossimo

* **Environnement technique actuel :**

L’application repose sur une architecture basée sur le framework Symfony 5.4, avec une structure modulaire organisée autour de différents services métiers, formulaires, contrôleurs et API REST sécurisées.

* Technologies principales :
  + Langage : PHP 7.3 (fin de vie en décembre 2021)
  + Framework : Symfony 5.4 (dernière version de la branche 5)
  + Gestion des dépendances : Composer 1.10
  + Base de données : Doctrine ORM
  + Gestion de cache en utilisant Redis
  + Interface utilisateur : Templates Twig dynamiques
  + Infrastructure : Déploiement via Docker

### Identification des problématiques

Le projet actuel repose sur la version 5.4 de Symfony, qui n’est plus maintenue depuis novembre 2024, ainsi que sur PHP 7.3, dont la fin de vie est intervenue en décembre 2021. Cette configuration technologique obsolète expose l’application à des risques accrus en termes de sécurité, de stabilité et de performance, en raison de l’absence de mises à jour et de correctifs. Par ailleurs, l’utilisation de versions dépassées limite l’accès aux fonctionnalités modernes et optimisées introduites dans PHP 8.1+ et Symfony 6.4. Une migration vers Symfony 6.4, dernière version LTS maintenue jusqu’en novembre 2025, permettrait de rétablir un environnement sécurisé et stable, tout en préparant le terrain pour une transition future vers Symfony 7. Cette mise à niveau représente une étape stratégique pour assurer la pérennité, l’évolutivité et la maintenabilité de la solution sur le long terme.

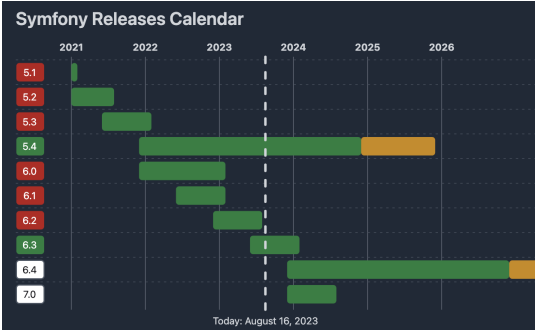


Figure : Calendrier des sorties de Symfony (1)

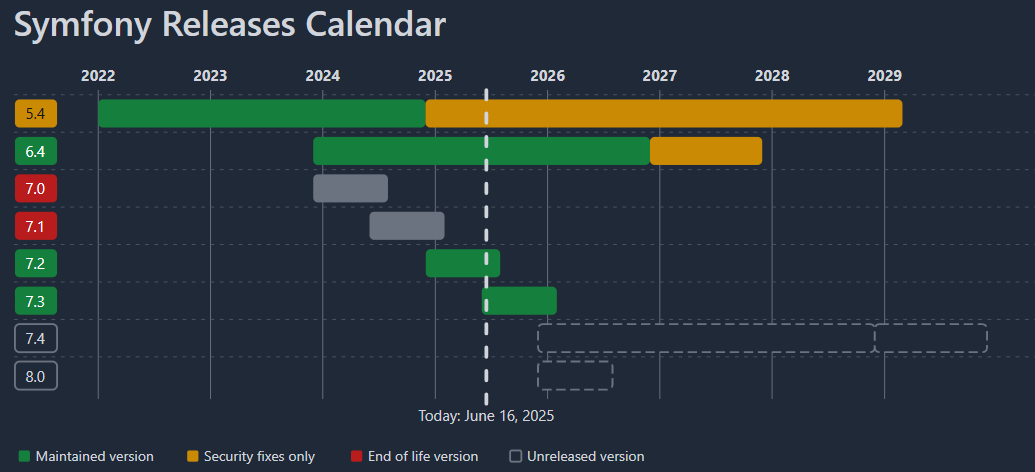


Figure : Calendrier des sorties de Symfony (2)

### Solution proposée par le projet

Par conséquent, la migration vers une version plus récente s’avère indispensable afin de garantir la fiabilité, la maintenabilité et la sécurité du projet Autossimo. Dans cette optique, mon Projet de Fin d’Études (PFE) a consisté à analyser en profondeur l’architecture existante du projet, à identifier les dépendances critiques et à étudier les impacts techniques liés à la mise à niveau. L’objectif principal était de définir une stratégie de migration structurée, permettant de passer vers une version plus récente du framework Symfony et de l’environnement PHP, tout en préservant la disponibilité du système, sans interruption des services en production et en minimisant les risques pour l’application métier.

## Planification et conduite du projet

La gestion de projet constitue une démarche structurée visant à organiser et piloter un projet de manière fluide, depuis son lancement jusqu’à sa finalisation. Lorsqu’elle englobe plusieurs projets interdépendants orientés vers un même objectif stratégique, elle prend la forme d’une gestion de programme. En pratique, le projet progresse vers sa cible finale tout en s’adaptant à d’éventuelles révisions, qui doivent néanmoins rester maîtrisées et planifiées. Il est ainsi essentiel de maintenir un équilibre constant entre les contraintes techniques, budgétaires et temporelles.

Dans le cadre de notre projet, deux exigences majeures nous ont été imposées : garantir la qualité du livrable et respecter les délais impartis. Pour répondre à ces contraintes, en cohérence avec les objectifs fixés et la nature de la mission, l’approche SCRUM a été retenue, enrichie par les bonnes pratiques issues de l’eXtreme Programming (XP). Cette méthodologie agile permet de découper le projet en sprints successifs, chacun correspondant à une itération au terme de laquelle une partie fonctionnelle du produit doit être développée et livrée.

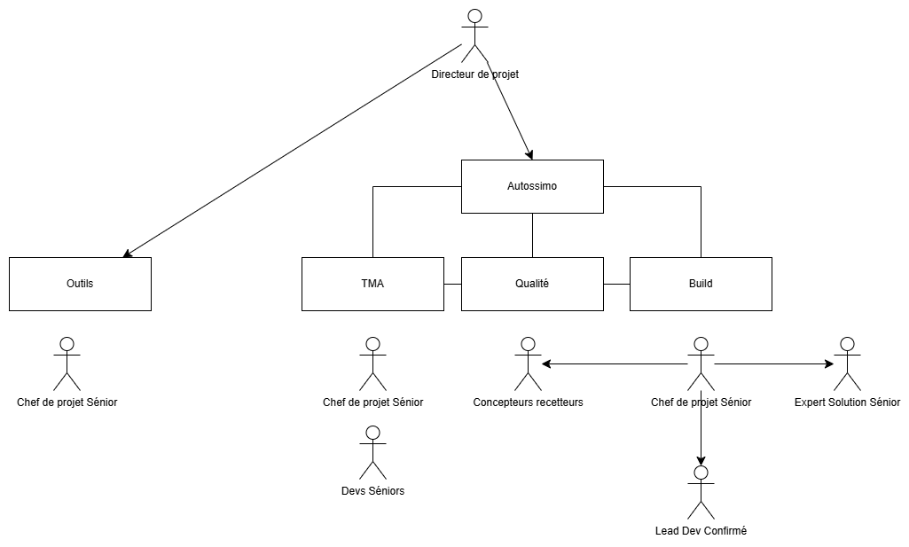


Figure : Diagramme qui représent les équipes

### Méthodologie SCRUM

#### Description de la méthodologie SCRUM

Scrum (mêlée du Rugby) est une méthodologie dédiée à la gestion de projet avec pour principal objectif l’optimisation de la productivité et qui est apparue pour la première fois en 1986 dans une publication de Hirotaka Takeuchi et Ikujiro Nonaka intitulée The New Product Development Game, qui s'appliquait alors au monde industriel.

Scrum est un cadre de travail permettant de répondre à des problèmes complexes et changeants, tout en livrant de manière productive et créative des produits de la plus grande valeur possible.

Scrum permet de travailler en équipe pour faire de l’amélioration continue sur des livraisons itératives incrémentales de produits afin de satisfaire vos clients.

Scrum est fondé sur la théorie du contrôle empirique de processus et soutenu par 3 piliers fondamentaux :

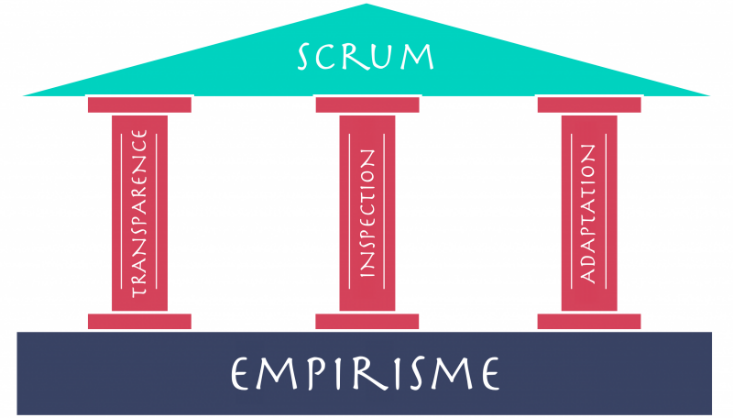


Figure : 3 piliers de Scrum

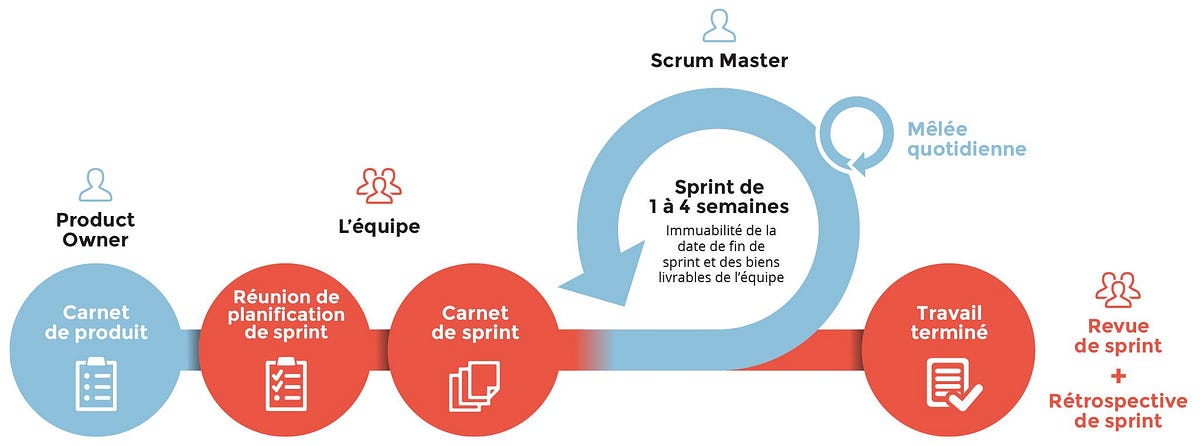


Figure : Processus de Scrum

* **Répartition des rôles :**

Dans le cadre de la mission réalisée avec SQLI pour accompagner AD dans le développement et l’évolution de l’application Autossimo, les équipes sont organisées selon les principes du framework Scrum. L'organisation projet repose sur une structure multidisciplinaire impliquant plusieurs pôles fonctionnels : TMA, Qualité, Build et Outils, chacun dirigé par un Chef de Projet Senior.

L’équipe Scrum est autoorganisée et multidisciplinaire, ce qui lui permet de choisir de manière autonome les méthodes de travail les plus efficaces. Les compétences nécessaires à la réalisation du projet sont réparties au sein des différents pôles, regroupant des profils tels que Dev Senior, Dev Confirmé, Architecte Junior, et Expert Solution Senior. Cette configuration optimise la flexibilité et la productivité collective sur Autossimo.

**Le Scrum Master**, souvent rattaché à la coordination inter-pôle (par exemple au sein de Build ou TMA), veille à l’application rigoureuse de la méthodologie Scrum. Il facilite les échanges, supprime les obstacles organisationnels et garantit la continuité dans l’exécution du backlog produit.

**Le Product Owner** est responsable de porter la vision métier d’AD concernant Autossimo. Il centralise les besoins utilisateurs, les priorise dans le Product Backlog, et valide les incréments livrés par les équipes. Ce rôle clé assure la cohérence entre les livrables et les attentes métier.

**L’équipe de développement** se compose de membres issus des pôles techniques mentionnés dans l’organigramme. Les développeurs, architectes et testeurs contribuent ensemble à la réalisation des fonctionnalités définies, dans un esprit collaboratif et sans hiérarchie stricte. Toutes les décisions relatives à l’implémentation sont prises de manière collective.

* **Les divers événements :**

La mise en œuvre de Scrum dans le projet Autossimo est rythmée par une série d’événements précis, garantissant l’organisation, la transparence et l’amélioration continue du travail au sein des équipes SQLI impliquées.

**Product Backlog :** Maintenu par le Product Owner, il regroupe et priorise l’ensemble des exigences d’AD concernant Autossimo. Ce document évolutif est continuellement mis à jour pour refléter les besoins métier en temps réel.

**Sprint Backlog :** À chaque début de Sprint, l’équipe de développement (Build, TMA, Outils) extrait du Product Backlog les éléments à traiter. Cette sélection constitue le Sprint Backlog, mis à jour quotidiennement pour suivre l’avancement des tâches.

**Planification du Sprint :** Durant cette réunion, le Product Owner et les développeurs se réunissent pour convenir des User Stories à intégrer dans le Sprint. L’équipe définit également l’approche technique à adopter, en s’appuyant sur les compétences transverses de chaque pôle.

**Sprint :** Chaque Sprint s'étale sur deux semaines, période durant laquelle les équipes livrent un incrément opérationnel. Ce découpage temporel permet une cadence soutenue et itérative dans les développements Autossimo.

**Mêlée quotidienne (Daily Scrum) :** Tous les matins, une réunion de 15 minutes a lieu pour faire un point rapide sur l’avancement. Elle réunit les membres des pôles concernés et permet une coordination efficace sur les tâches en cours.

**Revue de Sprint :** À la fin de chaque Sprint, les équipes (développement, Product Owner, QA) présentent les incréments réalisés. Cette démonstration est un moment clé pour valider les développements avec AD et collecter des retours.

**Rétrospective de Sprint :** Cette réunion vise à identifier les axes d’amélioration internes (communication, processus, qualité). Les membres de l’équipe Scrum y analysent le Sprint écoulé et définissent des actions concrètes à mettre en œuvre.

**Incrément :** Chaque Sprint aboutit à un incrément de produit potentiellement livrable. Cet incrément inclut les éléments terminés durant le Sprint actuel ainsi que ceux des Sprints précédents, assurant une continuité technique sur Autossimo.

**Burndown Chart :** Ce graphique quotidien, mis à jour par le Scrum Master, permet de visualiser la charge de travail restante par rapport au temps écoulé. Il constitue un outil de suivi essentiel pour évaluer la performance du Sprint en cours.

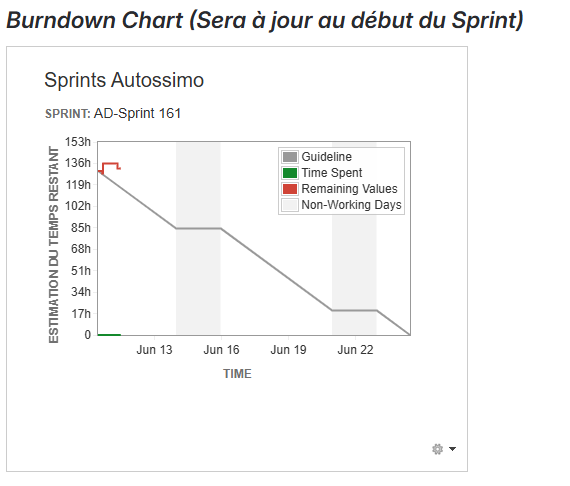


Figure : Example de Burndown Chart au début de Sprint

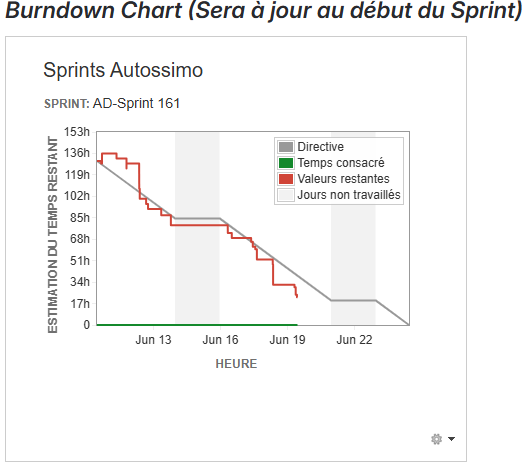


Figure : Example de Burndown Chart au milieu de Sprint

### Tableau Kandan

Un tableau Kanban est un outil de gestion de projet agile conçu pour visualiser le travail, limiter les travaux en cours et maximiser l'efficacité (ou le flux). Il peut aider les équipes agiles et DevOps à organiser leur travail quotidien. Les tableaux Kanban utilisent des cartes, des colonnes et une approche d'amélioration continue pour aider les équipes techniques et de service à s'engager sur la bonne quantité de travail et à la mener à bien !

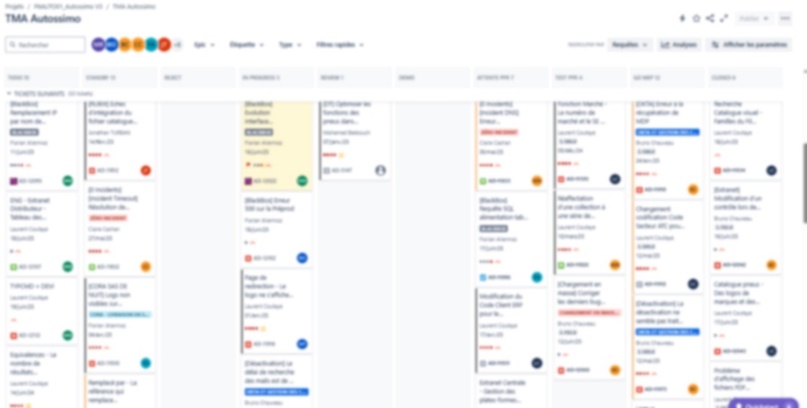


Figure : Exemple de Tableau de Kanban de Autossimo

### Les outils utilisés

#### Les outils de gestion du projet

Pour assurer un suivi rigoureux de l’avancement du projet et faciliter la répartition des tâches, des outils de gestion de projet sont utilisés. Ces solutions permettent de centraliser les activités, de suivre l’évolution des livrables, d’organiser les tâches par priorité, et de garantir une visibilité continue sur les responsabilités de chacun. Elles favorisent également la transparence au sein de l’équipe et permettent d’anticiper d’éventuels blocages tout en facilitant les échanges autour des jalons clés et des objectifs à atteindre. L’outils que nous utilisons c’est **Jira.**

* Jira :



Figure : Logo de Jira

Jira est un logiciel développé par Atlassian qui permet le suivi des bugs et des problèmes, ainsi que la gestion agile de projets. Jira est utilisé par un grand nombre de clients et d'utilisateurs dans le monde entier pour la gestion de projets, de délais, d'exigences, de tâches, de bugs, de modifications, de code, de tests, de versions et de sprints.

#### Les outils de communication

En raison de la répartition géographique des membres de l’équipe, l’usage d’outils de communication s’avère indispensable pour assurer la coordination et la continuité des échanges. Dans certaines situations, un membre de l’équipe peut être amené à travailler à distance sans que cela n'affecte le bon déroulement du projet, grâce à la possibilité de participer aux réunions, partager des informations et assurer ses tâches via des outils de messagerie ou de visioconférence. Étant donné que l’entreprise opère dans le domaine du développement web, le travail à distance est pleinement intégré dans son mode de fonctionnement. Les deux majeurs outils utilisés sont **Microsoft Teams** et **Slack**.

* Microsoft Teams :



Figure : Logo de Microsoft Teams

Microsoft Teams est une plateforme de collaboration d'équipe développée par Microsoft dans le cadre de la suite Microsoft 365. Elle offre des fonctionnalités telles que le chat en espace de travail, la visioconférence, le stockage de fichiers et l'intégration avec les applications et services Microsoft et tiers. Teams a progressivement remplacé les anciennes plateformes de messagerie et de collaboration Microsoft, notamment Skype Entreprise, Skype et Microsoft Classroom.

* Slack :



Figure : Logo de Slack

Slack est une plateforme cloud de communication d'équipe développée par Slack Technologies, propriété de Salesforce depuis 2020. Slack utilise un modèle freemium. Slack est principalement proposé comme un service interentreprises, sa base d'utilisateurs étant principalement composée d'entreprises travaillant en équipe, tandis que ses fonctionnalités sont principalement axées sur la gestion et la communication d'entreprise.

* Confluence :

Une image contenant Graphique, Police, graphisme, typographie

Le contenu généré par l’IA peut être incorrect.

Figure : Logo de Confluence

Confluence est un espace de travail numérique créé par Atlassian. Les équipes peuvent créer, organiser et collaborer sur divers types de contenus, notamment des documents, des comptes rendus de réunion, des plans de projet, etc. Fondamentalement, Confluence permet aux équipes de collaborer et de partager leurs connaissances, éliminant ainsi la dispersion des documents et les échanges de courriels.

## Conclusion

Le projet Autossimo, développé par SQLI pour le compte d’Autodistribution, constitue une plateforme e-commerce B2B essentielle pour la distribution de pièces automobiles. Actuellement basé sur PHP 7.3 et Symfony 5.4 — des technologies obsolètes — le système présente des risques en termes de sécurité, de performance et de maintenabilité. Dans ce contexte, mon PFE a porté sur l’analyse approfondie en vue de migrer l’application vers Symfony 6.4, dernière version LTS, afin de garantir sa pérennité, améliorer ses performances et préparer sa future évolution. Cette étude s’inscrit dans une démarche de modernisation stratégique, au sein d’une équipe utilisant la méthode Scrum et des outils agiles pour assurer la qualité et la continuité du projet.

# CHAPITRE 2 : ANALYSE ET ETUDE



## Introduction

Cette section a pour objectif de présenter les étapes préliminaires nécessaires à la migration du projet Autossimo, développé sous Symfony 5.4, vers la version 6.4 du framework. Elle vise à décrire en détail les étapes réalisées pour identifier les éléments clés qui influencent cette mise à jour, notamment les dépendances obsolètes, les composants incompatibles, ainsi que les bonnes pratiques à adopter.

## Embarquement

### Formation

Dans le cadre de mon stage de fin d’études au sein de l’entreprise **SQLI Oujda**, j’ai eu l’opportunité de suivre une **formation d’intégration dédiée aux nouveaux E-Challengers**. Cette formation, d’une durée d’un mois, était de nature générale et visait à nous familiariser avec les **bonnes pratiques de développement logiciel**. Animée par des experts de l’entreprise, elle couvrait plusieurs concepts fondamentaux tels que les **principes SOLID**, les **design patterns**, ainsi qu’une introduction aux technologies **Front-End**, notamment **Angular**.

Lors du **deuxième mois**, une spécialisation a été mise en place en fonction du profil de chaque stagiaire. En tant que développeur PHP, j’ai bénéficié d’un approfondissement dans le framework **Symfony**, encadré par deux experts techniques de SQLI : **Monsieur Issam Khadiri**, mon encadrant pédagogique, et **Monsieur Mohamed Rahmouni**, mon encadrant technique.  
Cette formation ciblée nous a permis d’acquérir des compétences concrètes et directement applicables dans les projets réels auxquels nous allions être affectés par la suite.

### Présentation de projet

Ce projet s’inscrit dans le cadre de la modernisation d’une application métier Autossimo. L’objectif principal est de migrer l’application Autossimo de la version 5.4 à la version 6.4 du framework Symfony, tout en passant de PHP 7.3 à PHP 8.2. Cette mise à jour vise à garantir la pérennité du système, améliorer sa sécurité, sa performance, ainsi que sa compatibilité avec les standards technologiques actuels.

### Documents : Traveaux précédents

À mon arrivée, j’ai pu m’appuyer sur plusieurs éléments déjà disponibles, facilitant la compréhension de l’existant et la préparation de la migration. Parmi ces éléments figurent :

* Un document décrivant les résultats d’un test de compatibilité avec PHP 8.2
* Un audit de code réalisé par SensioLabs, fournissant des recommandations précises sur la qualité et la robustesse du code
* Plusieurs documents de référence retraçant des migrations antérieures ayant été menées à bien

Ces ressources constituent une base d’analyse solide pour orienter les choix techniques et anticiper les impacts liés à la montée de version.

Le Projet Autossimo v3, contient plusieurs sous-projets, autodistribution-v3 c’est le projet où j’ai effectué mon travail.

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Figure : Aperçu des projets de Autossimo v3



## L’environnement de développement

Ci-dessous les outils rencontrés lors de la réalisation des tâches :

### Langages de programmation

#### PHP



Figure : Logo de PHP

PHP (acronyme récursif de PHP : Hypertext Preprocessor) est un langage de script open source à usage général largement utilisé, particulièrement adapté au développement Web et pouvant être intégré au HTML.

La version utilisée dans le projet est : PHP 7.3.2

#### JavaScript (JS)



Figure : Logo de JS

JavaScript (JS) est un langage de programmation interprété léger (ou compilé juste-à-temps) doté de fonctions de premier ordre. Bien qu'il soit surtout connu comme langage de script pour les pages web, de nombreux environnements hors navigateur l'utilisent également, tels que Node.js, Apache CouchDB et Adobe Acrobat. JavaScript est un langage dynamique, basé sur des prototypes, multi-paradigmes et monothread, prenant en charge les styles orientés objet, impératif et déclaratif (par exemple, la programmation fonctionnelle).

### Bibliothèques et Frameworks

#### Symfony



Figure : Logo de Symfony

Symfony est un Framework qui représente un ensemble de composants (aussi appelés librairies) PHP autonomes qui peuvent être utilisés dans des projets web privé ou open source. Mais c’est également un puissant Framework PHP développé par une société française : SensioLabs. Il permet de réaliser des sites internet dynamiques de manière rapide, structurée, et avec un développement clair. Les développeurs peuvent travailler sur ce Framework très facilement, seuls ou en équipe, grâce à la facilité de prise en main.

La version Symfony actuelle du projet est : 5.4

#### PHPUnit



Figure : Logo de PHPUnit

PHPUnit est un framework de tests unitaires pour le langage de programmation PHP. Il s'agit d'une instance de l'architecture xUnit pour les frameworks de tests unitaires, issue de SUnit et popularisée par JUnit. PHPUnit a été créé par Sebastian Bergmann et son développement est hébergé sur GitHub.

#### Rector



Figure : Logo de Rector

Rector est un outil PHP que vous pouvez exécuter sur n'importe quel projet PHP pour une mise à niveau instantanée ou une refactorisation automatisée.

Il vous aide à :

* Mises à niveau PHP et framework,
* Migrations de frameworks internes,
* Amélioration de la qualité de votre code pour proposer des fonctionnalités plus rapidement que la concurrence.

### Outils de développement

#### Git

Une image contenant capture d’écran, Graphique, dessin humoristique, créativité

Le contenu généré par l’IA peut être incorrect.

Figure : Logo de Git

Git est un système de gestion de versions distribué, gratuit et open source, conçu pour gérer tous types de projets, des plus petits aux plus grands, avec rapidité et efficacité.

Git est facile à prendre en main et offre un encombrement minimal, tout en offrant des performances ultra-rapides. Il surpasse les outils de gestion de versions comme Subversion, CVS, Perforce et ClearCase grâce à des fonctionnalités telles que la création de branches locales économiques, des zones de staging pratiques et des workflows multiples.

#### Gitlab



Figure : Logo de Gitlab

GitLab est une plateforme de développement logiciel open source complète avec contrôle de version intégré, suivi des incidents, revue de code, CI/CD, et bien plus encore. Hébergez GitLab sur vos propres serveurs, dans un conteneur ou chez un fournisseur cloud.

#### Composer



Figure : Logo de Composer

Composer est un outil de gestion des dépendances en PHP. Il vous permet de déclarer les bibliothèques dont dépend votre projet et de les gérer (installer/mettre à jour) pour vous.

#### Docker

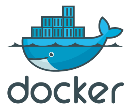


Figure : Logo de Docker

Docker est une plateforme ouverte pour le développement, la livraison et l'exécution d'applications. Docker vous permet de séparer vos applications de votre infrastructure afin de livrer vos logiciels rapidement. Avec Docker, vous gérez votre infrastructure de la même manière que vous gérez vos applications. En tirant parti des méthodologies Docker pour la livraison, les tests et le déploiement du code, vous pouvez réduire considérablement le délai entre l'écriture du code et son exécution en production.

#### VSCode

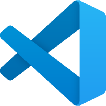


Figure : Logo de VSCode

Visual Studio Code est un éditeur de code extensible développé par Microsoft pour Windows, Linux et macOS

Les fonctionnalités incluent la prise en charge du débogage, la mise en évidence de la syntaxe, la complétion intelligente du code (IntelliSense), les snippets, la refactorisation du code et Git intégrer. Les utilisateurs peuvent modifier le thème, les raccourcis clavier, les préférences et installer des extensions qui ajoutent des fonctionnalités supplémentaires.

#### GNU Make



Figure : Logo GNU Make

GNU Make est un outil qui contrôle la génération d'exécutables et d'autres fichiers non-sources d'un programme à partir des fichiers sources du programme.

### Système de Gestion de base de données

#### Oracle base de données



Figure : Logo de Oracle Database

Oracle Database est un système de gestion de base de données relationnelle (SGBDR) qui depuis l'introduction du support du modèle objet dans sa version 8 peut être aussi qualifié de système de gestion de base de données relationnel-objet (SGBDRO). Fourni par Oracle Corporation, il a été développé par Larry Ellison, accompagné entre autres, de Bob Miner et Ed Oates.

#### Redis



Figure : Logo de Redis

Redis (de l'anglais REmote DIctionary Server qui peut être traduit par « serveur de dictionnaire distant » et jeu de mots avec Redistribute) est un système de gestion de base de données clé-valeur extensible, très hautes performances, écrit en C ANSI. Il fait partie de la mouvance NoSQL et vise à fournir les performances les plus élevées possible.

## Le contrôle de version

Avant d’aborder les étapes détaillées de la migration, il convient de rappeler les fondamentaux des numéros de version et de clarifier la signification de chacun des éléments qui les composent. Une bonne compréhension de ces notions est essentielle pour anticiper les impacts techniques liés aux évolutions majeures, notamment lors de la mise à jour d’un framework tel que Symfony.

Le contrôle de version, également appelé gestion des versions, consiste à attribuer des identifiants uniques, souvent numériques ou alphanumériques, aux différentes versions d'applications logicielles.

* **SemVer (Versionnage Sémantique) :**

Format : Majeur.Mineur.Correctif (3.9.0)

SemVer comprend trois éléments distincts : Majeur, Mineur et Correctif.

#### Version majeure :

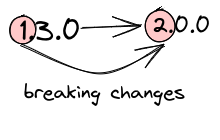


Figure : Example d'un changement d'une version majeure

Un changement de version majeure constitue une mise à jour significative du logiciel, introduisant souvent des fonctionnalités majeures et des évolutions structurelles importantes. Toutefois, ce type de mise à niveau peut également entraîner des ruptures de compatibilité avec les versions précédentes. Il est donc essentiel de procéder avec prudence lors de la migration vers une version majeure, afin de garantir la compatibilité avec le code existant et d’éviter d’éventuels dysfonctionnements.

#### Version mineure :

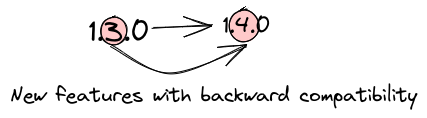


Figure : Example d'un changement d'une version mineure

Une version mineure correspond généralement à l’ajout de nouvelles fonctionnalités ou d’améliorations tout en conservant la compatibilité ascendante avec la version précédente. Cela signifie que les applications développées avec une version mineure antérieure peuvent en général fonctionner avec la nouvelle version sans modifications majeures ni régressions notables.

#### Version de correctif :

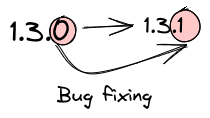


Figure : Example d'un changement d'une version de correctif

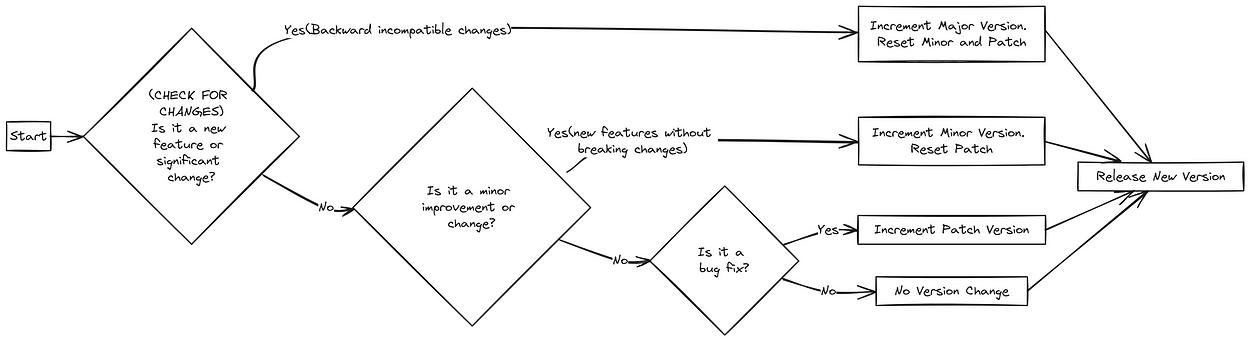
Une version de correctif, également appelée version de maintenance, désigne des mises à jour mineures apportant principalement des corrections de bugs et des améliorations ponctuelles. Ce type de mise à jour vise à résoudre des problèmes spécifiques sans modifier de manière significative le fonctionnement global du logic

Figure : Oragnigramme du control de version

#### Les symboles caret (^) et Tilde (~)

* **Le symbole Caret (^) :**

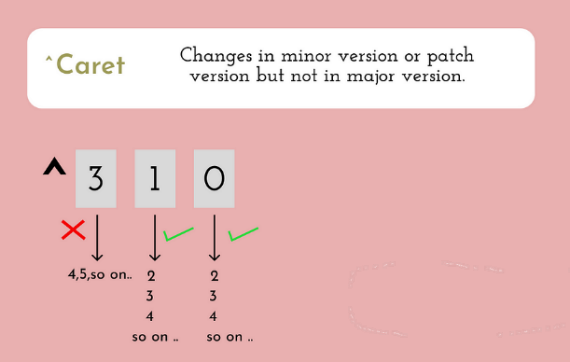


Figure : Le symbole Caret (^)

Le symbole caret dans un numéro de version (par exemple, ^3.2.1) guide votre logiciel pour qu'il conserve la même version majeure, tout en mettant à jour les versions mineures et les correctifs. Cela signifie que vous pouvez bénéficier de nouvelles fonctionnalités et de corrections de bugs sans craindre une transformation majeure de votre logiciel.

* **Le symbole Tilde (~) :**

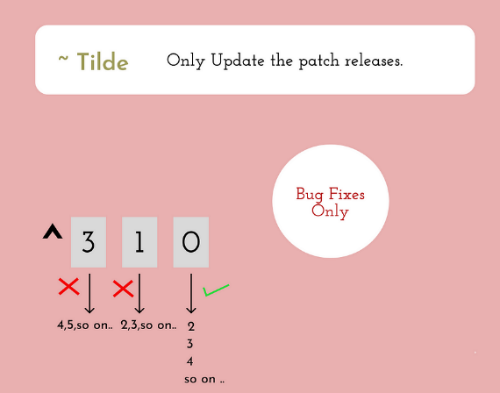


Figure : Le symbole Tilde (~)

Le symbole tilde (par exemple, ~3.2.1) vous permet de mettre à jour la version du correctif. Il vous garantit des corrections de bugs et des améliorations mineures sans introduire de modifications majeures susceptibles de poser des problèmes.

## Le processus d’analyse

### Migration d’un projet Symfony 5.4 propre

Dans le cadre de la préparation à la migration de Symfony 5.4 vers Symfony 6.4, une phase exploratoire a été engagée à partir d’un projet simplifié. Cette démarche visait à identifier les principales étapes du processus de migration en dehors du contexte métier, dans un environnement technique maîtrisé.

Un environnement de développement a été mis en place à l’aide de Docker Compose, permettant d’orchestrer les conteneurs nécessaires au fonctionnement de l’application, notamment pour les services PHP, base de données et Redis. Un projet Symfony vide a été initialisé (version 5.4, avec PHP 8.2), dans lequel deux entités simples – Author et Book – ont été créées, chacune accompagnée de fonctionnalités CRUD générées à l’aide du Maker Bundle.

Ce projet léger a servi de base d’expérimentation pour identifier les dépréciations liées à Symfony 5.4, en mesurer l’impact, et valider les correctifs nécessaires avant d’appliquer les changements à l’application métier Autossimo. L’utilisation de Docker Compose a permis de simuler un environnement proche de celui de la production tout en assurant un isolement des modifications. Cette approche a facilité la gestion des dépendances techniques et assuré une phase de test sans effet sur le projet principal.

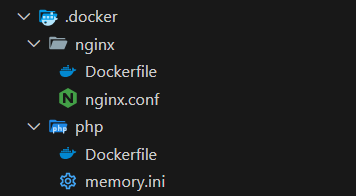


Figure : Structure du dossier .docker

#### Rector

Rector, comme indiqué sur leur site web officiel : « Rector est un outil PHP que vous pouvez exécuter sur n'importe quel projet PHP pour une mise à niveau instantanée ou une refactorisation automatisée. » Rector permet donc d'automatiser la refactorisation et d'écrire du code moderne, par opposition au code « legacy », et d'en améliorer la qualité.

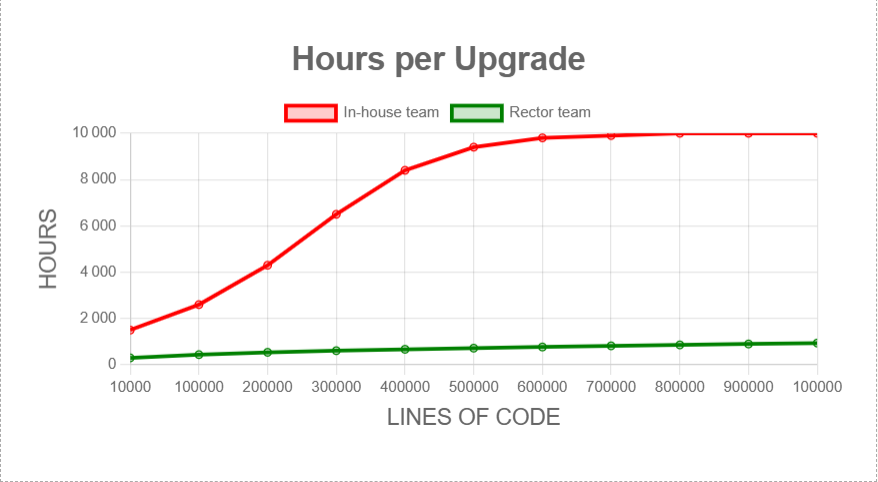


Figure : Comparaison en utilisant Rector contre sans l'utilisation de Rector

Dans le cadre de cette expérimentation, Rector a été installé sur une branche dédiée. L’installation a été réalisée via la commande suivante, conformément à la documentation officielle: composer require --dev rector/rector

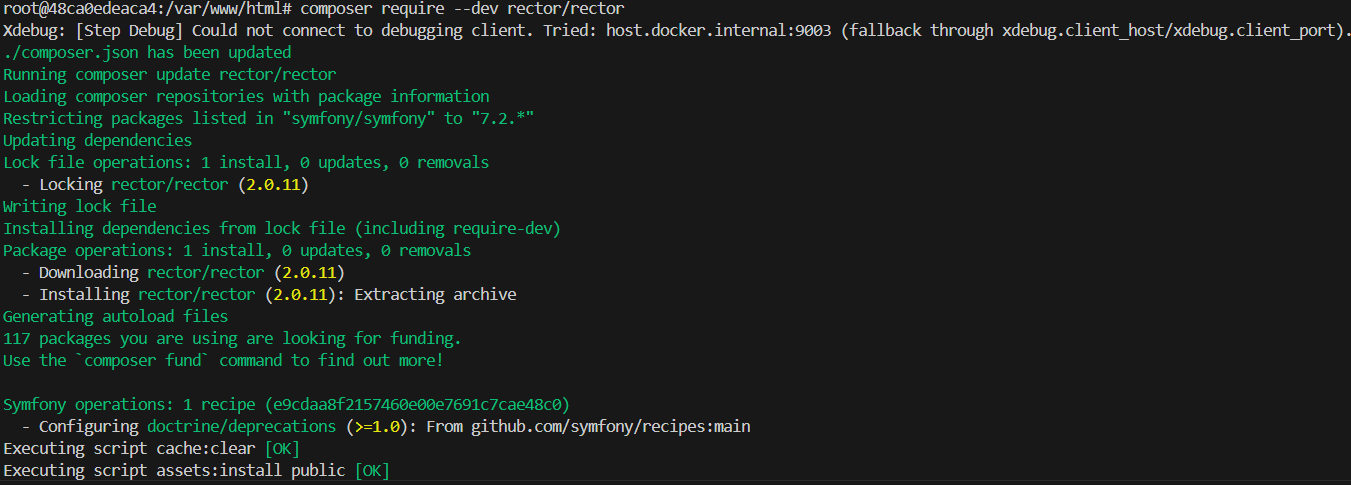


Figure : Installation du Rector (1)

Le fichier de configuration a été initialisé en exécutant la commande suivante : ./vendor/bin/rector.

Lors de l’exécution, l’outil a proposé de générer un nouveau fichier rector.php, ce qui a été accepté afin de configurer les règles de refactorisation à appliquer durant la phase de migration.

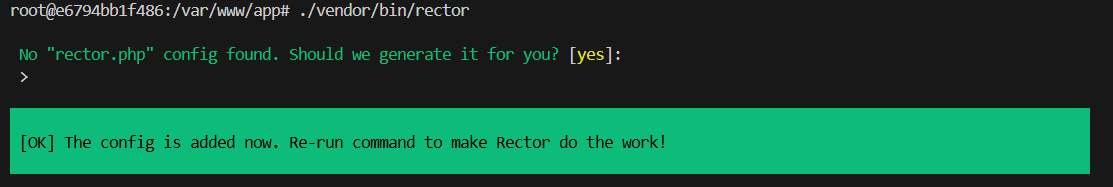


Figure : Initialisation de "rector.php"

Après consultation de la documentation officielle et d’articles spécialisés, une configuration adaptée de Rector a été identifiée pour répondre aux exigences du projet.



Figure : Configuration de Rector

**Configuration des Chemins :**

La configuration spécifie les chemins à analyser et à refactoriser automatiquement :

* Le répertoire src/, contenant le code source principal de l’application ;
* Le répertoire tests/, dédié aux tests unitaires.

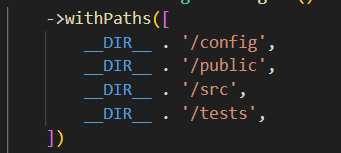


Figure : Configuration des chemins de Rector

Cette configuration permet d’appliquer systématiquement les règles de migration et de modernisation sur l’ensemble du code fonctionnel et des tests.

**Activation des Ensembles de Règles Symfony**

Les sets Symfony ont été activés afin de permettre une migration progressive entre les différentes versions du framework. Chaque ensemble applique un groupe de règles propres à une version donnée de Symfony, allant ici de la version 5.0 jusqu’à 5.4.

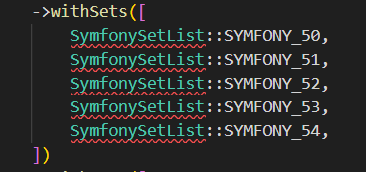


Figure : Configuration des sets de Rector

**Activation des ensembles de règles PHP**

Des ensembles spécifiques à PHP 8.1 ont également été activés, permettant d’aligner le code avec les bonnes pratiques et les exigences de cette version du langage.

Une image contenant texte, Police, Graphique, capture d’écran

Le contenu généré par l’IA peut être incorrect.

Figure : Configuration de PHP 8 sur Rector

**Résultat de l’exécution**

Après configuration, l’outil Rector a été exécuté à l’aide de la commande suivante :

./vendor/bin/rector process

L’analyse et la refactorisation automatiques ont généré plusieurs modifications, dont notamment le remplacement du composant RouteCollectionBuilder, désormais obsolète, par le composant recommandé RoutingConfigurator.



Figure : commit de "Rector resolu le conflit de RouteCollectionBuilder"

#### Dépréciations

##### Qu’est-ce qu’une dépréciation en programmation ?

En développement logiciel, une dépréciation (ou deprecation) désigne l’utilisation de composants, méthodes ou fonctions qui ne sont plus recommandés, souvent remplacés par des alternatives plus modernes ou plus sécurisées. Ces éléments ne sont pas supprimés immédiatement pour éviter les régressions, mais leur usage est fortement déconseillé.

##### Comment identifier les dépréciations dans le projet ?

Dans le contexte de Symfony, les dépréciations sont souvent signalées via :

* Le web profiler de Symfony (en mode dev)
* Les sorties de tests unitaires
* L’outil *symfony-deprecation-detector*

Pour détecter les dépréciations dans l’application, j’ai utilisé le mode développeur de Symfony (APP\_ENV=dev) et inspecté le toolbar du Web Profiler.

Certaines dépréciations fournissent des instructions claires sur la manière de les corriger. D’autres nécessitent une analyse approfondie du code source ou une recherche documentaire pour comprendre comment les remplacer.

##### Suppression des dépréciations

La gestion du code obsolète constitue une étape critique dans tout processus de migration. Elle permet de garantir la compatibilité du projet avec les versions futures du framework tout en assurant une meilleure maintenabilité du code.

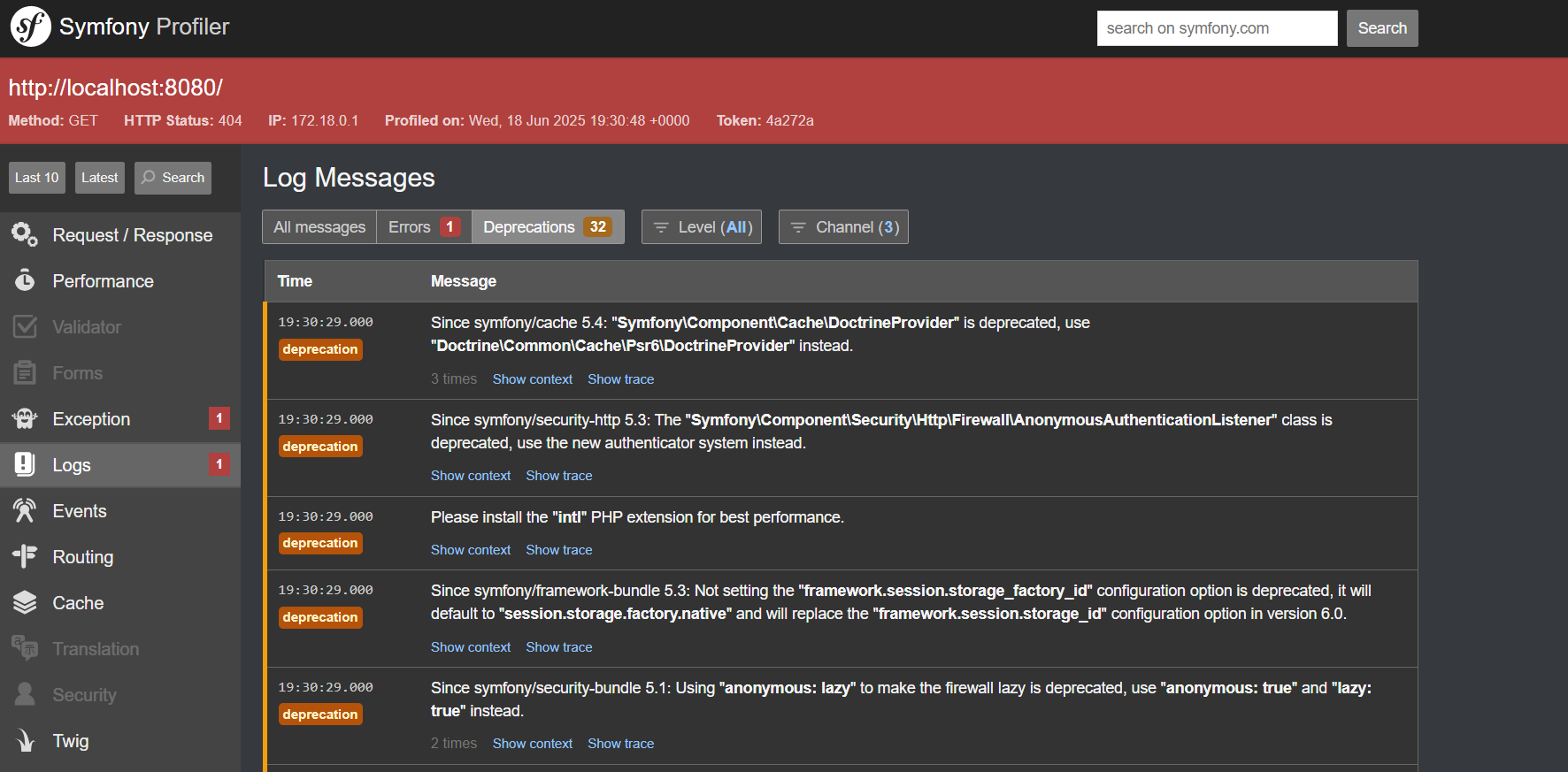


Figure : Aperçu des dépréciations dans Symfony profiler

L’analyse initiale a révélé que la majorité des dépréciations concernait la configuration, et non la logique applicative, en raison du faible volume de code métier présent dans le projet de test. Un ensemble d’actions correctives a donc été entrepris pour résoudre manuellement ces dépréciations à travers plusieurs étapes, dont les principaux sont listés ci-dessous :

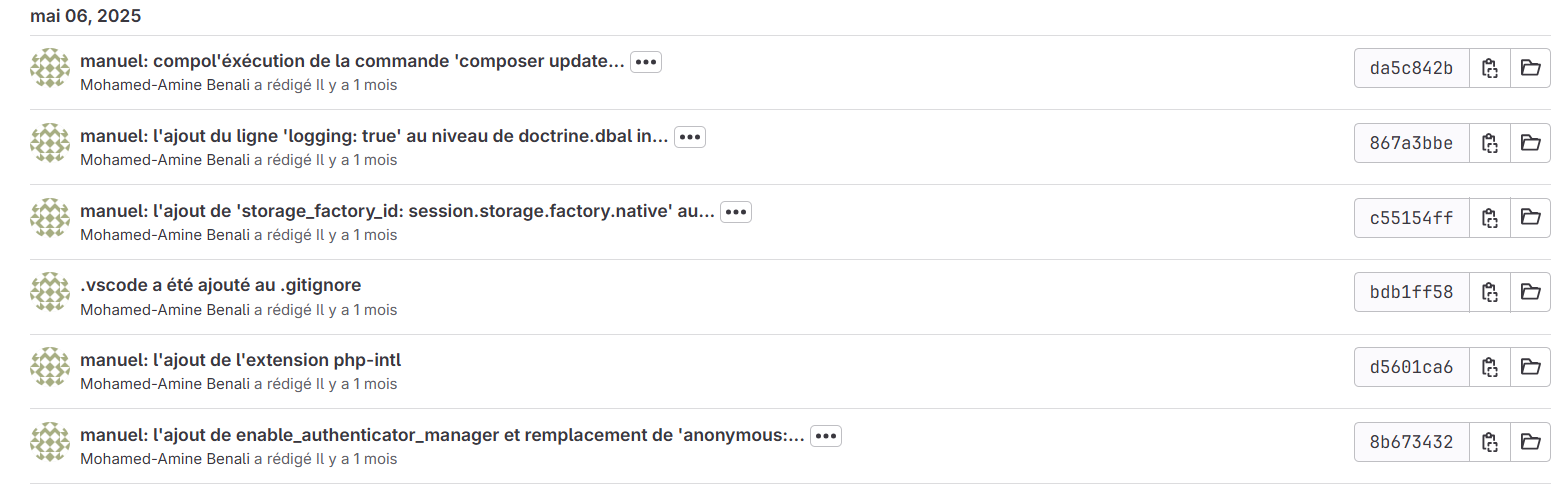


Figure : Liste des commits concernant la résolution manuelle des dépréciations de configuration

**Explication :**

* Ajout de la directive enable\_authenticator\_manager: true et remplacement de anonymous: true par lazy: true dans config/packages/security.yaml.

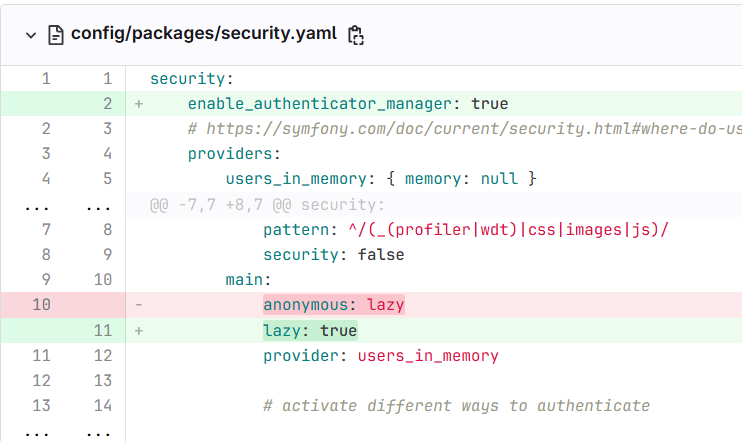


Figure : commit de "manuel: l'ajout de enable\_authenticator\_manager et remplacement de 'anonymous: lazy' to 'lazy: true' in config/packages/security.yaml"

* Intégration de l’extension PHP intl dans l’environnement Docker.



Figure : commit de "manuel: l'ajout de l'extension php-intl"

* Ajout de storage\_factory\_id: "session.storage.factory.native" dans config/packages/framework.yaml.

Une image contenant texte, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.

Figure : commit de "manuel: l'ajout de 'storage\_factory\_id: session.storage.factory.native' au fichier config/packages/framework.yaml"

* Activation du journal de requêtes Doctrine via logging: true dans config/packages/doctrine.yaml.

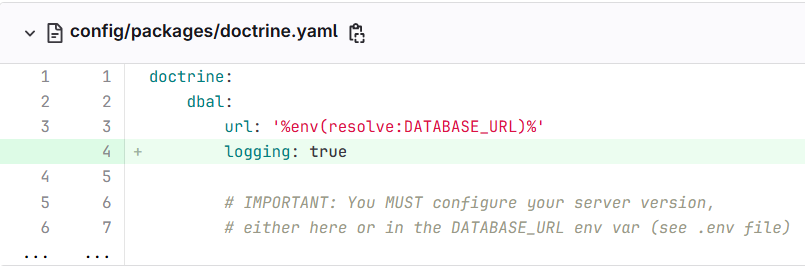


Figure : commit de "manuel: l'ajout du ligne 'logging: true' au niveau de doctrine.dbal in config/packages/doctrine.yaml"

* Migration de l’environnement vers PHP 8 en modifiant le fichier Docker.



Figure : commit de "Mise à jourée vers php 8"

#### Mise à niveau des Recettes

Les recettes Symfony Flex sont des scripts d’installation et de configuration automatique. Elles évoluent avec le temps et nécessitent également d’être mises à jour afin de garantir la cohérence avec les versions récentes du framework.

La commande suivante a été utilisée pour lancer la mise à jour des recettes :

composer recipes:update

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

Figure : Liste des recettes qui nécessite une mise à jour

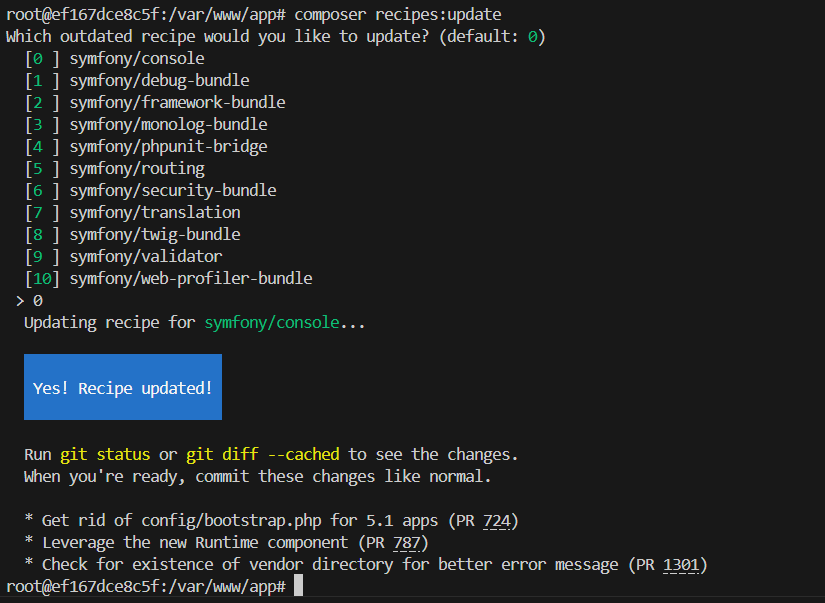


Figure : Mise à jour de Symfony/console

L’ensemble des recettes a été mis à jour progressivement. Toutefois, une erreur d’exécution est survenue après la mise à jour. L’analyse du message d’erreur a indiqué l’absence du package symfony/runtime, nécessaire à l’exécution du projet après mise à jour. Celui-ci a été installé avec la commande :

composer require symfony/runtime



Figure : Liste des commits liés à la mise à jour des recettes

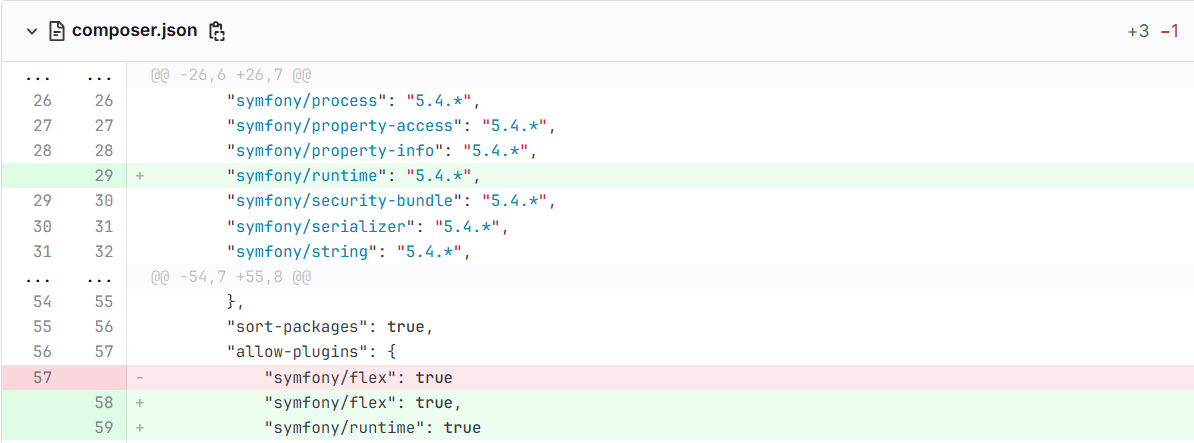


Figure : Commit concernant l’ajout de symfony/runtime

#### Mise à niveau vers Symfony 6.4

Une fois les dépréciations supprimées et les recettes à jour, les conditions techniques ont été réunies pour procéder à la mise à niveau vers Symfony 6.4. Les derniers messages d’avertissement dans le profiler Symfony provenaient uniquement de dépendances externes (dans vendor/), non modifiables directement.

Le fichier composer.json a été modifié pour mettre à jour tous les packages symfony/\* de la version 5.4.\* vers 6.4.\*.

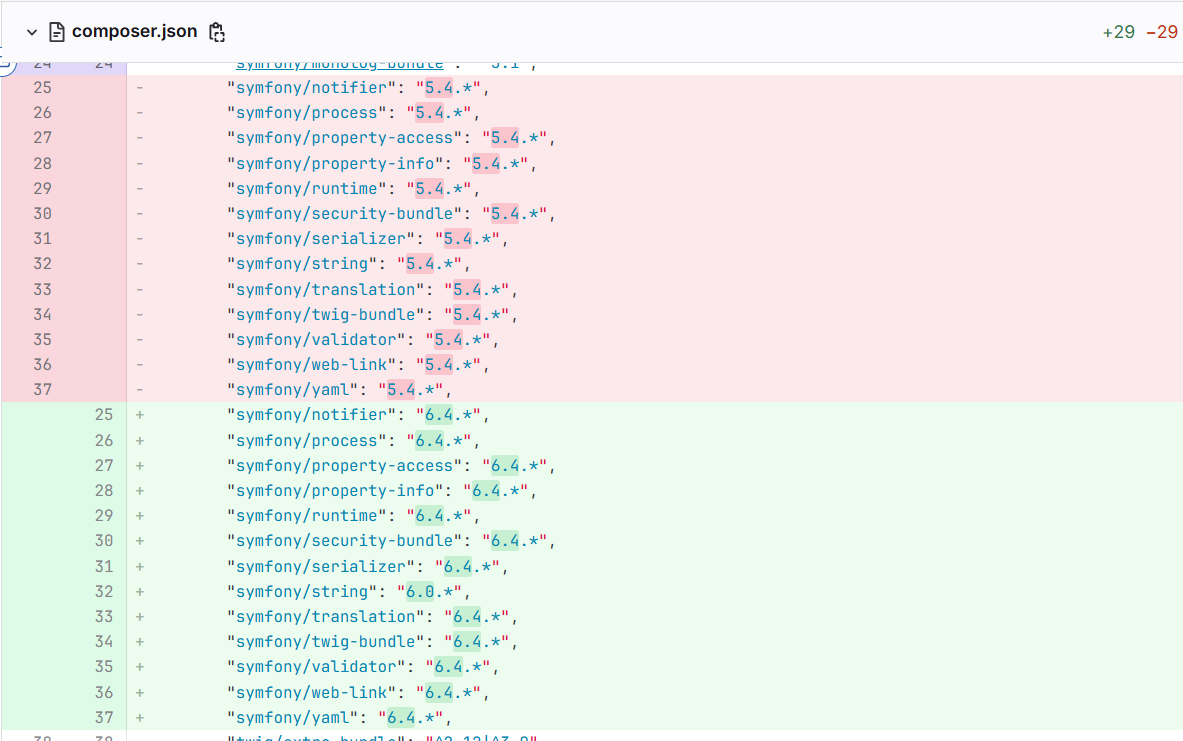


Figure : Commit de modification de composer.json pour la mise à jour de Symfony

La commande suivante a été exécutée pour appliquer la mise à niveau :

composer update 'symfony/\*'

Une image contenant texte, capture d’écran, menu

Le contenu généré par l’IA peut être incorrect.

Figure : Exécution de "symfony update symfony/\*"

Le processus s’est conclu avec succès. L’application a pu démarrer sous Symfony 6.4 sans erreur.

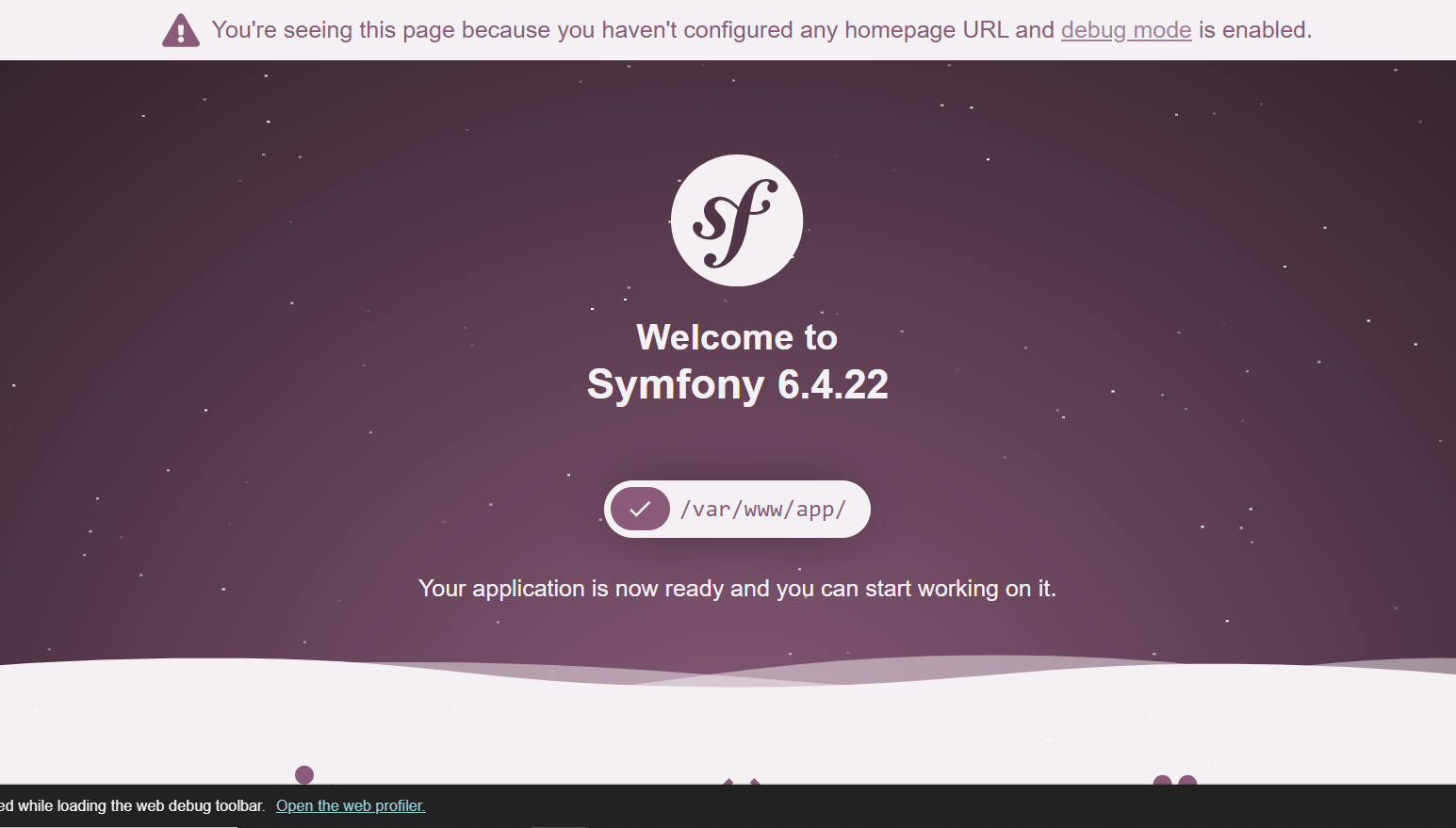


Figure : Page d'accueil de Symfony 6.4

#### Conclusion

Cette première phase expérimentale, menée sur un projet Symfony 5.4 minimal, a permis de valider le processus de migration vers Symfony 6.4 dans un environnement maîtrisé. Elle a servi de base d’apprentissage pour identifier les principales dépréciations, tester les outils d’automatisation tels que Rector, et consolider une méthodologie reproductible. Les enseignements tirés de cette étape ont constitué un socle technique solide avant d’aborder la migration du projet Autossimo proprement dit

### Migration de composer.json de projet Autossimo

Cette étape visait à intégrer le fichier composer.json du projet Autossimo dans la structure du projet de test précédemment utilisé. L’objectif principal était d’évaluer et gérer les dépréciations ainsi que les dépendances du projet sans intervenir directement sur le code métier. Cette intégration technique constitue une base essentielle avant d’entamer la migration complète du code source.

Une nouvelle branche intitulée migration-autossimo/composer.json a été créée pour encapsuler cette phase. Afin d’assurer la compatibilité initiale, le fichier Dockerfile a été modifié pour rétrograder temporairement la version de PHP, permettant ainsi l’exécution correcte de la commande :

composer install



Figure : commit de « composer install fonctionne bien »

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Figure : Détails de commit "Composer install fonctionne bien"

Une fois l’installation des dépendances terminée, l’outil Symfony Profiler a été utilisé pour identifier les dépréciations toujours présentes dans l’application.

Une image contenant texte, capture d’écran, Police, logiciel

Le contenu généré par l’IA peut être incorrect.

Figure : Aperçu des dépréciations au niveau de Symfony Profiler

#### Dépréciations

Les mêmes avertissements de dépréciation identifiés précédemment sont réapparus, principalement liés à la configuration. Les mesures correctives appliquées incluent :

* La mise à jour des recettes Symfony via composer recipes:update ;
* L’ajout de l’extension PHP intl dans l’environnement Docker.



Figure : Commits de mise à jour des recettes et l'ajout de PHP Intl extension

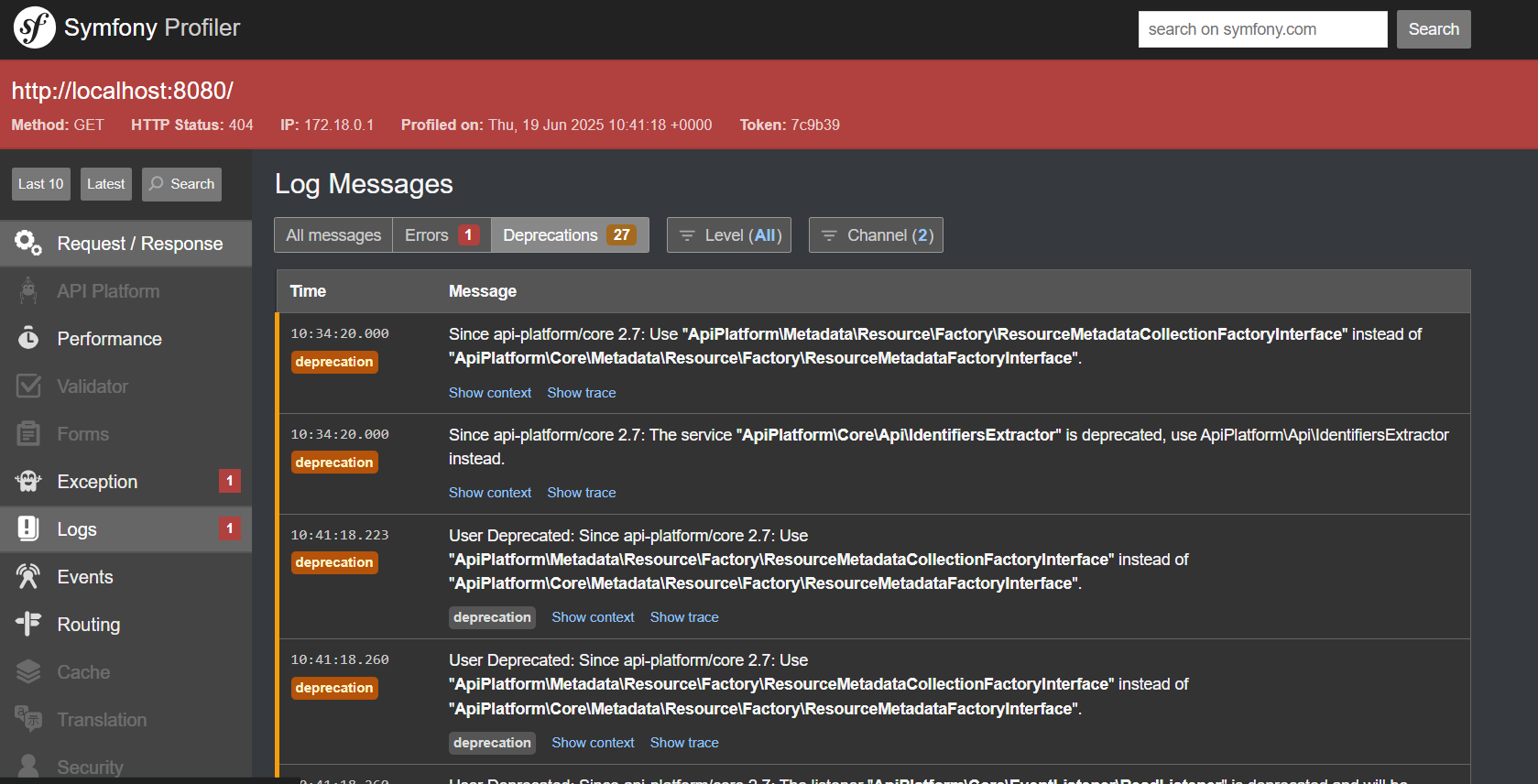


Figure : Dépréciations après la mise à jour des recettes

#### Mise à niveau vers PHP 8.2

La majorité des dépréciations ayant été résolue, une migration vers PHP 8.2 a été entreprise. Cette étape comprenait :

* La mise à jour de l’image Docker pour intégrer PHP 8.2 ;
* La modification de la contrainte PHP dans composer.json ;
* L’exécution de la commande composer update pour aligner les dépendances avec les nouvelles exigences.

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Figure : L'ajout de l'image de PHP 8.2 dans Docker



Figure : Modification de la contrainte PHP dans `composer.json`

Toutefois, l’opération a généré une erreur de compatibilité liée aux versions actuelles de certains packages. Ces packages ont été mis à niveau vers des versions prenant en charge PHP 8.2, ce qui a permis de finaliser l’installation avec succès.

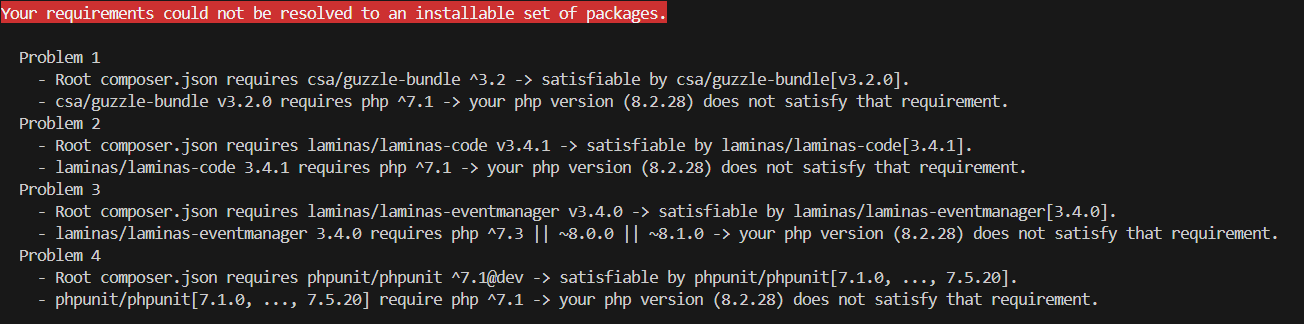


Figure  : Erreur rencontrée lors de `composer update`

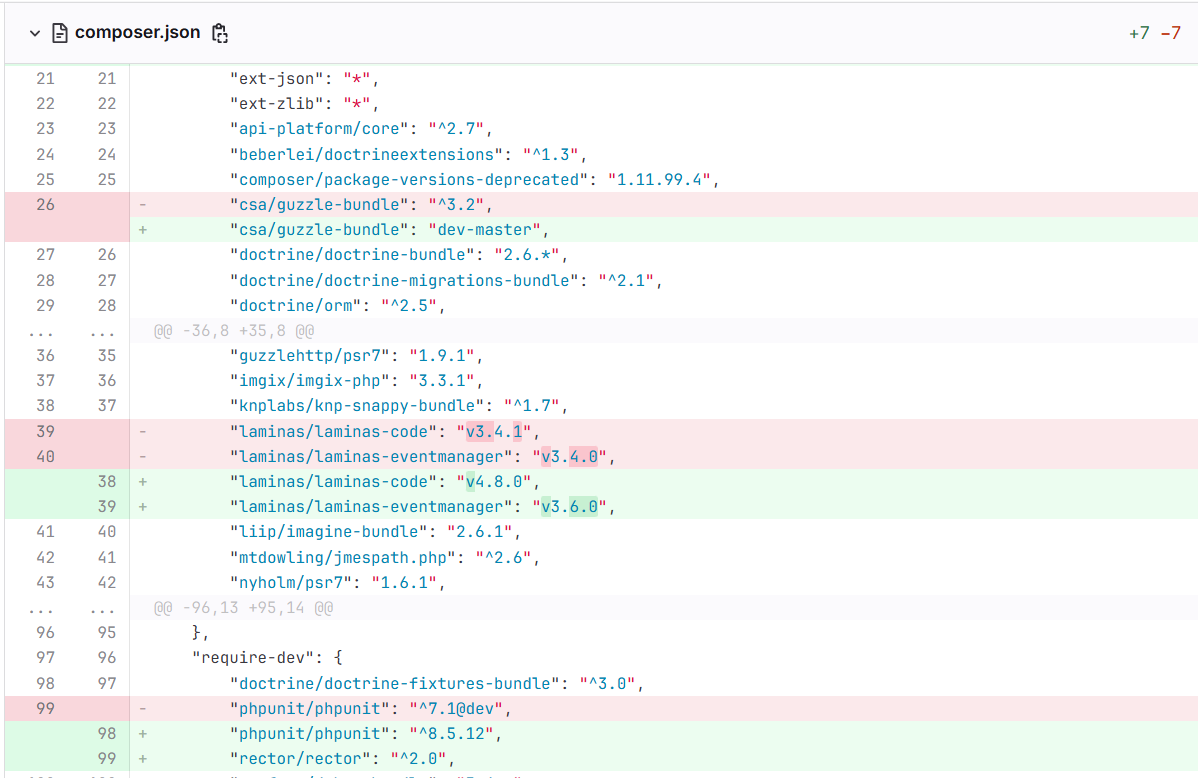


Figure  : Détail du commit de mise à jour des packages

Une image contenant texte, capture d’écran

Le contenu généré par l’IA peut être incorrect.

Figure : Confirmation du passage à PHP 8.2

#### Installation de Rector

Dans la continuité des travaux précédents, Rector a été installé afin d’automatiser la refactorisation et de tirer parti des optimisations proposées par PHP 8.2. L’outil a été intégré dans le fichier composer.json pour préparer les ajustements futurs sur le code métier.



Figure : Intégration de Rector dans les dépendances du projet

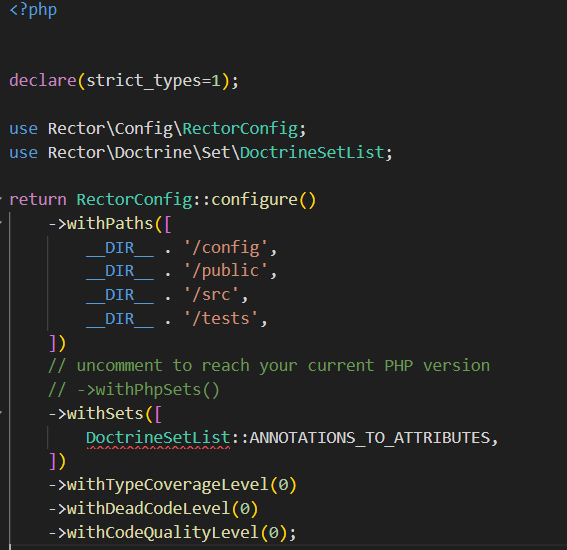


Figure : Configuration de Rector

La configuration appliquée à Rector permet à l’outil d’identifier les annotations utilisées par Doctrine et de les convertir automatiquement en attributs PHP, conformément aux standards introduits avec les versions récentes du langage. Une fois la configuration terminée, l’exécution du processus s’est faite via la commande suivante :

./vendor/bin/rector process.



Figure : Résultat de l’exécution de Rector

#### Mise à niveau vers Symfony 6.4

Une fois l’environnement stabilisé sous PHP 8.2 et les principales dépréciations corrigées, la mise à niveau vers Symfony 6.4 a pu être envisagée. Un examen du Symfony Profiler a néanmoins révélé la persistance de certaines alertes.

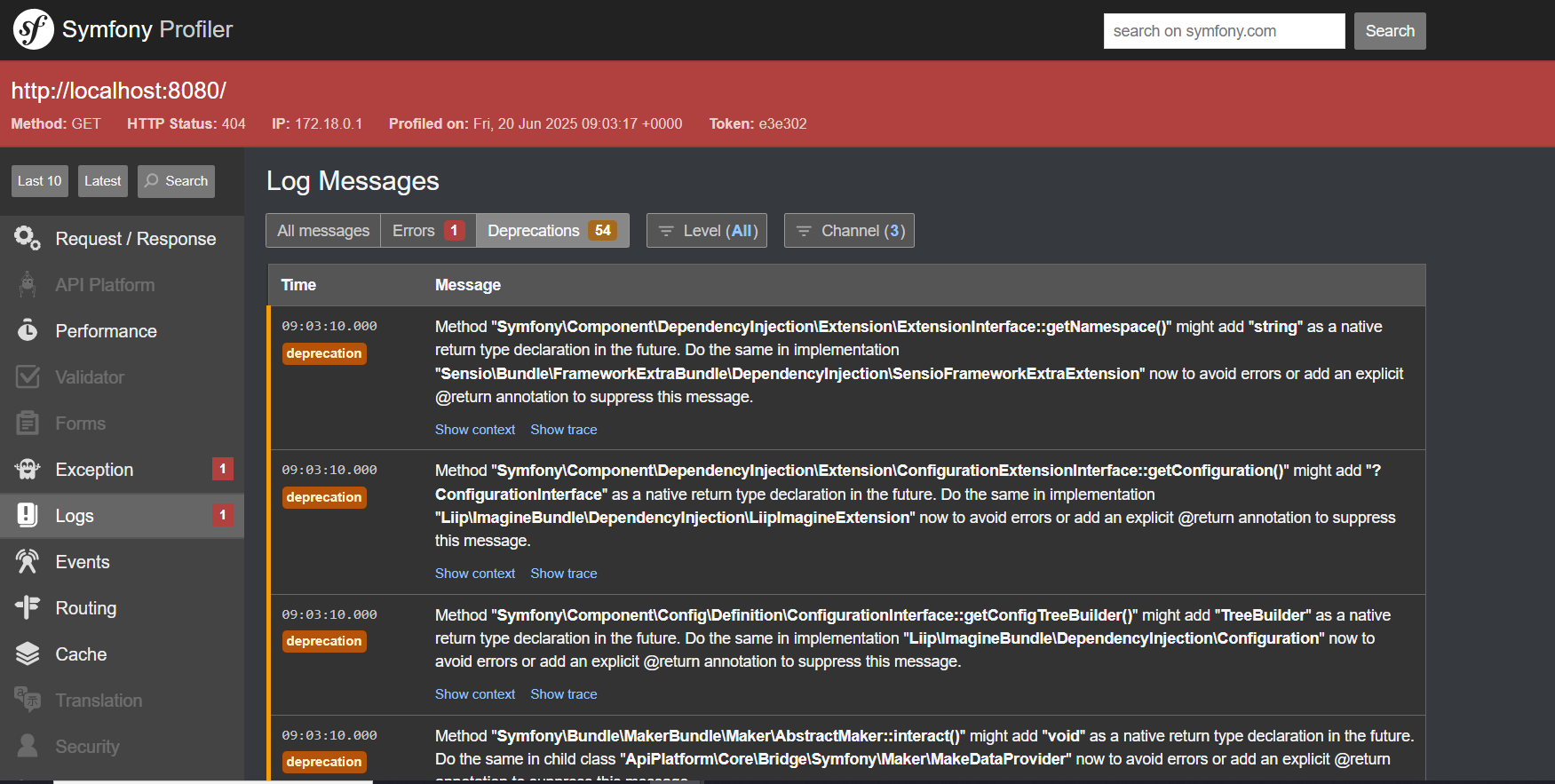


Figure : Symfony profiler

L’analyse de ces messages a permis de confirmer que la majorité des dépréciations restantes provenait de dépendances externes situées dans le dossier vendor/, donc hors du périmètre direct de modification. La mise à jour vers Symfony 6.4 étant techniquement justifiée, les versions des paquets symfony/\* ont été ajustées à 6.4.\* dans le fichier composer.json, suivie de l’exécution de la commande :

composer update symfony/\*.

#### Problèmes rencontrés

La tentative de mise à jour a échoué avec un message d’erreur relatif à des contraintes de dépendances.

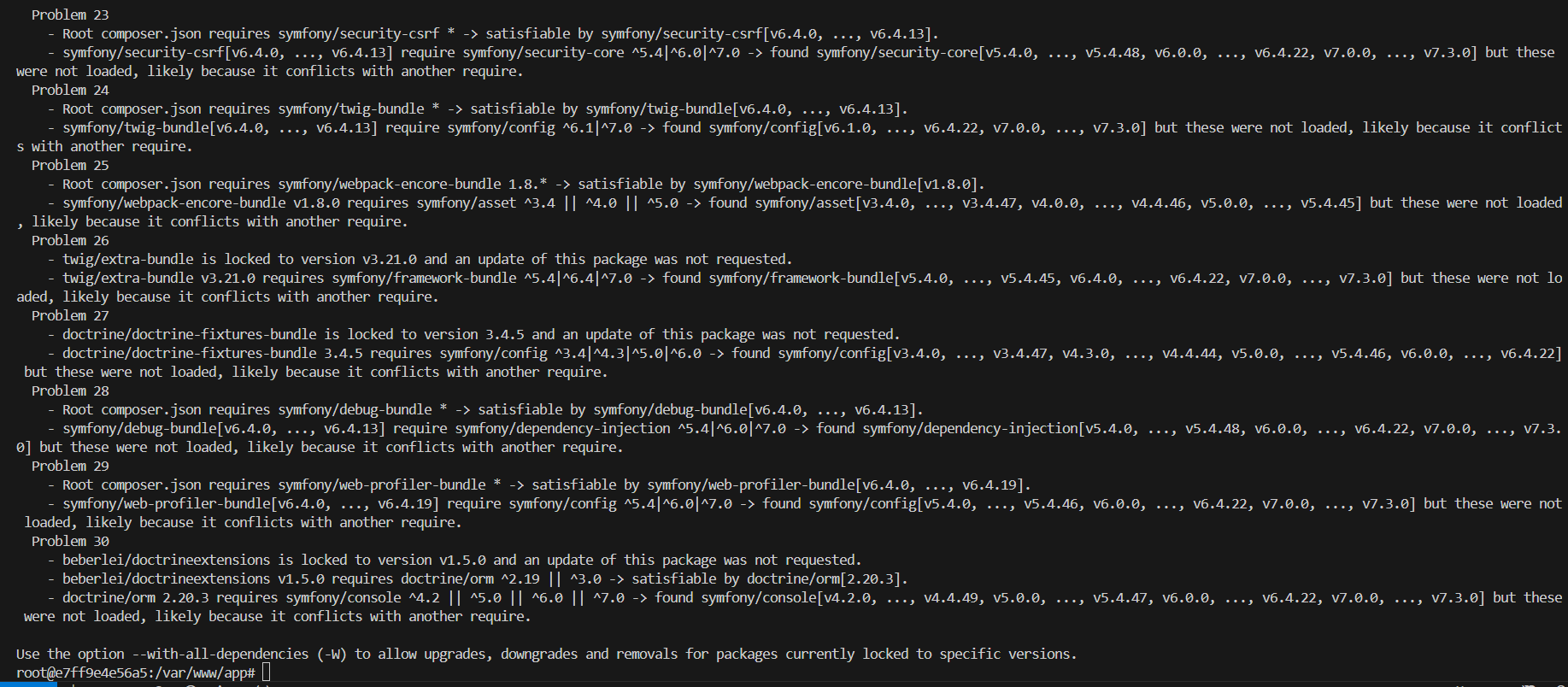


Figure : Erreur de composer update symfony/\*

Le gestionnaire de paquets a suggéré de relancer la commande avec l’option -W (ou --with-all-dependencies) afin de forcer la résolution globale :

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Figure : Erreur persistante malgré l’option -W

L’erreur subsistait, indiquant l’incompatibilité de plusieurs paquets tiers avec Symfony 6.4. L’analyse a permis d’identifier deux catégories problématiques :

##### Les packages non-compatible :

Plusieurs dépendances du projet sont obsolètes ou non compatibles avec Symfony 6.4. Des remplacements ont été identifiés :

* **csa/guzzle-bundle :** utilisé pour l’envoi de requêtes HTTP, à remplacer par symfony/http-client.
* **symfony/swiftmailer-bundle :** utilisé pour la gestion des courriels, à remplacer par symfony/mailer (la migration partielle est déjà en place).
* **trikoder/oauth2-bundle :** utilisé pour l’authentification OAuth2, à remplacer par league/oauth2-server-bundle.

##### Les packages abandonnés :

L’exécution de la commande composer outdated a permis d’identifier des packages obsolètes ou non maintenus, nécessitant un remplacement :

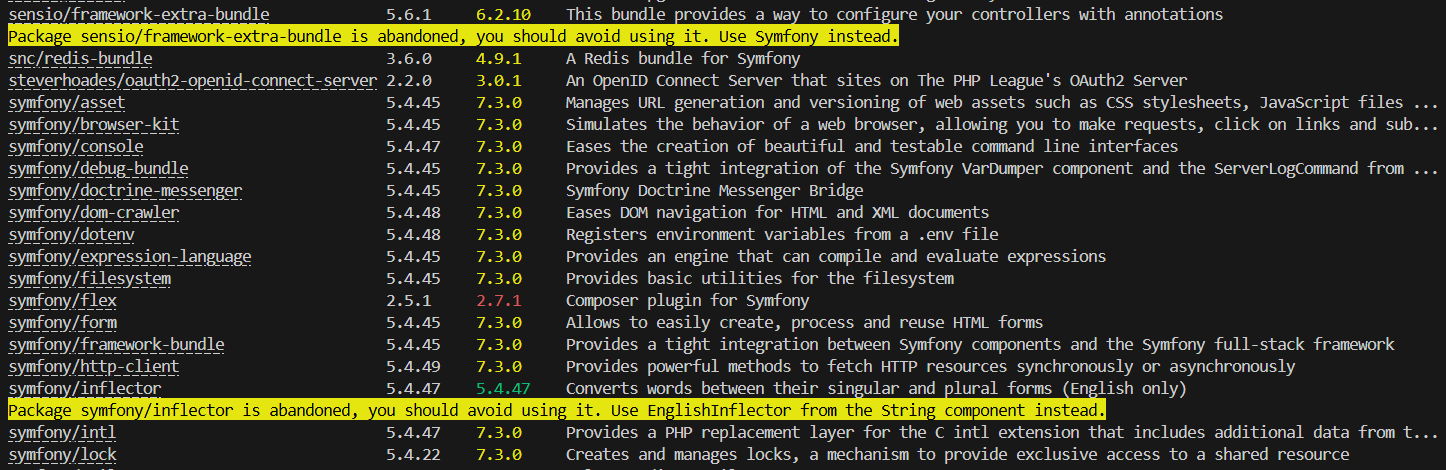


Figure : Resultat de la commande "composer outdated"

* **symfony/inflector :** composant de transformation linguistique (pluriel/singulier en anglais), à remplacer par EnglishInflector du composant symfony/string.
* **sensio/framework-extra-bundle :** utilisé pour la configuration via annotations dans les contrôleurs Symfony, dont les fonctionnalités sont désormais nativement prises en charge par le noyau de Symfony.

#### Conclusion

La migration du fichier composer.json du projet Autossimo a constitué une étape déterminante dans la validation de la compatibilité de l’environnement technique avec Symfony 6.4 et PHP 8.2. Cette phase a permis de confronter les dépendances réelles du projet à l’ensemble des contraintes de versions imposées par les nouvelles technologies cibles.

L’analyse approfondie des erreurs générées lors des mises à jour a révélé la présence de dépendances obsolètes ou incompatibles, nécessitant leur remplacement par des alternatives modernes et maintenues. Cette démarche a également mis en lumière l’importance d’un alignement progressif entre les exigences du framework Symfony et l’écosystème de bibliothèques tiers utilisé par le projet.

Enfin, l’intégration de Rector et la résolution manuelle des dépréciations ont renforcé la stabilité de l’environnement, ouvrant la voie à une transition sécurisée vers une base de code conforme aux standards actuels. Cette étape de migration technique constitue ainsi le fondement sur lequel reposera la future migration du code métier de l'application Autossimo.

### Analyse du code Autossimo

Après avoir initialisé un projet Symfony vierge à des fins d’expérimentation, le fichier composer.json issu du projet Autossimo y a été intégré. Cette démarche a permis de reconstituer un environnement technique représentatif du projet réel, sans y inclure dans un premier temps le code métier, afin de focaliser l’analyse sur les dépendances et les dépréciations techniques.

Cette phase d’analyse a pour objectif d’identifier les éléments obsolètes ou incompatibles avec Symfony 6.4 et PHP 8.2. En particulier, elle permet de relever les dépréciations générées par les packages installés, les usages de composants techniques devenus obsolètes, ainsi que les éventuelles limitations imposées par des bibliothèques tierces.

L’analyse repose sur l’exploitation d’outils tels que Symfony Profiler, Rector et Composer, combinée à une vérification manuelle de la configuration et des services. Les résultats obtenus serviront de base à la planification des actions correctives et à l’évaluation des efforts nécessaires pour garantir la compatibilité complète du projet Autossimo avec le nouveau stack technique.

#### Liste des dépréciations identifiées dans Autossimo

La liste des dépréciations que j’ai identifiées dans Autossimo en utilisant Symfony Profiler :

##### [SessionInterface](https://symfony.com/blog/new-in-symfony-5-3-session-service-deprecation)

Il est déconseillé d'utiliser le service de session dans Symfony 5.3, et celui-ci sera supprimé dans Symfony 6.0.

À la place de l'injection directe du service de session, il convient désormais d'injecter le service RequestStack et d'utiliser la méthode getSession()

* **Solution :**
* Dans chaque injection du dépendance (soit via \_\_construct ou setter), il faut inject RequestStack au lieu du SessionInterface
* Par la suite, on doit stocker le $requestStack au niveau du propriété de la classe
* Et en fin lorsqu’on a besoin d’utiliser la session, on peut :

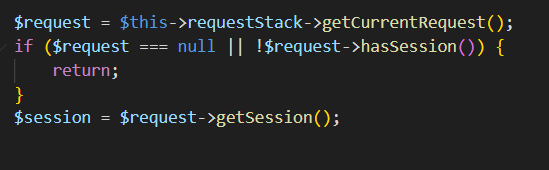


Figure : Code pour récupérer la session

(Il faut faire cette vérification car parfois il n’y a pas de requête http, et s’il n’y a pas de requête http et on essaie de récupérer la session, il va déclencher un erreur « No Session found »)

* **Impact sur le code :**

Cette interface est référencée dans 123 fichiers à travers l'application.

##### [ContainerInterface](https://symfony.com/doc/5.x/components/dependency_injection.html" \l "avoiding-your-code-becoming-dependent-on-the-container)

Il n'est plus recommandé d'injecter le ContainerInterface, il est plutôt recommandé d'injecter les services dont vous avez besoin à la place.

* **Solution :**
* L'injection du service directement à chaque fois qu’on en a besoin.
* Sinon on peut continuer de l’utiliser, il suffit de le définir explicitement dans *«services.yaml ».*

Symfony\Component\DependencyInjection\ContainerInterface: '@service\_container'

* **Impact sur le code :**

Cette interface est référencée dans 32 fichiers à travers l'application.

##### [LogoutSuccessHandlerInterface :](https://symfony.com/blog/new-in-symfony-5-1-simpler-logout-customization)

Dans Symfony 5.1, l'interface LogoutSuccessHandlerInterface a été abandonnée au profit d'une approche événementielle utilisant LogoutEvent. Ce changement offre plus de flexibilité et s'inscrit dans la tendance générale de Symfony vers une architecture événementielle.

* **Solution :**
* On doit maintenant créer un écouteur d’événement pour LogoutEvent
* **Impact sur le code :**

Cette Interface est référencée dans un seul fichier.

##### [Composer Runtime API is deprecated :](https://symfony.com/blog/new-in-symfony-5-3-runtime-component)

À partir de Symfony 5.3, le framework a commencé à encourager l'utilisation de l'API Runtime de Composer pour l'amorçage des applications. La méthode traditionnelle de configuration des applications Symfony sans cette API est désormais obsolète. Pour notre cas nous sommes déjà en train d’utiliser le nouveau runtime, mais probablement qu’il faut mis à jour le composer à la version 2.1 pour que la dépréciation disparue.

* **Solution :**
* Mise à jour du composer de la version 1.10 vers la version 2.1.

##### [PasswordAuthenticatedUserInterface :](https://symfony.com/blog/new-in-symfony-5-3-improvements-for-security-users#decoupled-passwords-from-users)

L’entité User doit implémenter PasswordAuthenticatedUserInterface :

* **Solution :**

<?php

namespace App\Entity;

use App\Validator\Constraints\SaneString;

use App\Validator\Constraints\EmailUniqueOpisto;

use Doctrine\ORM\Mapping as ORM;

use League\OAuth2\Server\Entities\Traits\EntityTrait;

use League\OAuth2\Server\Entities\UserEntityInterface;

use OpenIDConnectServer\Entities\ClaimSetInterface;

use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;

use Symfony\Component\Security\Core\User\UserInterface;

use Symfony\Component\Validator\Constraints as Assert;

use App\Validator\Constraints\EmailUnique;

use Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface;

/\*\*

 \* App\Entity\User.

 \*

 \* @ORM\Table(name="COMPTEREP")

 \* @ORM\Entity(repositoryClass="App\Repository\UserRepository")

 \*/

class User implements UserInterface, \Serializable, UserEntityInterface, ClaimSetInterface, PasswordAuthenticatedUserInterface

{

    ...

    /\*\*

     \* {@inheritdoc}

     \*

     \* @return string

     \*/

    public function getPassword(): ?string

    {

        return $this->password;

    }

    ...

}

* **Impact sur le code :**

Un seul fichier.

##### [UserPasswordEncoderInterface => UserPasswordHasherInterface :](https://symfony.com/blog/new-in-symfony-5-3-passwordhasher-component)

UserPasswordEncoderInterface est déprécié, il doit être remplacé par UserPasswordHasherInterface.

* **Solution :**
* Implémenter UserPasswordHasherInterface au lieu de UserPasswordEncoderInterface
* Injecter UserPasswordHasherInterface dans le constructeur au lieu de UserPasswordEncoderInterface
* (Facultatif) Renommer la propriété de la classe et le nom de la variable de la classe :  
  $passwordEncoder => $passwordHasher
* **Impact sur le code :**

Un seul fichier impacté.

##### [Sécurité et encodeur](https://symfony.com/blog/new-in-symfony-5-3-passwordhasher-component)

Le nœud enfant « encoders » dans le chemin « security » est obsolète, utilisez plutôt « password\_hashers »

* **Solution :**
* Dans le sécurity.yaml, changer le « encoders » au « password\_hashers »

security:

    erase\_credentials: false

    enable\_authenticator\_manager: true

    access\_decision\_manager:

        strategy: unanimous

    password\_hashers:

        App\Entity\User: plaintext

* **Impact sur le code :**

Un seul fichier impacté.

##### [UserInterface::getUserIdentifier() :](https://symfony.com/blog/new-in-symfony-5-3-improvements-for-security-users" \l "renamed-username-to-identifier)

Les classes qui implémentent cette interface doit implémenter aussi getUserIdentifier méthode au lieu de getUsername.

* **Solution :**
* On peut changer le nom du getUsername => getUserIdentifier dans la classe qui implémente cette interface ou juste ajouter getUserIdentifier() dans la classe et faire une retourne du valeur « $this->getUsername() »
* Sinon si on change getUsername => getUserIdentifier, donc on doit se rassurer de faire la même chose dans tous les emplacements qui font appels à la méthode getUsername().
* **Impact sur le code :**

L’impact de cette méthode est présent dans 2 fichiers.

##### [RedisSessionHandler :](https://symfony.com/blog/new-in-symfony-5-3-improvements-for-security-users" \l "renamed-username-to-identifier)

Il est recommandé d’utiliser la classe RedisSessionHandler native au lieu de SncRedis pour l’emplacement de stockage des sessions.

* **Solution :**
* Dans le services.yaml :

app.redis:

        class: Predis\Client

        arguments:

            - 'redis://redis'

    Symfony\Component\HttpFoundation\Session\Storage\Handler\RedisSessionHandler:

        arguments:

            - '@app.redis'

            - { prefix: 'sess:', ttl: 3600 }

App\Security\SessionHandler:

        arguments:

            $redis: '@app.redis' # FROM '@snc\_redis.session' => '@app.redis'

            $options:

                prefix: 'session'

* Dans framework.yaml :

    session:

        name: p\_Session

        handler\_id: Symfony\Component\HttpFoundation\Session\Storage\Handler\RedisSessionHandler # Change from App\Security\SessionHandlerFactory

* On doit commenter les lignes de config/packages/snc\_redis.yaml

snc\_redis:

  clients:

    # session:

    #   type: predis

    #   alias: session

    #   dsn: "%redis\_server\_session%"

    #   options:

    #     connection\_timeout: 2

    #     read\_write\_timeout: 2

    cache:

      type: predis

      alias: cache

      dsn: "%redis\_server\_cache%"

  # session:

  #   client: session

* **Impact sur le code :**

L’impact dans 5 fichiers.

##### [AuthenticationProviderInterface :](https://github.com/hwi/HWIOAuthBundle/issues/1783)

AuthentificationProviderInterface est déprécié et il ne doit pas être utiliser.

##### AdapterInterface => CacheItemPoolInterface :

AdapterInterface est déprécié et il est recommandé d’utiliser CacheItemPoolInterface à la place.

* **Solution :**

public function \_\_construct(ManagerRegistry $registry, CacheItemPoolInterface $cacheItemPool, LoggerInterface $logger)

* **Impact sur le code :**

L’impact de cette interface est présent seulement dans un fichier.

##### [Reset\_on\_message :](https://symfony.com/blog/new-in-symfony-5-4-messenger-improvements" \l "reset-container-services-between-messages)

Ne pas définir l'option de configuration « framework.messenger.reset\_on\_message » est obsolète, elle sera par défaut sur « true » dans la version 6.0.

* **Solution :**

framework:

...

    messenger:

        reset\_on\_message: true

...

* **Impact sur le code :**

L’impact est sur un seul fichier.

##### [Storage\_factory\_id :](https://github.com/symfony/framework-bundle/blob/7.2/CHANGELOG.md" \l "user-content-53)

Ne pas définir l'option de configuration « framework.session.storage\_factory\_id » est obsolète, elle sera par défaut « session.storage.factory.native » et remplacera l'option de configuration « framework.session.storage\_id » dans la version 6.0.

* **Solution :**

# see https://symfony.com/doc/current/reference/configuration/framework.html

framework:

    messenger:

        reset\_on\_message: true

    secret: "%secret%"

    csrf\_protection: true

    http\_method\_override: true

    session:

        name: p\_Session

        handler\_id: App\Security\SessionHandlerFactory

        cookie\_secure: false

        storage\_factory\_id: 'session.storage.factory.native'

* **Impact sur le code :**

Un seul fichier qui doit être changer

##### PHP Intl Extension :

L’installation du PHP Intl

* **Solution :**

RUN docker-php-ext-install intl

* **Impact sur le code :**

Un fichier qui doit être changer (dans la production doit être change par l’équipe d’exploitation).

#### Gestion des dépréciations avec PHPUnit

Lors de l’exécutez des tests avec PHPUnit, aucun avis de dépréciation n'est affiché. Pour nous aider, Symfony fournit un pont PHPUnit. Ce pont affiche un résumé clair de tous les avis de dépréciation à la fin du rapport de test.

En ce qui concerne le projet, symfony/phpunit-bridge a été déjà installé (en utilisant composer require --dev symfony/phpunit-bridge), cela permet de visualiser les dépréciations directement lors de l’exécution des tests : ./vendor/bin/phpunit

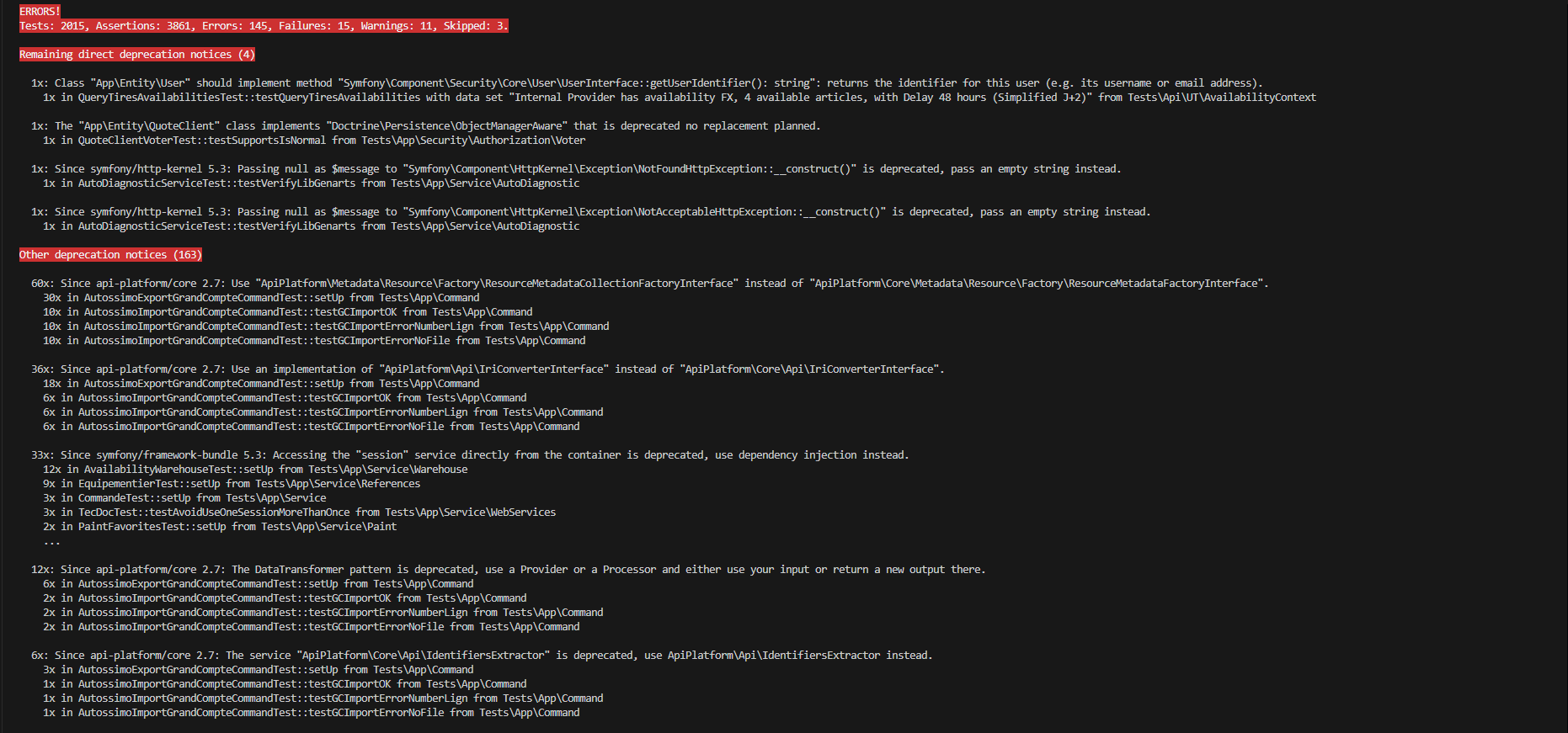


Figure : Dépréciations au niveau de PHPUnit

#### Les packages abandonnées et incompatible

Afin de poursuivre notre migration, il est important de remplacer les bundles et packages incompatibles présents dans notre projet.

Après avoir analysé les messages d'erreur, exécuté quelques commandes de Composer (composer why-not php ^8.2, composr why-not symfony/framework-bundle ^8.2...), consulté les journaux et la page officielle GitHub de chaque package

##### [Csa/guzzle-bundle :](https://packagist.org/packages/csa/guzzle-bundle)

* **Description :**
* Ce bundle est abandonné.
* Il ne supporte pas les versions supérieures du 5 de [symfony/framework-bundle](https://packagist.org/packages/symfony/framework-bundle).
* La version actuelle (v3.2) ne supporte pas la version 8.2 de PHP.
* **Solution :**
* Doit être remplacé être par : [symfony/http-client](https://packagist.org/packages/symfony/http-client).
* **Impact sur le code :**

Ce packages est référencé dans 52 fichiers à travers l'application.

##### [Symfony/swiftmailer-bundle :](https://packagist.org/packages/symfony/swiftmailer-bundle)

* **Description :**
* Ce bundle est abandonné.
* Il ne supporte pas les versions supérieures du 5 de [symfony/framework-bundle](https://packagist.org/packages/symfony/framework-bundle).
* **Solution :**
* Doit être remplacé par [symfony/mailer](https://packagist.org/packages/symfony/mailer) (apparemment déjà la plupart de travail est déjà faite)
* **Impact sur le code :**

Ce package est référencé dans 5 fichiers.

##### [Trikoder/oauth2-bundle :](https://packagist.org/packages/trikoder/oauth2-bundle)

* **Description :**
* Ce bundle est abandonné.
* Il ne supporte pas les versions supérieures du 5.4 de [symfony/framework-bundle](https://packagist.org/packages/symfony/framework-bundle).
* **Solution :**
  + Doit être remplacé par [league/oauth2-server-bundle](https://packagist.org/packages/league/oauth2-server-bundle).
* **Impact sur le code :**

Ce package est référencé dans 4 fichiers.

##### [Symfony/inflector :](https://packagist.org/packages/symfony/inflector)

* **Description :**
* Le composant Inflector convertit les mots entre leurs formes singulières et plurielles (anglais uniquement).
* Ce bundle est abandonné.
* **Solution :**
  + Doit être remplacé par EnglishInflector de [Symfony/string](https://packagist.org/packages/symfony/string?type=application&page=12)

##### [Sensio/framework-extra-bundle :](https://packagist.org/packages/sensio/framework-extra-bundle)

* **Description :**
* Ce bundle fournit un moyen de configurer vos contrôleurs avec des annotations.
* Ce bundle est abandonné.
* **Solution :**
* Doit être remplacé par les fonctionnalités de Symfony native.
* **Impact sur le code :**

Ce packages est référencé dans 29 fichiers à travers l'application.

##### [Laminas/laminas-code :](https://packagist.org/packages/laminas/laminas-code)

* **Description :**
  + Extensions de l'API PHP Reflection, analyse de code statique et génération de code.
* La version actuelle ("v3.4.1") ne supporte pas la version 8.2 de PHP.
* **Solution :**
* Il faudrait mettre à jour vers la version "v4.8.0".

##### [Laminas/laminas-eventmanager :](https://packagist.org/packages/laminas/laminas-eventmanager)

* **Description :**
* Déclencher et écouter des événements dans une application PHP
* La version actuelle ("v3.4.0") ne supporte pas la version 8.2 de PHP.
* **Solution :**
  + Il faudrait mettre à jour vers la version "v3.6.0".

##### [Phpunit/phpunit :](https://packagist.org/packages/phpunit/phpunit)

* **Description :**
* Un puissant framework de test pour PHP, conçu pour aider les développeurs à écrire des tests unitaires afin de garantir le bon fonctionnement de leur code.
* La version actuelle ("^7.1@dev") ne supporte pas la version 8.2 de PHP.
* **Solution :**
* Il faudrait mettre à jour vers la version "^8.5.12.

**Remarque :**

La version minimale compatible a été retenue, dans la mesure où une version antérieure réduit le nombre d’ajustements requis pour assurer le bon fonctionnement de la version cible.

#### Passage à PHP 8.2

Symfony 6.4 nécessite une version minimale de PHP 8.1. Ce passage vers une version plus récente du langage a permis de tirer parti d’améliorations notables en matière de performances ainsi que de fonctionnalités modernes, telles que les attributs natifs et les constructeurs promus. Pour des raisons de compatibilité et d’alignement avec les environnements cibles, la décision a été prise d’effectuer directement la migration vers PHP 8.2.

##### Les packages/bundles incompatible avec PHP 8.2

La migration a nécessité une révision des dépendances listées dans le fichier composer.json, notamment pour les bibliothèques ne prenant pas encore en charge PHP 8.2. Certaines dépendances ont pu être mises à jour vers des versions compatibles ; d’autres, telles que csa/guzzle-bundle, ont été temporairement pointées vers une branche de développement (dev-master) en attendant leur remplacement par des alternatives modernes.

Avant :

        "csa/guzzle-bundle": "^3.2",

        "laminas/laminas-code": "v3.4.1",

        "laminas/laminas-eventmanager": "v3.4.0",

        "phpunit/phpunit": "^7.1@dev",

Après :

        "csa/guzzle-bundle": "dev-master",

        "laminas/laminas-code": "v4.8.0",

        "laminas/laminas-eventmanager": "v3.6.0",

        "phpunit/phpunit": "^8.5.12",

##### Mise à jour de l’environnement Docker

L’image Docker a été adaptée pour refléter la version cible :

ARG PHP\_VERSION=8.2

### PHP

FROM php:${PHP\_VERSION}-fpm AS symfony\_docker\_php

Après modification, l’environnement a été reconstruit à l’aide de la commande :

docker compose up php -d –build

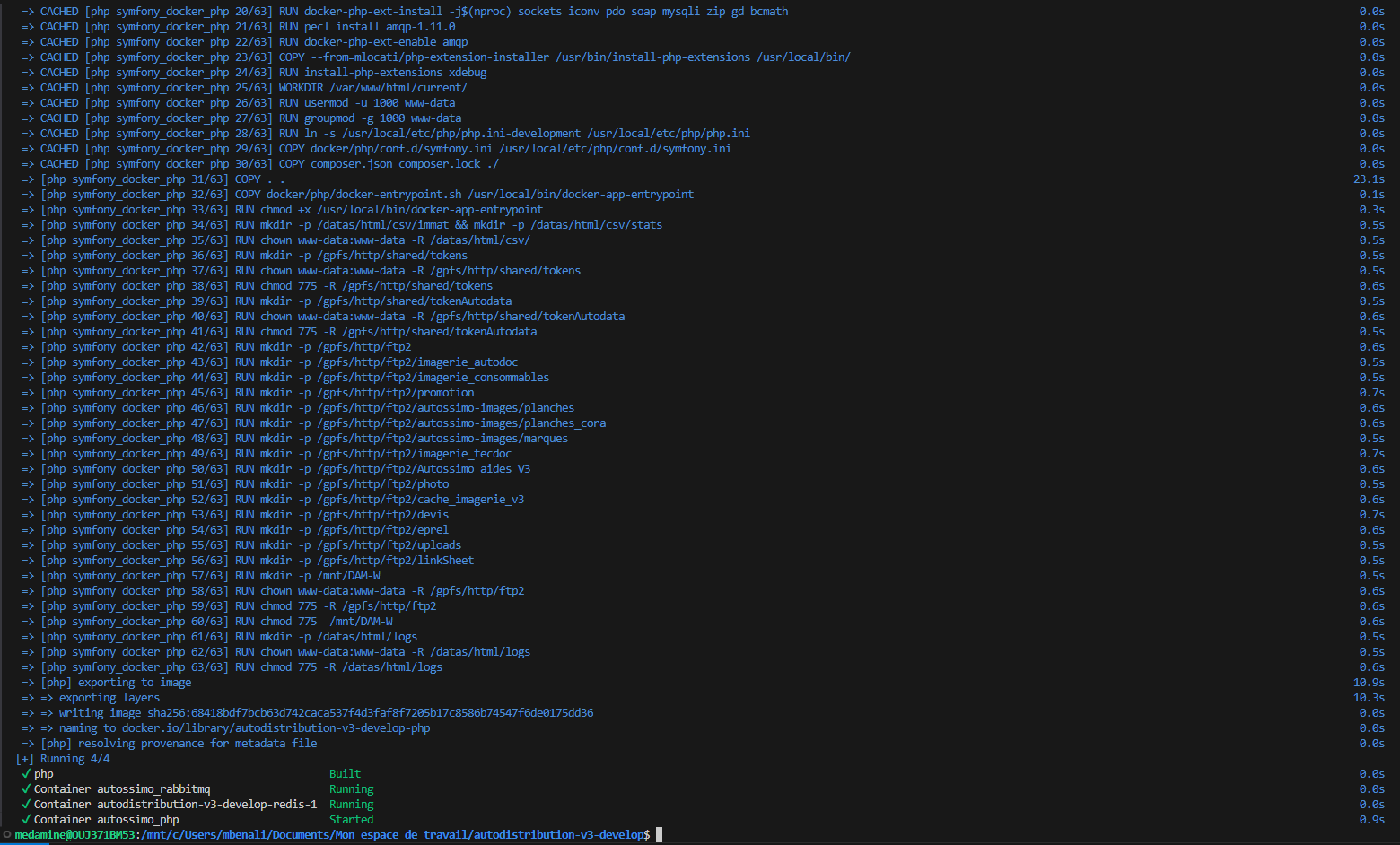


Figure : L'exécution de la commande "docker compose up php -d --build"

##### Mise à jour dans le fichier composer.json

Le fichier composer.json a été ajusté pour exiger PHP 8.2 :

  "require": {

    "php": "^8.2",

    "ext-json": "\*",

    "ext-zlib": "\*",

Les mises à jour ont été appliquées dans le conteneur après exécution de :

make enter

composer update

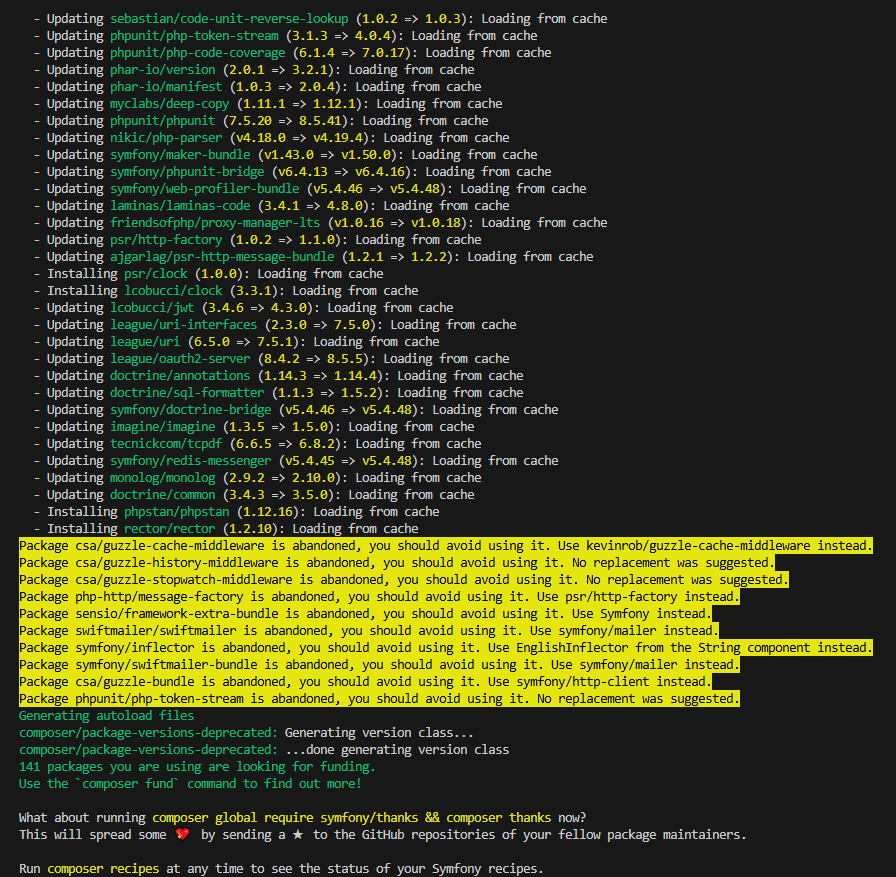


Figure : L'exécution de la commande "composer update"

#### Utilisation des nouvelles fonctionnalités de PHP 8

Le passage à PHP 8.2 ouvre la voie à l’intégration de fonctionnalités avancées :

* Utilisation des attributs natifs à la place des annotations (ex. Doctrine) ;
* Adoption du constructeur promu pour l’injection de dépendances simplifiée ;
* Renforcement du typage strict ;
* Optimisation générale des performances.

Exemple de migration Doctrine :

<?php

declare(strict\_types=1);

use Doctrine\ORM\Mapping as ORM;

// Ancien style avec annotations

/\*\*

 \* @ORM\Entity()

 \*/

class User { }

// Nouveau style avec attributs PHP

#[ORM\Entity]

class User { }

L’automatisation de cette migration est possible via Rector :

$rectorConfig->withSets([

    LevelSetList::UP\_TO\_PHP\_82,

    DoctrineSetList::ANNOTATIONS\_TO\_ATTRIBUTES,

]);

Commande : ./vendor/bin/rector process

#### Mise à jour de Symfony 5.4 vers 6.4

Une fois les dépréciations corrigées et les dépendances compatibles avec PHP 8.2, la mise à niveau vers Symfony 6.4 a été entreprise.

##### Modifications dans composer.json

Il faut mettre à jour toutes les dépendances commençant par "symfony/" et dont la version est définie sur : v5.4.\*, 5.4.\* ou ^5.4. Il est important de respecter ces deux conditions : d'une part, nous souhaitons mettre à jour uniquement les packages officiels Symfony vers la version 6.4, et d'autre part, certains packages Symfony ont leur propre système de gestion des versions, comme Flex par exemple. Le fait de les traiter tous de la même manière pourrait engendrer des problèmes à l'avenir.

Donc en changeant la version de toutes les dépendances qui respectent ces conditions, et en les définissant sur : v6.4.\*, 6.4.\*, ^6.4 ; elles sont toutes identiques.

{

"...": "...",

"require": {

- "symfony/cache": "5.4.\*",

+ "symfony/cache": "6.4.\*",

- "symfony/config": "5.4.\*",

+ "symfony/config": "6.4.\*",

- "symfony/console": "5.4.\*",

+ "symfony/console": "6.4.\*",

"...": "...",

"...": "A few libraries starting with symfony/ follow their own

versioning scheme (e.g. symfony/polyfill-[...],

symfony/ux-[...], symfony/[...]-bundle).

You do not need to update these versions: you can

upgrade them independently whenever you want",

"symfony/monolog-bundle": "^3.5",

},

"...": "...",

}

Une autre façon est de changer la version de toutes les dépendances qui respectent à nouveau les deux conditions, et de les définir sur "\*", après cela, nous allons dans extra.symfony.require et changeons sa valeur de 5.4.\* à 6.4.\*. Dans les applications Symfony utilisant Symfony Flex, ce paramètre limite les packages Symfony à une seule version spécifique, améliorant ainsi la gestion des dépendances et les performances de mise à jour de Composer, c'est un peu comme utiliser une variable ou un paramètre, ce qui est vraiment pratique car cela nous permet de changer la version une seule fois (dans extra.symfony.require).

{

"...": "...",

"require": {

- "symfony/cache": "6.4.\*",

+ "symfony/cache": "\*",

- "symfony/config": "6.4.\*",

+ "symfony/config": "\*",

- "symfony/console": "6.4.\*",

+ "symfony/console": "\*",

"...": "...",

},

"...": "...",

+ "extra": {

+ "symfony": {

+ "require": "6.4.\*"

+ }

+ }

}

##### Commande de mise à jour

composer update symfony/\*

Cette commande a permis de mettre à jour tout le projet vers Symfony 6.4, à condition que toutes les dépendances soient compatibles.

Une bonne pratique après une mise à jour vers une nouvelle version majeure consiste à vider le cache. Au lieu d'exécuter la commande cache:clear (inutile si l'application n'est pas démarrable dans la console après la mise à niveau), il est préférable de supprimer l'intégralité du contenu du répertoire de cache.

# Exécutez cette commande sous Linux et macOS

$ rm -rf var/cache/\*

# Exécutez cette commande sous Windows

C:\> rmdir /s /q var\cache\\*

###### Erreur de dépendance :

Lors de la mise à jour du projet vers Symfony 6.4, il est possible de rencontrer des erreurs liées aux dépendances. Ces erreurs peuvent indiquer que certaines bibliothèques tierces utilisées dans le projet ne sont pas compatibles avec les nouvelles versions de Symfony ou de PHP.

Une erreur de ce type peut signifier que d'autres bibliothèques, qui sont des dépendances indirectes des composants Symfony, doivent également être mises à jour. En effet, certaines librairies peuvent avoir des contraintes de version incompatibles avec celles imposées par Symfony 6.4.

Pour résoudre cela, il est conseillé d’utiliser l’option suivante lors de la commande de mise à jour :

composer update "symfony/\*" --with-all-dependencies

Cette commande permet non seulement de mettre à jour tous les paquets Symfony, mais aussi toutes leurs dépendances transitives. Cela garantit une cohérence globale entre les différentes bibliothèques installées dans le projet.

###### Contrôler les versions des dépendances

Bien que cette commande soit très utile, elle peut parfois entraîner des mises à jour non désirées si les contraintes de version dans le fichier composer.json ne sont pas suffisamment précises. Il est donc recommandé de définir des contraintes strictes pour chaque package dans le fichier composer.json. Par exemple :

"require": {

        "php": "^8.2",

        "symfony/symfony": "6.4.\*",

        "doctrine/doctrine-migrations-bundle": "^3.2"

    }

Cela permet de garder le contrôle sur les versions des bibliothèques tout en autorisant les mises à jour mineures nécessaires.

###### Problèmes de compatibilité persistants

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Figure : Erreur pendant la mise à jour de Symfony

Si malgré ces mesures la commande composer update continue de renvoyer des erreurs, cela peut indiquer que certaines bibliothèques sont bloquées sur des versions obsolètes ou incompatibles avec Symfony 6.4. Dans ce cas, il faut inspecter manuellement les versions spécifiées dans « composer.json » et envisager de les modifier pour utiliser des versions plus récentes et compatibles.

Il est également possible de tomber sur des conflits de versions entre bibliothèques. Composer affiche généralement des messages précis indiquant quelles dépendances entrent en conflit. Ces situations nécessitent souvent une analyse fine du graphe de dépendances et une adaptation progressive des versions dans le fichier « composer.json ».

###### Erreurs liées à l’environnement serveur

Un autre problème courant survient lorsque les dépendances s'installent correctement en local, mais échouent sur le serveur distant. Cela arrive souvent lorsque les versions de PHP diffèrent entre les deux environnements.

* **Solution technique :**

Ajouter une configuration spécifique dans le fichier composer.json pour forcer Composer à respecter la version de PHP du serveur cible. Cela permet d’éviter l’installation de packages incompatibles avec la version de PHP utilisée en production.

Exemple de configuration à ajouter dans « composer.json » :

"config": {

        "platform": {

            "php": "8.2.0"

        }

    },

Avec cette option, Composer tiendra compte de cette contrainte lors des mises à jour, simulant ainsi un environnement identique à celui du serveur distant.

###### Mise à jour d'autres packages

Une fois la migration vers Symfony 6.4 réalisée avec succès, il est pertinent de mettre à jour l’ensemble des bibliothèques externes utilisées dans le projet. Cette étape permet d’assurer une meilleure cohérence entre les différentes dépendances, tout en bénéficiant des dernières améliorations en matière de sécurité, de performance et de stabilité. Elle offre également l’opportunité de corriger d’éventuels bugs connus dans les versions précédentes.

La commande suivante a été utilisée pour effectuer cette mise à jour globale :

composer update

Celle-ci met à jour toutes les dépendances du projet vers leurs dernières versions stables compatibles, en respectant les contraintes définies dans le fichier composer.json.

Cette approche graduelle s’est avérée particulièrement utile dans le cas du projet Autossimo , notamment pour les paquets tels que laminas/laminas-code, laminas/laminas-eventmanager ou encore phpunit/phpunit, qui nécessitaient des mises à jour précises pour garantir leur compatibilité avec PHP 8.2 et Symfony 6.4.

Il est recommandé de réaliser cette mise à jour de manière progressive, surtout lorsque certaines bibliothèques présentent des risques de rupture potentielle. Par exemple, mettre à jour certains paquets spécifiques en isolé :

composer update vendor/package1 vendor/package2

Cela permet de garder le contrôle sur les modifications introduites et de tester chaque changement de manière ciblée, afin d’éviter toute régression imprévue.

#### Mise à jour des recettes flex

Au fil du temps, Symfony met à jour ces recettes. Parfois, ces mises à jour sont mineures… comme l’ajout d’un commentaire dans un fichier de configuration. Mais d’autres fois, elles sont plus importantes, comme le renommage de clés de configuration pour s’adapter aux changements dans Symfony lui-même. Et bien que vous ne soyez pas obligé de mettre à jour vos recettes, c’est un excellent moyen de garder votre application conforme aux standards d’une application Symfony. C’est aussi un moyen gratuit de mettre à jour le code déprécié. Je n’ai pas l’inclus dans la partie de « [Suppression des dépréciations](#_Suppression_des_dépréciations_1)» car pour le projet actuel, il n’aide pas vraiment à mettre à jour le code déprécié dans ce projet, mais il peut absolument fonctionne bien dans d’autres projets.

#### Vérification des recettes à mettre à jour

Pour afficher les recettes qui ont besoin d’une mise à jour, il suffit d’exécuter la commande suivante :

composer recipes:update

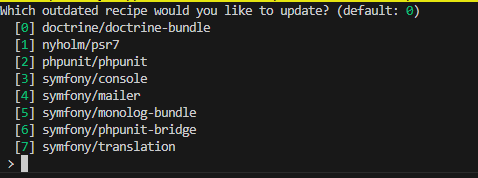


Figure : Sortie de la commande "compose recipes:update"

Un numéro est ensuite demandé pour choisir de mettre à jour la recette souhaitée. Il est recommandé de commencer par les recettes susceptibles d'apporter des modifications importantes, comme celles de « symfony/framework-bundle » ou « symfony/console ».

La commande composer recipes:update est intelligente : elle examine les différences entre la recette installée et la dernière version. Elle crée ensuite un correctif et l'applique à votre application. En cas de conflit, vous pouvez le résoudre comme un conflit Git classique et le commiter normalement.



#### Mise à jour le code pour qu'il fonctionne avec la nouvelle version.

Dans certains cas rares, la nouvelle version majeure peut introduire des ruptures de compatibilité ascendante (breaking changes). Il est donc essentiel de consulter le fichier UPGRADE-X.0.md (où X correspond à la nouvelle version majeure) fourni dans le dépôt officiel de Symfony, afin d’identifier toutes les modifications critiques à prendre en compte lors de la migration.

#### Conclusion

La migration vers PHP 8.2 et Symfony 6.4 a permis de moderniser l’environnement du projet Autossimo, tout en assurant la compatibilité du code avec les standards actuels. Grâce à une approche progressive, les dépendances critiques ont été mises à jour, les fonctionnalités obsolètes remplacées, et le projet adapté à son nouvel environnement cible. Cette étape valide la préparation technique à une migration complète et maitrisée de l’application.

## Conclusion

Ce chapitre a permis de structurer l’ensemble des étapes techniques préparatoires à la migration du projet Autossimo. À travers l’analyse de projets tests, l’évaluation des dépendances, la suppression des dépréciations, et l’adaptation de l’environnement vers PHP 8.2 et Symfony 6.4, les bases nécessaires à une migration fiable ont été établies. Ces travaux ont mis en évidence les enjeux de compatibilité, les choix d’outillage, et les ajustements à apporter au code existant. L’ensemble des constats et validations obtenus constitue une fondation solide pour la planification et l’exécution de la migration, qui seront détaillées dans la feuille de route du chapitre suivant.

# CHAPITRE 3 : STRATEGIE DE MIGRATION TECHNIQUE



## Introduction

Dans le cadre de l’évolution de l’application Autossimo, une phase de migration s’est avérée nécessaire afin d'assurer la pérennité, la performance et la maintenabilité de la solution. Cette migration peut répondre à divers objectifs : modernisation technologique, changement d’infrastructure, rationalisation des environnements ou encore optimisation des performances.

Ce chapitre présente tout d’abord une synthèse des principaux types de migration existants, en mettant en évidence leurs caractéristiques, avantages, contraintes et cas d’usage. Sur cette base, le choix d’un type de migration adapté au contexte du projet Autossimo est justifié, en tenant compte des impératifs techniques, organisationnels et fonctionnels identifiés en amont.

Enfin, une feuille de route détaillée est proposée afin de structurer et planifier les différentes étapes de la migration, depuis la phase préparatoire jusqu’à la mise en production. Cette approche vise à assurer une transition maîtrisée, en limitant les risques opérationnels tout en garantissant la continuité des services.



## Types de migrations

### Migration monolithique (One-shot)

La migration monolithique consiste à remplacer l’ensemble du système existant par la nouvelle version en une seule étape. Cela implique une coupure totale avec l’ancien système une fois la nouvelle version validée et déployée.

* **Avantages :**
* Simplicité conceptuelle : un seul cycle de test et déploiement.
* Absence de maintenance parallèle des deux versions.
* Gain de temps global si la migration est bien planifiée.
* **Inconvénients :**
* Risque élevé : toute erreur impacte immédiatement l’ensemble du système.
* Nécessite une interruption complète du service pendant le déploiement.
* Complexité accrue pour les grands systèmes.

### Migration progressive / incrémentale

Cette approche consiste à migrer progressivement certaines parties de l’application, tout en maintenant les autres sur la version précédente. Le système évolue donc par étapes successives.

* **Avantages :**
* Moindre risque global grâce à une validation par composants.
* Possibilité de maintenir la disponibilité du système pendant la migration.
* Meilleure gestion des retours utilisateurs sur chaque phase.
* **Inconvénients :**
* Complexité technique liée à la coexistence temporaire de deux versions.
* Besoin accru de coordination entre les équipes.
* Période de migration plus longue.

### Migration parallèle (Dual Maintenance)

Il s’agit de faire coexister les deux versions du système pendant une période donnée. Les utilisateurs peuvent être dirigés vers l’une ou l’autre version selon des critères prédéfinis (fonctionnalités testées, profils utilisateurs, etc.).

* **Avantages :**
* Faible risque opérationnel : possibilité de revenir immédiatement à l’ancienne version.
* Test grandeur nature de la nouvelle version avec des données réelles.
* Transition progressive sans perturbation majeure.
* **Inconvénients :**
* Coût infrastructurel accru : double déploiement nécessaire.
* Charge de maintenance doublée pendant la période de transition.
* Gestion complexe de la synchronisation des données entre les deux versions.

### Pattern Strangler Fig (Étouffoir)

Inspirée de la croissance de l’arbre "figuier étrangleur", cette méthode consiste à remplacer progressivement des parties du système ancien par des versions nouvelles et modernisées, jusqu’à ce que le système initial soit complètement remplacé.

* **Avantages :**
* Permet une évolution continue sans rupture globale.
* Facilite l'intégration continue et les itérations rapides.
* Réduit les risques liés à la migration globale.
* **Inconvénients :**
* Exige une architecture modulaire et bien découplée.
* Peut prendre beaucoup de temps si la migration est incomplète.
* Complexité accrue durant la phase transitoire.

### Migration Blue-Green (ou basculement)

Cette stratégie repose sur le déploiement simultané de deux environnements : un "bleu" (version actuelle) et un "vert" (version cible). Le trafic est redirigé d’un environnement à l’autre après validation de la nouvelle version.

* **Avantages :**
* Temps d’arrêt minimal voire inexistant.
* Facilité de rollback : il suffit de rediriger le trafic vers l’environnement précédent.
* Bonne intégration avec les outils DevOps modernes.
* **Inconvénients :**
* Coût matériel ou cloud accru.
* Nécessite une automatisation poussée du déploiement.
* Doit gérer la cohérence des données entre les deux environnements.

### Migration basée sur des Feature Toggles

L’utilisation de Feature Flags permet d’activer ou désactiver dynamiquement certaines fonctionnalités en fonction de leur niveau de migration ou de leur stabilité.

* **Avantages :**
* Contrôle granulaire sur l’exposition des nouvelles fonctionnalités.
* Possibilité de tester en production de manière sécurisée.
* A/B testing possible entre anciennes et nouvelles versions.
* **Inconvénients :**
* Complexification du code source avec gestion de multiples branches logiques.
* Risque d’accumulation de flags obsolètes si non nettoyés.
* Nécessite une bonne gouvernance des configurations.

## Le choix de migration

Dans le contexte du projet Autossimo, il a été décidé d’opter pour une migration progressive plutôt qu’une refonte complète en mode One-Shot . En effet, bien que l’approche monolithique permette une transition rapide vers la nouvelle version, elle présente un niveau de risque élevé, notamment en raison de la complexité du code existant et de l’interdépendance des modules. De surcroît, cette méthode implique souvent une période prolongée d’indisponibilité ou une charge importante en phase de test préalable, ce qui n’est pas compatible avec les contraintes opérationnelles liées à l’application.

Par ailleurs, bien que la migration progressive soit techniquement plus complexe à mettre en œuvre, elle offre un meilleur équilibre entre maîtrise des risques , continuité de service et gestion des impacts utilisateur. Elle permet de migrer l’application module par module, en validant chaque étape grâce à un ensemble de tests fonctionnels automatisés , assurant ainsi la stabilité globale du système tout au long du processus. Cette démarche garantit une évolution maîtrisée de l’architecture, notamment lors de la mise à jour technique telle que la migration de Symfony 5.4 vers 6.4. Ainsi, en s’appuyant sur des tests rigoureux et une planification structurée, la migration progressive apparaît comme la solution la plus adaptée pour moderniser l’application tout en préservant sa disponibilité et sa fiabilité.

## Feuille de route

La migration du projet Autossimo vers Symfony 6.4 et PHP 8.2 implique une planification rigoureuse afin d’assurer la continuité fonctionnelle, la stabilité applicative et la maintenabilité du code. La feuille de route ci-dessous présente les différents chantiers techniques à mener, les outils mobilisés, ainsi que les étapes de validation progressive jusqu’à la mise en production.



### Traitement des interfaces dépréciées

Certaines interfaces ou services utilisés dans Autossimo sont dépréciés dans les versions récentes de Symfony, comme :

* SessionInterface pour la gestion des sessions ;
* L’utilisation directe du conteneur de services (container->get()) ;
* D'autres composants liés à la gestion des événements, de la requête ou des services internes.

Un chantier sera ouvert pour cartographier l’utilisation de ces interfaces, proposer les alternatives recommandées et effectuer les modifications nécessaires. Ces changements seront réalisés sans impacter les fonctionnalités métier.

### Spike d’analyse des dépréciations

Un spike technique sera mis en place afin d’identifier de façon exhaustive toutes les dépréciations présentes dans le code ou dans les dépendances. Ce spike comprendra :

* L’analyse du Symfony Profiler à l’exécution ;
* L’usage d’outils d’analyse statique tels que Rector ou PHPStan ;
* L’identification des dépendances obsolètes via Composer (composer outdated).

Chaque dépréciation identifiée donnera lieu à la création d’un ticket technique, incluant:

* L’élément concerné (fichier, service, classe) ;
* La criticité de l’impact ;
* Une estimation de charge pour la résolution.

### Migration des packages obsolètes

Plusieurs packages utilisés dans Autossimo ne sont plus compatibles avec Symfony 6.4 ou ne sont plus maintenus. Parmi les plus critiques :

* csa/guzzle-bundle → à remplacer par symfony/http-client ;
* trikoder/oauth2-bundle → à migrer vers league/oauth2-server-bundle ;
* swiftmailer-bundle → à remplacer par symfony/mailer.

Pour chaque package, un chantier individuel sera mis en œuvre, incluant un spike d’analyse, la documentation du remplacement proposé, la mise à jour du code, et les tests fonctionnels de validation.

### Résolution des problèmes post-migration

Malgré la rigueur du processus, des effets de bord ou anomalies résiduelles peuvent survenir. Une phase de correction post-migration est donc prévue, visant à :

* Traiter les erreurs de configuration ou de dépendances non détectées ;
* Corriger les écarts de comportement liés à des changements internes du framework ;
* Finaliser l'ajustement de l’environnement (Docker, services externes…).

### TNR manuel de l’application

Avant toute mise en production, un Test de Non-Régression (TNR) manuel sera effectué. Ce TNR aura pour objectif de garantir que :

* Toutes les fonctionnalités existantes sont toujours opérationnelles ;
* Aucune régression fonctionnelle n’a été introduite ;
* L'application est stable, conforme et exploitable en production.

Le plan de test couvrira à la fois les modules techniques (authentification, appels API) et les modules métier spécifiques à l’environnement Autossimo.

## Conclusion

La stratégie de migration définie pour le projet Autossimo repose sur une approche progressive et maîtrisée, axée sur une montée de version technique sans refonte fonctionnelle. Le choix du type de migration, la structuration en chantiers ciblés (interfaces dépréciées, packages obsolètes) ainsi que l’intégration de phases d’analyse et de validation successives permettent d’encadrer le processus tout en minimisant les risques. Cette feuille de route assure une transition vers Symfony 6.4 et PHP 8.2 dans des conditions optimales, tout en garantissant la stabilité, la maintenabilité et la conformité de l’application avant son déploiement final.

# Conclusion générale

Ce projet de fin d’études avait pour objectif principal l’étude approfondie de la migration d’un projet Symfony 5.4 vers Symfony 6.4, dans un contexte réel d’évolution technique et de maintien de la qualité logicielle. La démarche adoptée s’est voulue progressive, méthodique et orientée vers la maîtrise des risques liés à une montée de version majeure.

Dans un premier temps, une migration expérimentale a été conduite sur un projet Symfony 5.4 minimal afin de valider, dans un environnement maîtrisé, les étapes techniques nécessaires à la transition vers Symfony 6.4. Cette phase a permis d’identifier les principales dépréciations du framework, de tester des outils d’automatisation tels que Rector, et de construire une première approche méthodologique.

Par la suite, le fichier composer.json du projet Autossimo a été intégré dans ce projet de test afin d’isoler et d’analyser les dépendances réelles. Cette phase a révélé un ensemble de dépendances obsolètes ou abandonnées, dont certaines incompatibles avec les versions cibles de Symfony et PHP. Enfin, une analyse complète du code Autossimo a été menée, permettant de dresser un inventaire détaillé des dépréciations, d’évaluer les impacts techniques, et de consolider les prérequis à une migration future.

Ces travaux ont permis d’obtenir une vision claire des adaptations nécessaires pour garantir la compatibilité de l’application avec Symfony 6.4. Ils ont contribué à l’identification de l’ensemble des freins techniques à lever et ont fourni les fondations d’un plan de migration structuré.

Ce projet m’a permis de mobiliser et de renforcer un ensemble de compétences, à la fois techniques et méthodologiques, telles que la gestion des tâches techniques, l’analyse structurée du code, la migration logicielle, l’audit de dépendances, ainsi que la maîtrise du framework Symfony et du langage PHP.

La mise en production du projet n’a pas été abordée dans le cadre de ce PFE, celle-ci relevant d’un périmètre ultérieur à la phase de préparation. Toutefois, une feuille de route opérationnelle a été définie en conclusion du travail, détaillant les différents chantiers techniques à engager dans la suite du projet : traitement des interfaces dépréciées, remplacement des packages obsolètes, analyse finale de stabilité, correction post-migration et phase de test de non-régression avant déploiement.

Ainsi, cette étude constitue un socle solide pour accompagner, de manière structurée et maîtrisée, la modernisation technique de l’application Autossimo.

# WEBOGRAPHIE

* <https://www.autodistribution.fr/>
* La définition de Slack: <https://en.wikipedia.org/wiki/Slack_(software)>
* La définition de MS Teams: <https://en.wikipedia.org/wiki/Microsoft_Teams>
* La définition de Jira: <https://en.wikipedia.org/wiki/Jira_(software)>
* La définition de Confluence: <https://www.kolekti.com/resources/blog/what-is-confluence>
* La définition de Symfony: <https://www.pure-illusion.com/lexique/definition-de-symfony>
* La définition de Git: <https://git-scm.com/>
* La définition de Rector: <https://getrector.com/>
* La définition du PHP: <https://www.php.net/manual/en/introduction.php>
* La définition du PHPUnit: <https://en.wikipedia.org/wiki/PHPUnit>
* La définition de Gitlab: <https://gitlab.com/gitlab-org/gitlab>
* La définition de Composer: <https://getcomposer.org/doc/00-intro.md>
* La définition de Docker: <https://docs.docker.com/get-started/docker-overview/>
* La définition de VSCode: <https://fr.wikipedia.org/wiki/Visual_Studio_Code>
* La définition de Redis: <https://fr.wikipedia.org/wiki/Redis>
* La définition de JS: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
* La définition de GNU Make: <https://www.gnu.org/software/make/>
* La définition d'Oracle DB: <https://fr.wikipedia.org/wiki/Oracle_Database>
* La définition du Kanban: [https://www.atlassian.com/agile/kanban/boards#:~:text=A%20kanban%20board%20is%20an,order%20in%20their%20daily%20work.](https://www.atlassian.com/agile/kanban/boards%23:~:text=A%20kanban%20board%20is%20an,order%20in%20their%20daily%20work.)
* Le versionnage: <https://medium.com/@shilpasyal55/demystifying-software-versioning-understanding-patch-minor-and-major-versions-11927673f3d3>
* La recette [: https://symfonycasts.com/screencast/symfony5/flex](:%20https:/symfonycasts.com/screencast/symfony5/flex)
* La définition d’une dépréciation : [https://moodledev.io/general/development/policies/deprecation#:~:text=Deprecation%2C%20in%20its%20programming%20sense,so%20may%20cause%20regression%20errors.](https://moodledev.io/general/development/policies/deprecation%23:~:text=Deprecation%2C%20in%20its%20programming%20sense,so%20may%20cause%20regression%20errors.)