

****

**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

Autossimo :  
Migration de Symfony version 5.4 vers 6.4

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 |
| PHP | Hyper-Text Pre Processor |
| Redis | REmote DIctionary Server |
|  |  |
|  |  |
|  |  |
| WS | Web Service |

# *Résumé*

Ce rapport présente les travaux réalisés lors de mon projet de fin d'études au sein du GROUPE SQLI.

# *Abstract*

This report outlines the work conducted during my graduation project at SQLI GROUP

# *ملخص*

SQLI يتناول هذا التقرير العمل الذي قمت به أثناء مشروع تخرجي في مجموع

# Sommaire

[Dédicaces 3](#_Toc201068151)

[*Remerciements* 4](#_Toc201068152)

[Tableau des abréviations 5](#_Toc201068153)

[*Résumé* 6](#_Toc201068154)

[*Abstract* 7](#_Toc201068155)

[*ملخص* 8](#_Toc201068156)

[Sommaire 9](#_Toc201068157)

[Table des figures 11](#_Toc201068158)

[*Introduction générale* 12](#_Toc201068159)

[CHAPITRE 1 : PRESENTATION DU PROEJT 13](#_Toc201068160)

[1.1. Introduction : 14](#_Toc201068161)

[1.2. Présentation de l’entreprise d’accueil : 14](#_Toc201068162)

[1.2.1. Présentation du Groupe SQLI : 14](#_Toc201068163)

[1.3. Présentation du contexte du projet « Autossimo v3 » : 17](#_Toc201068164)

[1.3.1. Présentation du projet « Autossimo v3 » 18](#_Toc201068169)

[1.3.2. Objectifs d’Autossimo 19](#_Toc201068170)

[1.3.3. Description de l’environnement du projet 20](#_Toc201068171)

[1.4. Présentation du projet de fin d’études 21](#_Toc201068172)

[1.4.1. Etude de l’existant 21](#_Toc201068173)

[1.4.2. Identification des problématiques 23](#_Toc201068174)

[1.4.3. Solution proposée par le projet 24](#_Toc201068175)

[1.5. Planification et conduite du projet 24](#_Toc201068176)

[1.5.1. Méthodologie SCRUM 24](#_Toc201068177)

[1.5.2. Tableau Kandan 27](#_Toc201068178)

[1.5.3. Les outils utilisés 27](#_Toc201068179)

[1.5.4. Conclusion 29](#_Toc201068180)

[CHAPITRE 2 : ANALYSE ET ETUDE 31](#_Toc201068181)

[2.1. Introduction : 32](#_Toc201068183)

[2.1.1. Périmètre fonctionnel du projet 32](#_Toc201068187)

[2.2. Le contrôle de version 33](#_Toc201068188)

[2.3. Étapes de migration du projet Autossimo de Symfony 5.4 vers 6.4 35](#_Toc201068189)

[2.3.1. Suppression des dépréciations 35](#_Toc201068190)

[2.3.2. Gestion des dépréciations avec PHPUnit 43](#_Toc201068191)

[2.3.3. Automatisation partielle avec Rector 43](#_Toc201068192)

[2.3.4. Les packages abandonnées et incompatible : 45](#_Toc201068193)

[2.3.5. Passage à PHP 8.2 47](#_Toc201068194)

[2.3.6. Utilisations des nouvelles fonctionnalités de PHP 8 50](#_Toc201068195)

[2.3.7. Mise à jour de Symfony 5.4 vers 6.4 51](#_Toc201068196)

[2.3.8. Mise à jour des recettes flex 56](#_Toc201068197)

[2.3.9. Mettez à jour le code pour qu'il fonctionne avec la nouvelle version. 57](#_Toc201068208)

[2.3.10. Les outils utilisés 57](#_Toc201068209)

[2.3.11. Etudes préliminaires réalisées 57](#_Toc201068210)

[2.3.12. Conclusion 58](#_Toc201068211)

[2.4. Outils rencontrés ou utilisés 59](#_Toc201068212)

[2.4.1. Langages de programmation 59](#_Toc201068213)

[2.4.2. Bibliothèques et Frameworks : 60](#_Toc201068214)

[2.4.3. Outils de développement 61](#_Toc201068215)

[2.4.4. Système de Gestion de base de données 64](#_Toc201068216)

[CHAPITRE 3 : TECHNOLOGIES ET OUTILS UTILISES ET RENCONTRES 65](#_Toc201068217)

[5.1. Introduction 66](#_Toc201068221)

[5.2. Conclusion 66](#_Toc201068222)

[CHAPITRE 4 : REALISATION DU PROJET 67](#_Toc201068223)

[3.3. Introduction 68](#_Toc201068224)

[3.3.1. Migration réussie d’un projet test 68](#_Toc201068225)

[3.3.2. Réutilisation du composer.json du projet réel 68](#_Toc201068226)

[3.3.3. Tentative de migration du projet réel 69](#_Toc201068227)

[3.3.4. Obstacles rencontrés 69](#_Toc201068228)

[3.3.5. Solutions envisagées 70](#_Toc201068229)

[3.3.6. Conclusion de la réalisation 70](#_Toc201068230)

# Table des figures

[Figure 1: Logo de SQLI 14](#_Toc200534996)

[Figure 2 : Répartition des agences 15](#_Toc200534997)

[Figure 3 : Partenaires de SQLI 16](#_Toc200534998)

[Figure 4: Organigramme de SQLI Maroc 17](#_Toc200534999)

[Figure 5 : Logo Autossimo 17](#_Toc200535000)

[Figure 6 : Canaux de distribution 19](#_Toc200535001)

[Figure 7: Architecture Autossimo v3 22](#_Toc200535002)

[Figure 8: 3 piliers de Scrum 24](#_Toc200535003)

[Figure 9: Processus de Scrum 25](#_Toc200535004)

[Figure 10: Logo de Jira 27](#_Toc200535005)

[Figure 11: Logo de Microsoft Teams 28](#_Toc200535006)

[Figure 12: Logo de Slack 29](#_Toc200535007)

[Figure 13: Logo de Confluence 29](#_Toc200535008)

[Figure 14: Logo de PHP 33](#_Toc200535009)

[Figure 15: Logo de JS 33](#_Toc200535010)

[Figure 16: Logo de Symfony 34](#_Toc200535011)

[Figure 17: Logo de PHPUnit 34](#_Toc200535012)

[Figure 18: Logo de Rector 35](#_Toc200535013)

[Figure 19: Logo de Git 36](#_Toc200535014)

[Figure 20: Logo de Gitlab 36](#_Toc200535015)

[Figure 21: Logo de Composer 37](#_Toc200535016)

[Figure 22: Logo de Docker 37](#_Toc200535017)

[Figure 23: Logo de VSCode 38](#_Toc200535018)

[Figure 24: Logo de GNU Make 38](#_Toc200535019)

[Figure 25: Logo de Oracle DB 39](#_Toc200535020)

[Figure 26: Logo de Redis 39](#_Toc200535021)

# *Introduction générale*

L’évolution rapide des technologies de l’information a profondément transformé le paysage économique mondial. Dans ce contexte, les entreprises sont appelées à adopter des solutions numériques performantes pour maintenir leur compétitivité, répondre aux attentes croissantes des clients et optimiser leur efficacité opérationnelle. La digitalisation s’impose ainsi comme un levier stratégique permettant de supprimer les contraintes spatio-temporelles et d’offrir une expérience utilisateur fluide, réactive et personnalisée.

Dans le secteur de l’après-vente automobile, cette transition digitale se traduit notamment par l’adoption de plateformes e-commerce B2B, capables de simplifier les interactions entre les professionnels, d’optimiser les chaînes d’approvisionnement et de renforcer la relation client. Autodistribution, leader de la distribution indépendante de pièces détachées automobiles et poids lourds en France, s’inscrit pleinement dans cette dynamique à travers sa plateforme Autossimo, solution e-commerce B2B développée pour répondre aux besoins de ses partenaires.

En partenariat avec SQLI, acteur reconnu dans l’accompagnement digital des entreprises, la plateforme Autossimo poursuit un processus continu d’amélioration et de modernisation. C’est dans ce cadre que s’inscrit le présent projet de fin d’études, réalisé au sein de l’agence SQLI-Oujda, qui a pour objectif principal la migration du cœur applicatif de la plateforme vers une version plus récente et plus robuste, tout en assurant la fiabilité de ses fonctionnalités à travers des tests automatisés de non-régression.

Ce travail a ainsi consisté à participer aux évolutions techniques du projet Autossimo, à travers des interventions sur le core Symfony de l’application et la mise en place de mécanismes de validation automatisée des régressions fonctionnelles. Il s’inscrit dans une logique d’amélioration continue du produit, répondant aux exigences croissantes en matière de performance, de sécurité et de maintenabilité.

Ce rapport est structuré en quatre chapitres. Le premier présente l’organisme d’accueil et le contexte global du projet. Le second est consacré à l’analyse et conception. Le troisième décrit l’environnement technique utilisé et rencontrés. Le quatrième détaille la phase de réalisation, en mettant l’accent sur les travaux de migration, les tests mis en place ainsi que les résultats obtenus. Le document se clôture par une conclusion synthétique, suivie d’annexes et de références complémentaires.

# 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 1: 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 2 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 3 : Partenaires de SQLI

#### Expertises clés du Groupe SQLI :

* Conseil, Gouvernance, Stratégie digitale, Conduite du changement, Lab Innovation, Agilisation.
* Expérience design, Marketing digital, Stratégie d’acquisition et fidélisation, Web Analytics, Social media.
* E-commerce, Commerce unifié, Commerce Omnicanal, Digital in store, PIM et DAM.
* Solutions Web et Mobiles.
* Digital Workplace, Management de l’innovation.
* Digitalisation des processus, Solutions Achats, Finance, RH, Business Intelligence, RGPD.
* Big Data, Mobilité, Objets connectés, Maintenance en condition opérationnelle.
* Socle digital, Stratégie SI, APIsation.

#### 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 :

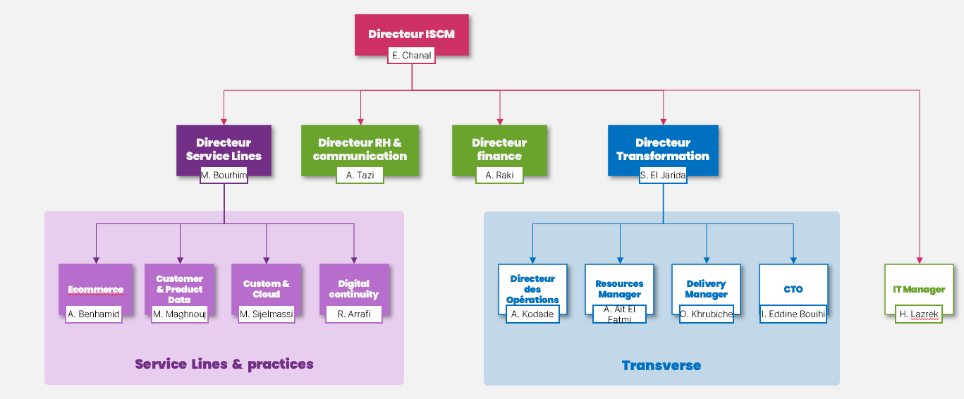


Figure 4: Organigramme de SQLI Maroc

## Présentation du client 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 5 : 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 6 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.

L’offre catalogue en ligne Autossimo propose :

* Plus de 2 100 000 de références : 78 marques, 40 000 modèles, 530 familles.
* Une gamme de pièces automobiles couvrant la majorité du parc véhicules à réparer ou à entretenir.
* Une recherche par numéro d’immatriculation.
* Un panier de commande visible et modifiable à tout moment.
* Un accès facile aux caractéristiques des pièces voiture, avec des éclatés images précis.
* Une visualisation du stock distributeurs.

Il englobe également des bases de données techniques mises à jour en permanence, simples d’utilisation, couvrant un pourcentage important du parc multimarque à un coût adorable.

Il propose des applications pour faciliter la gestion de votre activité :

* L’accès à un outil d’élaboration des devis de réparation automobile simple.
* L’accès aux temps barémés, tout en permettant de contrôler des marges par pièce détachée.
* Un module permettant de relancer les devis par SMS et email.

Des outils pour la promotion des services des garagistes et la vente de VO/VN, permettant aux réparateurs de :

* Communiquer avec leurs clients.
* Mettre en avant leurs prestations réparation auto.
* Mettre en avant leurs VO (Véhicules d’Occasion).
* Proposer des VO (Véhicules d’Occasion) et des VN (Véhicules Neufs) à leurs clients.

### Description de l’environnement du projet

Le projet Autossimo communique avec plusieurs composants :

* **Extranet :** Il s’agit d’un outil interne à Autodistribution et ses distributeurs pour le paramétrage du business, Il y en a 2 : l’Extranet Centrale et l’Extranet Distributeur.
* **Extranet Centrale :** C’est un outil d’administration business pour la gestion des clients (création, modification des données, …), gestion des réseaux, gestion des distributeurs…
* **Extranet Distributeur :** un outil d’administration pour la gestion des clients, gestion  
  des magasiniers, gestion des collections distributeurs, gestion des commandes. C’est un outil dédié pour les distributeurs.
* **CORE Service :** permet de récupérer les prix et le stock d’un ensemble de pièces pour un client et un dépôt donné , ainsi que la récuperation des consommables.
* **SupRef :** Fournir les données des catalogues pièces, consommables et Cora sous forme des fichiers texte pour les charger dans la base de données Autossimo chaque dimanches.

#### Flux de données :

Supref prépare les données à charger en BDD Oracle, pour que des scripts soient lancés pour l’indexation, Pour extraire les données Core Catalogue interroge premièrement la base ElasticSearch. En cas d’erreur (défaillance du serveur elastic, manque d’indexes …) on bascule au mode dégradé (mode base de données). Il existe des WS qui communiquent directement avec la base de données.

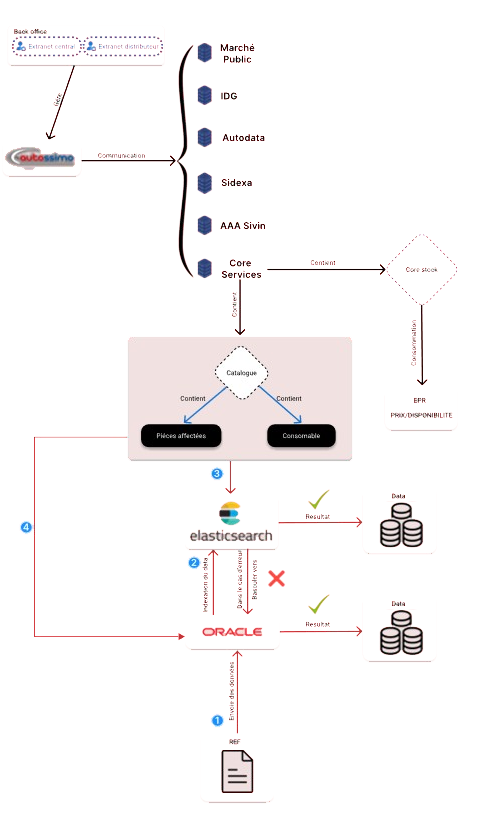


Figure 7: Architecture Autossimo v3

## Présentation du projet de fin d’études (Contexte su projet)

### Etude de l’existant

Le projet autossimo actuelle, fonctionne dans un ecosystéme PHP version 7.3 et le framework symfony 5.4, et utilise une base de donnée oracle 19 et systéme de cache via redis, rabbitMQ comme transport

* **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
  + Authentification / Autorisation : SecurityBundle + OAuth2 via trikoder/oauth2-bundle
  + Envoi d’e-mails : symfony/swiftmailer-bundle
  + Appels externes : csa/guzzle-bundle pour les requêtes HTTP
  + Sessions : Gérées via Redis en environnement de production
  + Interface utilisateur : Templates Twig dynamiques
  + Tests : PHPUnit + tests unitaires et fonctionnels
  + Infrastructure : Déploiement via Docker
* Architecture générale :
  + Utilisation de contrôleurs classiques et de routes définies via annotations
  + Services métiers centralisés et injectés via le conteneur de services
  + Gestion avancée des rôles utilisateurs et des accès sécurisés
  + API REST sécurisée avec token OAuth2
  + Communication avec des systèmes externes via des WebServices
* **Organisation du code :**

Le projet suit une architecture typique d’une application Symfony :

* Contrôleurs (src/Controller)
* Services métiers (src/Service)
* Entités et repositories (src/Entity, src/Repository)
* Fichiers de configuration (config/packages, config/routes)
* Tests (tests/App)

Il inclut également de nombreux composants spécifiques liés aux besoins métier, tels que:

* Génération de devis et commandes
* Gestion des réparateurs et clients
* Authentification Okta
* Impersonation utilisateur
* Appels externes vers des systèmes ERP

### 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.

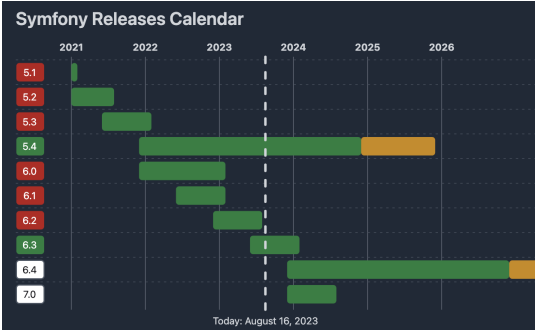


Figure 8: Calendrier des sorties de Symfony (1)

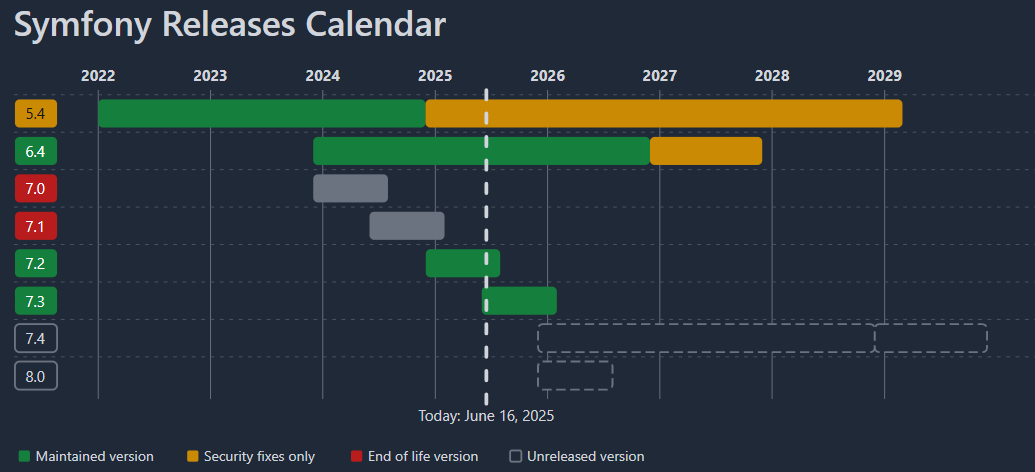


Figure 9: Calendrier des sorties de Symfony (2)

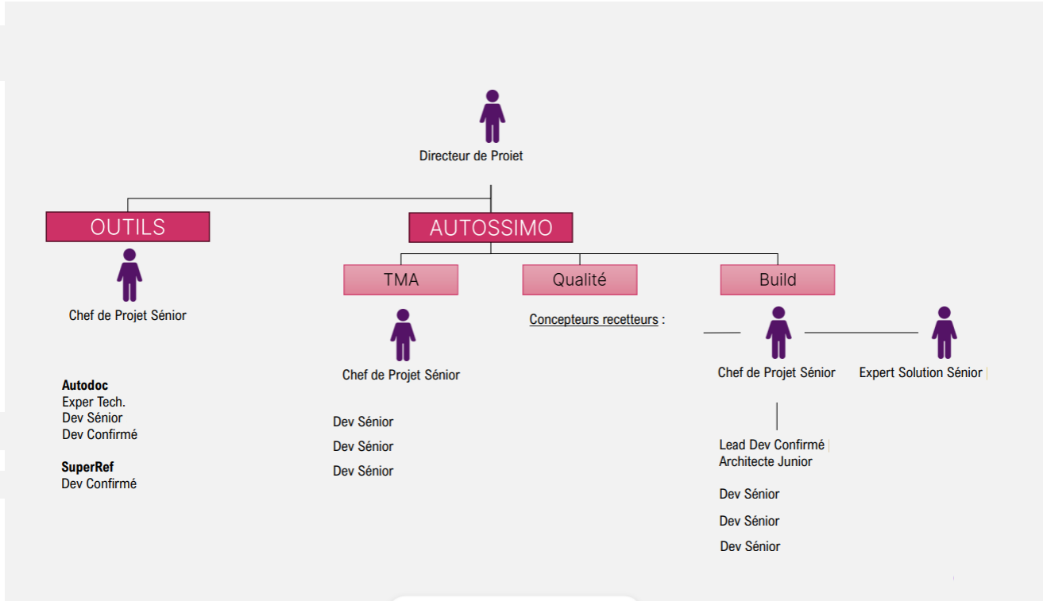
### Solution proposée par le projet

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.



### 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 :

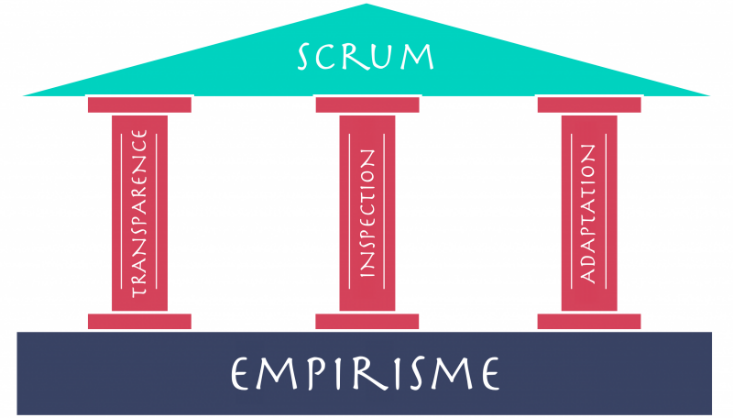


Figure 10: 3 piliers de Scrum

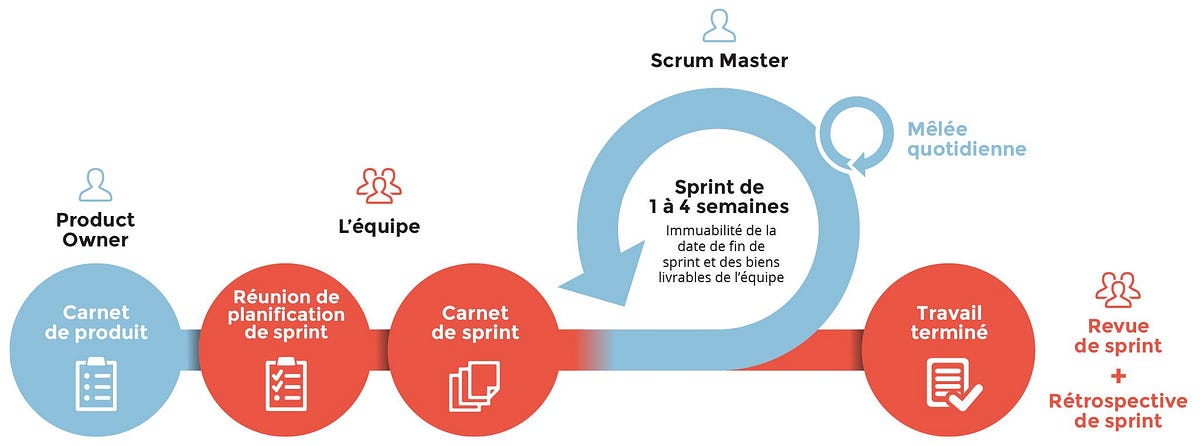


Figure 11: Processus de Scrum

* **Répartition des rôles :**

L’équipe Scrum est autoorganisée et multidisciplinaire, c'est-à-dire qu'elle choisit La meilleure façon d’effectuer le travail et elle possède toutes les compétences nécessaires pour mener à bien le projet. La flexibilité, la créativité et la productivité de l'équipe sont ainsi optimisées.

L’équipe Scrum se compose de :

**Le Scrum Master** est responsable de la compréhension, du respect et de la mise en œuvre de la méthode Scrum qu'il maîtrise parfaitement. Il veille au respect des principes et des valeurs de la méthodologie. C'est un facilitateur qui aide à améliorer la communication au sein des équipes et cherche à maximiser la productivité et l'expertise de l'équipe. Il est considéré comme l'entraîneur de l'équipe de développement.

**Le Product Owner** porte la vision du produit à réaliser. Il interagit avec l'équipe de développement qui doit suivre ses instructions. Il lui appartient de hiérarchiser les fonctionnalités à développer ou à corriger, et de vérifier les fonctionnalités complétées. Il est responsable de la gestion du backlog produit.

**L'équipe de développement** est chargée de traduire les exigences définies par le propriétaire du produit en fonctionnalités utilisables. Il est multidisciplinaire et possède toutes les compétences nécessaires pour exécuter un projet sans avoir recours à des services externes. Ses membres comprennent des architectes, des développeurs, des testeurs, etc. La taille idéale d'une équipe de développement est de 3 à 9 personnes. Il n'y a pas de notion de hiérarchie, toutes les décisions sont prises ensemble.

* **Les divers événements :**

Le cycle de vie d'un projet Scrum est cadencé par une série de réunions définies avec précision et limitées dans le temps.

**Le product backlog :** Il s'agit d'une liste hiérarchisée des exigences initiales du client concernant le produit à réaliser. Ce document évolue sans cesse durant le projet, en fonction des besoins du client. Le product owner est responsable du product backlog.

**Le Sprint backlog** **:** C'est le plan détaillé de la réalisation de l'objectif du Sprint, défini lors de la réunion de planification du Sprint. Le Sprint backlog est mis à jour régulièrement par l'équipe afin d'avoir une vision précise de la progression du Sprint.

**Planification du sprint :** Au cours de cette réunion, l'équipe de développement détermine les éléments prioritaires du Product Backlog (liste ordonnancée des exigences fonctionnelles et non fonctionnelles du projet) qu'elle pense pouvoir réaliser au cours du Sprint, en accord avec le Product Owner. Cette réunion permet donc à l'équipe d'établir les éléments qu'elle traitera au cours du Sprint et comment elle procédera.

**Sprint :** est une période courte et limitée dans le temps (2 semaines dans notre cas) où une équipe Scrum travaille pour compléter une quantité de travail définie. Les sprints sont au cœur même de Scrum et des méthodologies agiles, et obtenir les bons sprints aidera votre équipe agile à livrer de meilleurs logiciels avec moins de maux de tête.

**Mêlée quotidienne :** Cette réunion quotidienne de 15 minutes est très importante. Elle se fait debout (d'où son nom anglais de "stand-up meeting") afin d'éviter de s'éterniser et de permettre de rester agile. Le but est de faire un point sur la progression journalière du Sprint. Elle permet à l'équipe de synchroniser ses activités et de faire un plan pour les prochaines 24 heures. La mêlée a lieu à la même heure et au même endroit chaque jour.

**Revue de sprint :** consiste à démontrer le travail acharné de toute l'équipe : concepteurs, développeurs et Product Owner. Les membres de l'équipe se rassemblent autour d'un bureau pour des démonstrations informelles et décrivent le travail qu'ils ont effectué pour cette itération. C'est le moment de poser des questions, d'essayer de nouvelles fonctionnalités et de donner votre avis. Le partage du succès est un élément important de la construction d'une équipe agile.

**Rétrospective de sprint :** Après la revue du Sprint, cette réunion est l'occasion de déterminer ce qui peut être amélioré à la suite du Sprint écoulé (productivité, qualité, efficacité, conditions de travail, etc.). Cette réunion repose sur le principe d’amélioration continue.

**L'incrément :** Il s'agit de l'ensemble des éléments terminés du product backlog pour le Sprint en cours, ainsi que ceux des Sprints précédents. L'incrément doit fonctionner et être utilisable.

**Burndown Chart :** Ce graphique simple indique l'état d'avancement dans la réalisation des tâches du Sprint backlog. Il s’agit du tracé de la charge de travail restante (exprimée généralement en heures) en fonction du temps (en jours). Le Burndown Chart est actualisé tous les jours par le Scrum Master après la mêlée quotidienne.

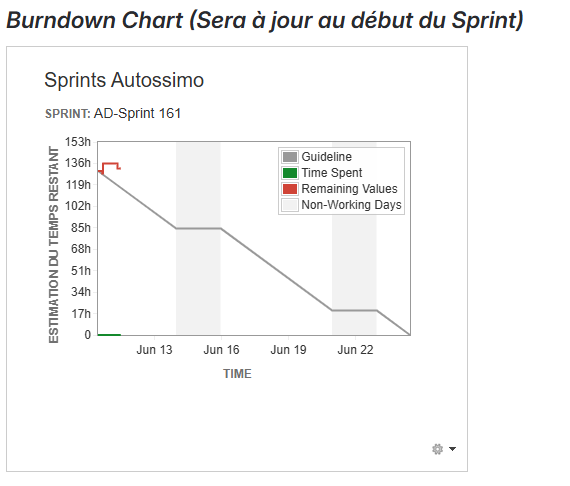


Figure 12: Example de Burndown Chart au début de Sprint

### 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 !

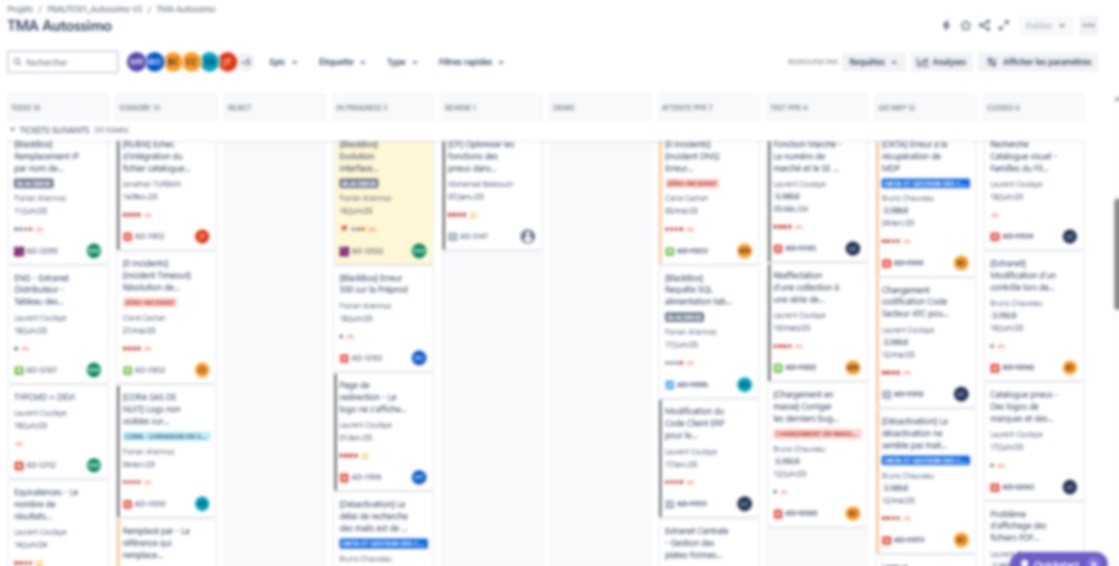


Figure 13: Exemple du workflow d'Autossimo

### 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 14: 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 15: 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 16: 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 17: 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

Dans ce chapitre, j’ai défini le contexte général de mon projet en présentant, tout d’abord, l’entreprise d’accueil (SQLI) qui accompagne ses clients dans la mise en œuvre et le pilotage de leur transformation digitale. J’ai également introduit le contexte de mon projet basé sur le Framework Symfony. Finalement, une présentation de la conduite du projet qui est une combinaison de JIRA et du Scrum.

# 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.

* + - onboarding

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 18: Aperçu des projets de Autossimo v3



### Périmètre fonctionnel du projet

Le projet Autossimo est une application métier utilisée par Autodistribution. Elle inclut plusieurs modules complexes tels que :

* Authentification et gestion des utilisateurs
* Gestion de session via Redis
* Appels externes (API REST) via Guzzle
* Envoi de mails avec SwiftMailer
* OAuth2 pour la sécurité
* Services métiers (commandes, devis, réparateurs)
* APIs sécurisées
* Templates Twig dynamiques
* Tests unitaires et fonctionnels
* La migration doit garantir la stabilité fonctionnelle de ces modules après passage à Symfony 6.4.

## Outils rencontrés ou utilisés

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

### Langages de programmation

#### PHP



Figure 19: 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 20: 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 21: 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 22: 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 23: 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 24: 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 25: 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 26: 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

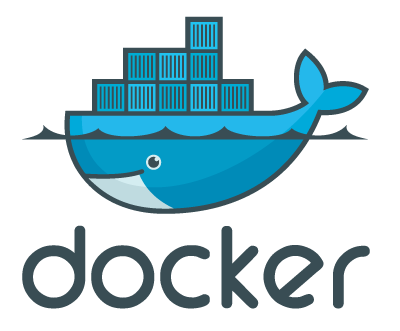


Figure 27: Logo de Docker

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

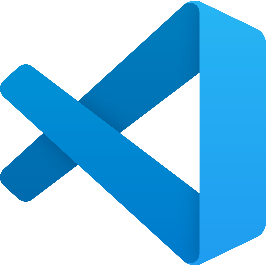


Figure 28: Logo de VSCode

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 29: Logo de 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 30: Logo de Oracle DB

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 31: 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

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 :

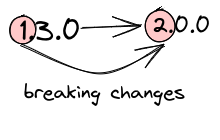


Figure 32: Example d'un changement d'une version majeure

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 :

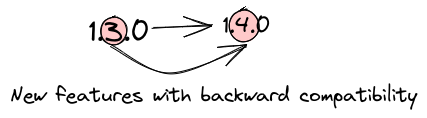


Figure 33: Example d'un changement d'une version mineure

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 :

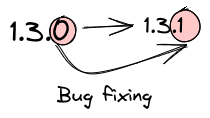


Figure 34: Example d'un changement d'une version de correctif

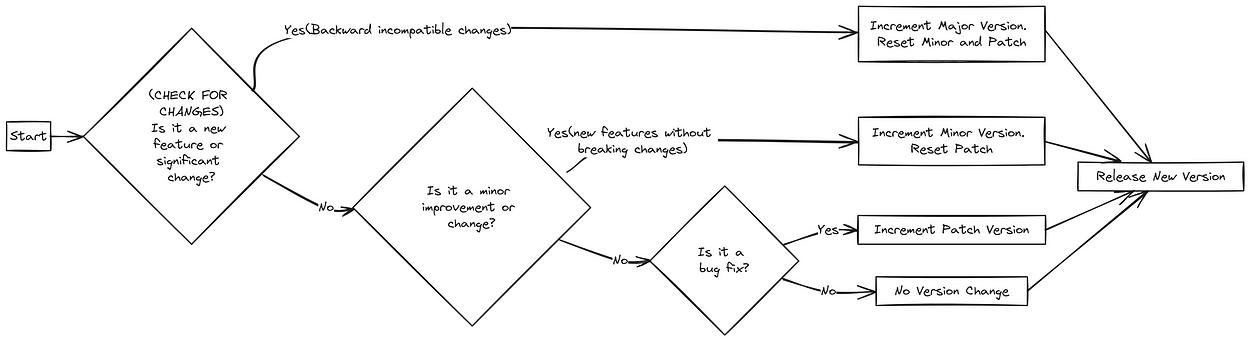
Une version de correctif, également appelée version de maintenance, désigne des mises à jour mineures apportant principalement des corrections de bugs et des améliorations ponctuelles. Ce type de mise à jour vise à résoudre des problèmes spécifiques sans modifier de manière significative le fonctionnement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

Figure 35: Oragnigramme du control de version

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

* **Le symbole Caret (^) :**

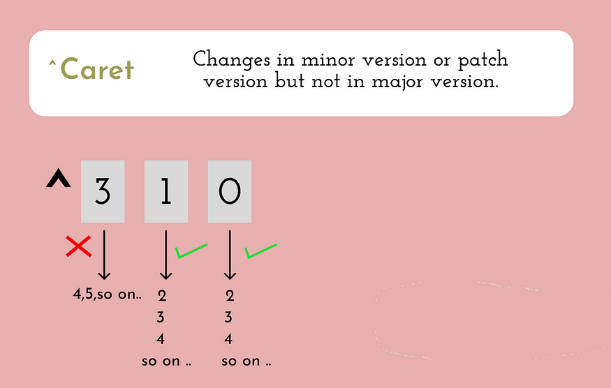


Figure 36: Le symbole Caret (^)

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 (~) :**

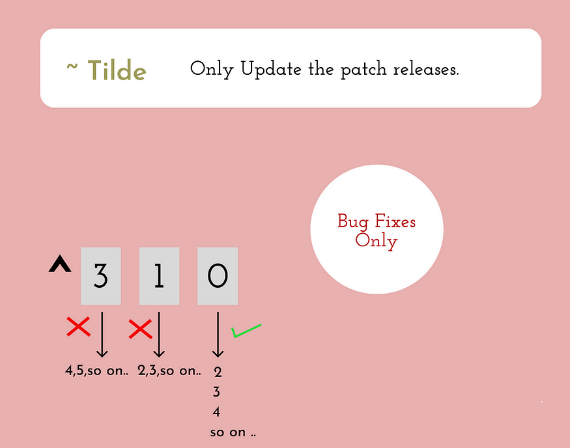


Figure 37: Le symbole Tilde (~)

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.

Introduction

Onborading

Formation

Présentation de projet

Documents : Travaux précédents

L’environnement de dev

Le processus d’analyse

Migration d’un Project 5.4 clean avec un CRUD simple (travail théorique)

Rector

Dépreciation

Remonté php 8

Remonté sf 6.4

* + - Conclusion migration bien passé

Migration le vendor de projet autossimo

Rector

Dépreciation

Remonté php 8

Remonté sf 6.4

Problème rencontré :

Package non compatible

Package obsolète

* + - Résultat

Analyse de code autossimo :

Rector

Dépreciation

Compatibilité php 8.2

….

* + - Conclusion

Conclusion

## L'approche suivie

Après l'attribution du projet, mon superviseur m'a recommandé un outil appelé Rector et m'a conseillé d'en savoir plus.

### Qu'est-ce que 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.

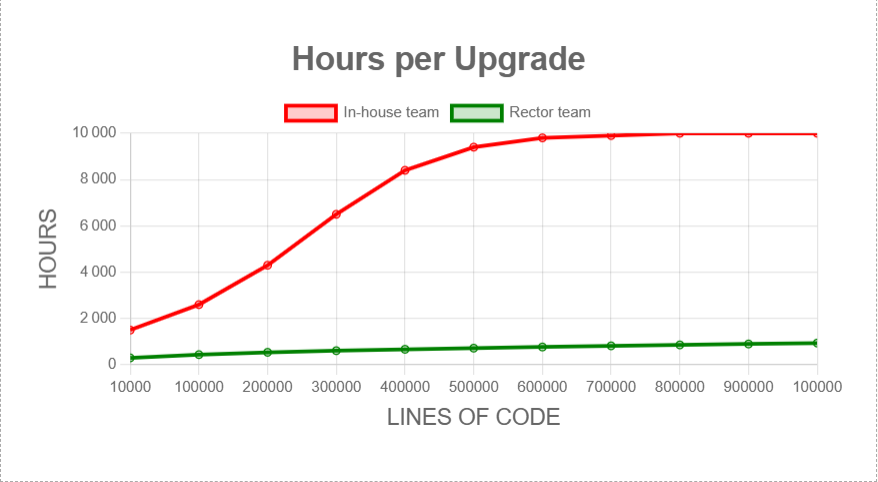


Figure 38: Comparaison en utilisant Rector contre sans l'utilisation de Rector

Mon superviseur a créé un dépôt dans Gitlab pour suivre ma progression. J'y ai créé une nouvelle branche appelée « Projet-fresh ». J'ai créé mon environnement Docker, ce qui consiste à créer plusieurs fichiers de configuration Dockerfile pour Nginx, le serveur web, php-fpm (l'interpréteur) et une base de données (postrges).

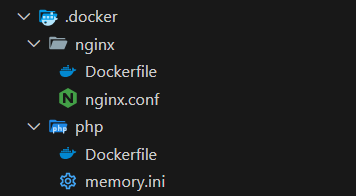


Figure 39: Structure du dossier .docker

Pour gérer et orchestrer tous ces packages, j'ai utilisé Docker Compose, intégré à Docker Desktop :

services:

  php:

    build: ./.docker/php

    volumes:

      - ./:/var/www/app

    expose:

      - 9000

    networks:

      - app-network

  webserver:

    build: ./.docker/nginx

    ports:

      - 8080:80

    volumes:

      - ./:/var/www/app

    networks:

      - app-network

    ###> doctrine/doctrine-bundle ###

  database:

    image: postgres:${POSTGRES\_VERSION:-16}-alpine

    environment:

      POSTGRES\_DB: ${POSTGRES\_DB:-app}

      # You should definitely change the password in production

      POSTGRES\_PASSWORD: ${POSTGRES\_PASSWORD:-!ChangeMe!}

      POSTGRES\_USER: ${POSTGRES\_USER:-app}

    volumes:

      - database\_data:/var/lib/postgresql/data:rw

    networks:

      - app-network

      # You may use a bind-mounted host directory instead, so that it is harder to accidentally remove the volume and lose all your data!

      # - ./docker/db/data:/var/lib/postgresql/data:rw

      ###< doctrine/doctrine-bundle ###

volumes:

  ###> doctrine/doctrine-bundle ###

  database\_data:

    ###< doctrine/doctrine-bundle ###

networks:

  app-network:

J'ai ensuite démarré un nouveau projet Symfony avec la version 5.4.\*, puis j'ai créé deux entités (Author et Book) pour simplifier les choses :

<?php

namespace App\Entity;

use App\Repository\AuthorRepository;

use Doctrine\Common\Collections\ArrayCollection;

use Doctrine\Common\Collections\Collection;

use Doctrine\ORM\Mapping as ORM;

/\*\*

 \* @ORM\Entity(repositoryClass=AuthorRepository::class)

 \*/

class Author

{

    /\*\*

     \* @ORM\Id

     \* @ORM\GeneratedValue

     \* @ORM\Column(type="integer")

     \*/

    private $id;

    /\*\*

     \* @ORM\Column(type="string", length=255)

     \*/

    private $name;

    /\*\*

     \* @ORM\OneToMany(targetEntity=Book::class, mappedBy="author", orphanRemoval=true)

     \*/

    private $books;

    public function \_\_construct()

    {

        $this->books = new ArrayCollection();

    }

    public function getId(): ?int

    {

        return $this->id;

    }

    public function getName(): ?string

    {

        return $this->name;

    }

    public function setName(string $name): self

    {

        $this->name = $name;

        return $this;

    }

    /\*\*

     \* @return Collection<int, Book>

     \*/

    public function getBooks(): Collection

    {

        return $this->books;

    }

    public function addBook(Book $book): self

    {

        if (!$this->books->contains($book)) {

            $this->books[] = $book;

            $book->setAuthor($this);

        }

        return $this;

    }

    public function removeBook(Book $book): self

    {

        if ($this->books->removeElement($book)) {

            // set the owning side to null (unless already changed)

            if ($book->getAuthor() === $this) {

                $book->setAuthor(null);

            }

        }

        return $this;

    }

}

<?php

namespace App\Entity;

use App\Repository\BookRepository;

use Doctrine\ORM\Mapping as ORM;

/\*\*

 \* @ORM\Entity(repositoryClass=BookRepository::class)

 \*/

class Book

{

    /\*\*

     \* @ORM\Id

     \* @ORM\GeneratedValue

     \* @ORM\Column(type="integer")

     \*/

    private $id;

    /\*\*

     \* @ORM\Column(type="string", length=255)

     \*/

    private $title;

    /\*\*

     \* @ORM\ManyToOne(targetEntity=Author::class, inversedBy="books")

     \* @ORM\JoinColumn(nullable=false)

     \*/

    private $author;

    public function getId(): ?int

    {

        return $this->id;

    }

    public function getTitle(): ?string

    {

        return $this->title;

    }

    public function setTitle(string $title): self

    {

        $this->title = $title;

        return $this;

    }

    public function getAuthor(): ?Author

    {

        return $this->author;

    }

    public function setAuthor(?Author $author): self

    {

        $this->author = $author;

        return $this;

    }

}

J'ai ensuite installé Rector dans cette même branche, comme indiqué dans la documentation :

composer require --dev rector/rector

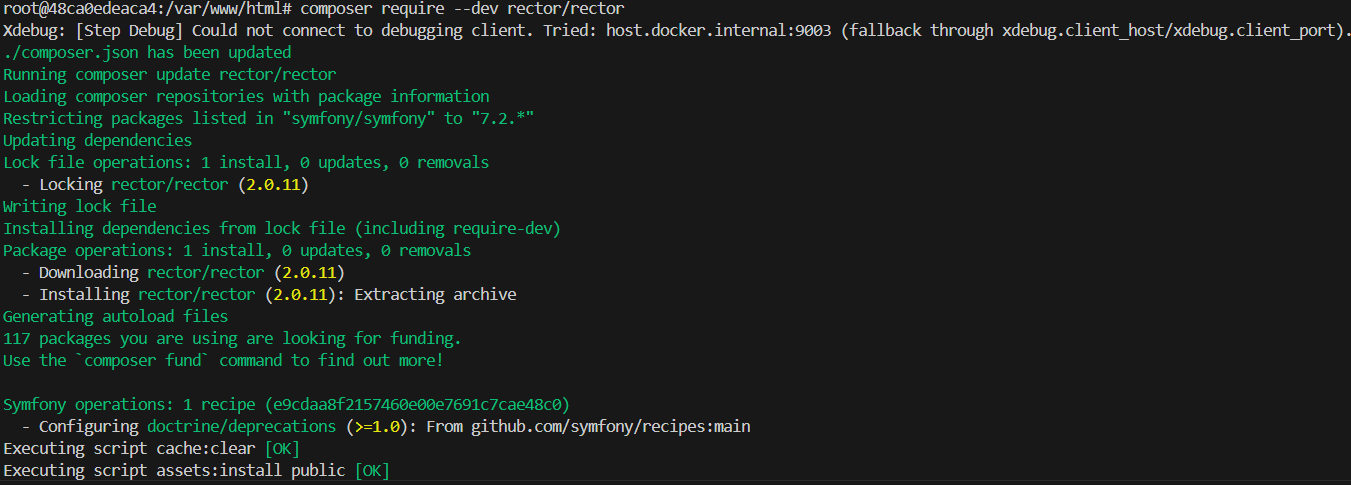


Figure 40: Installation du Rector (1)

J'ai ensuite initialisé le fichier de configuration de Rector avec : ./vendor/bin/rector. Un message m'a demandé si je souhaitais créer un nouveau fichier de configuration rector.php. J'ai accepté bien sûr :

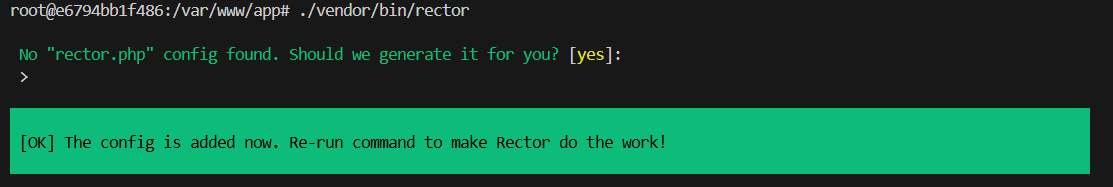


Figure 41: Initialisation de "rector.php"

Puis après avoir lu les articles en ligne et la documentation, je suis tombé sur la configuration suivante :

<?php

declare(strict\_types=1);

use Rector\Config\RectorConfig;

use Rector\Symfony\Set\SymfonySetList;

use Rector\Set\ValueObject\LevelSetList;

return RectorConfig::configure()

    ->withPaths([

        \_\_DIR\_\_ . '/config',

        \_\_DIR\_\_ . '/public',

        \_\_DIR\_\_ . '/src',

        \_\_DIR\_\_ . '/tests',

    ])

    ->withSets([

        SymfonySetList::SYMFONY\_50,

        SymfonySetList::SYMFONY\_51,

        SymfonySetList::SYMFONY\_52,

        SymfonySetList::SYMFONY\_53,

        SymfonySetList::SYMFONY\_54,

    ])

    ->withSets([

        LevelSetList::UP\_TO\_PHP\_81,

    ])

    // uncomment to reach your current PHP version

    ->withPhpSets()

    ->withTypeCoverageLevel(0)

    ->withDeadCodeLevel(0)

    ->withCodeQualityLevel(0);

**Configuration des Chemins :**

->withPaths([

        \_\_DIR\_\_ . '/config',

        \_\_DIR\_\_ . '/public',

        \_\_DIR\_\_ . '/src',

        \_\_DIR\_\_ . '/tests',

    ])

Cela signifie que Rector va analyser et refacturer :

* Le code source principal situé dans le dossier src/.
* Les tests unitaires situés dans le dossier tests/.

**Activation des Ensembles de Règles Symfony**

->withSets([

        SymfonySetList::SYMFONY\_50,

        SymfonySetList::SYMFONY\_51,

        SymfonySetList::SYMFONY\_52,

        SymfonySetList::SYMFONY\_53,

        SymfonySetList::SYMFONY\_54,

    ])

Ces ensembles permettent de migrer progressivement depuis Symfony 5.0 jusqu’à Symfony 5.4. Chaque ensemble contient des règles spécifiques à chaque version de Symfony.

**Activation des ensembles de règles PHP :**

    ->withSets([

        LevelSetList::UP\_TO\_PHP\_81,

    ])

Cela garantit que le code respecte les bonnes pratiques et les exigences de PHP 8.1.

Après avoir exécuté rector : « ./vendor/bin/rector process », j’ai obtenu le résultat suivant :



Figure 42: commit de "Rector resolu le conflit de RouteCollectionBuilder"

Rector a apporté de nombreuses améliorations, dont une dépréciation qui recommande l’utilisation de RoutingConfigurator au lieu de RouteCollectionBuilder.

### Dépréciation

Les dépréciations, ou code déprécié, sont du code qui sera bientôt supprimé dans la prochaine version.

Gérer le code obsolète est un pas important vers la création et la maintenance d’un code de qualité.

Pour visualiser toutes les dépréciations, nous pouvons utiliser le profileur Symfony :

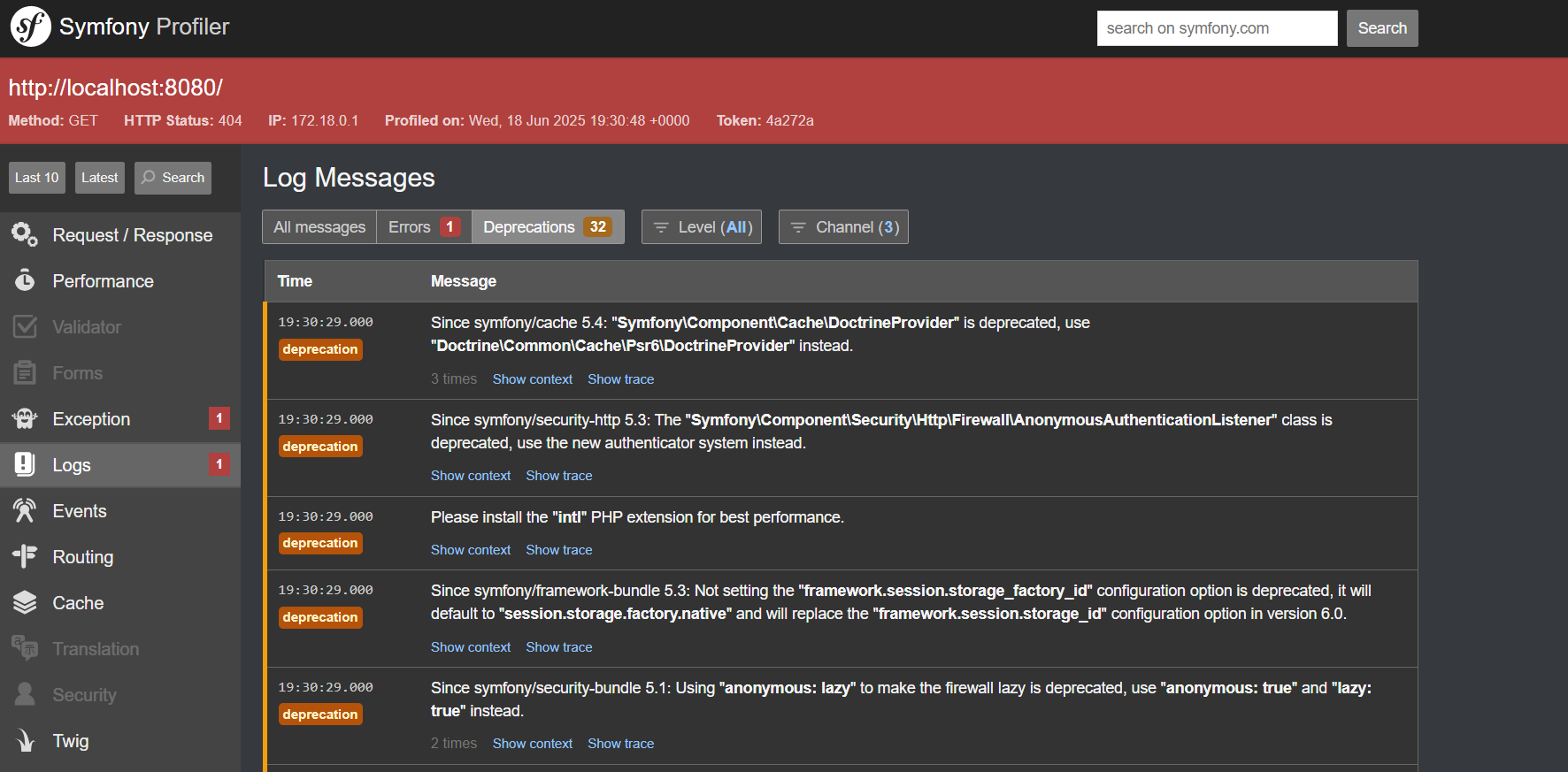


Figure 43:Aperçu des dépréciations dans Symfony profiler

Nous constatons que la plupart des dépréciations concernent la configuration, principalement parce que je n'ai pas beaucoup de code écrit. Dans les commits suivants, j'ai essayé de résoudre manuellement la dépréciation de la configuration :

Dans les commits suivants, j'ai essayé de résoudre manuellement la dépréciation de la configuration :

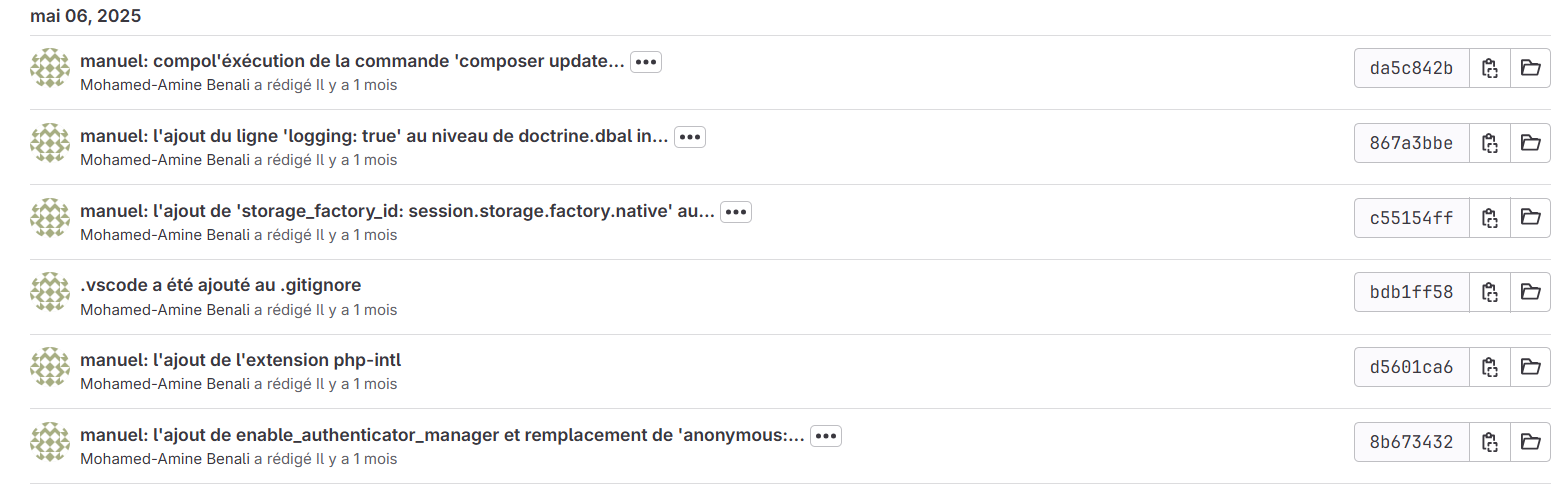


Figure 44: Liste des commits concernant la résolution manuelle des dépréciations de configuration

**Explication :**

* J'ai ajouté la ligne enable\_authenticator\_manager: true et remplacé « anonymous: true » → « lazy: true » dans config/packages/security.yaml :

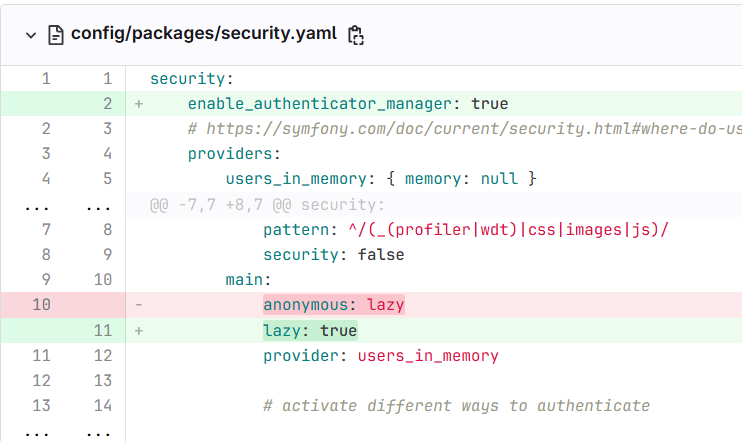


Figure 45: commit de "manuel: l'ajout de enable\_authenticator\_manager et remplacement de 'anonymous: lazy' to 'lazy: true' in config/packages/security.yaml"

* J'ai ajouté l'extension PHP-Intl à mon fichier Docker :



Figure 46: commit de "manuel: l'ajout de l'extension php-intl"

* J'ai ajouté la ligne 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 47: commit de "manuel: l'ajout de 'storage\_factory\_id: session.storage.factory.native' au fichier config/packages/framework.yaml"

* J'ai ajouté la ligne "logging: true" dans config/packages/doctrine.yaml :

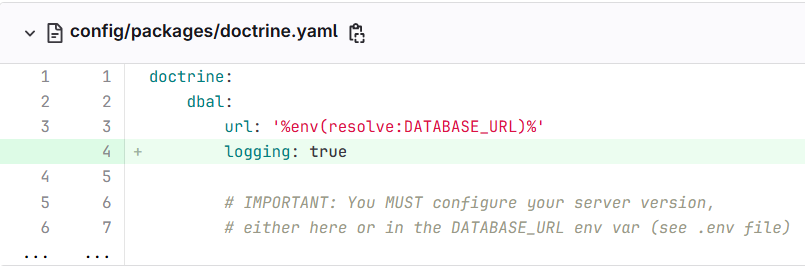


Figure 48: commit de "manuel: l'ajout du ligne 'logging: true' au niveau de doctrine.dbal in config/packages/doctrine.yaml"

* J'ai ensuite migré vers PHP 8, simplement en modifiant le fichier Docker :



Figure 49: commit de "Mise à jourée vers php 8"

**Recettes :**

Les recettes sont simplement des instructions exécutées par Symfony Flex. Au fil du temps, ces instructions sont mises à jour. Par conséquent, ces recettes doivent également être mises à jour :

J'ai donc commencé par « composer recipes:update ». Ce qui m'affiche la liste des recettes à mettre à jour :

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

Le contenu généré par l’IA peut être incorrect.

Figure 50: Liste des recettes qui nécessite une mise à jour

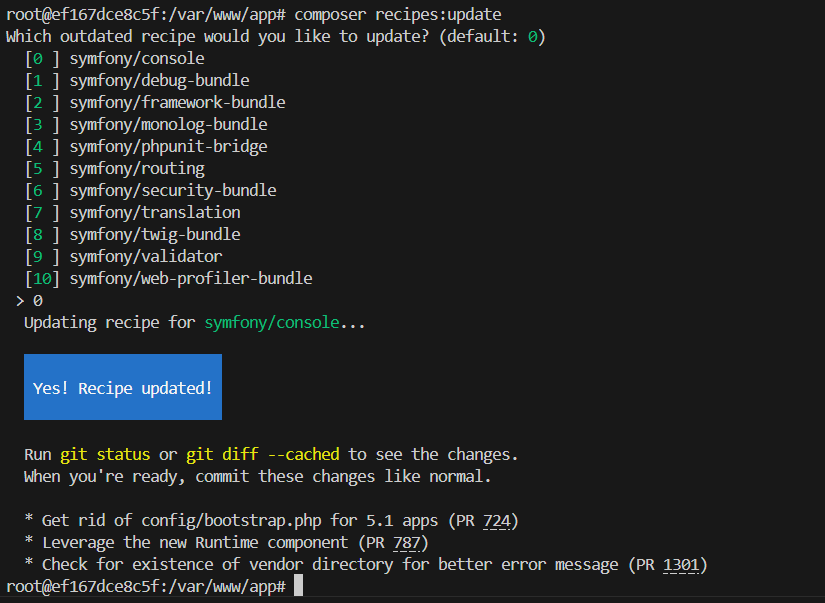


Figure 51: le choix de mettre à jour le symfony/console

Puis j'ai continué ainsi jusqu'à ce que j'aie terminé.



Figure 52: Liste de commit des mises à jour des recettes

Après le plantage de l'application et la lecture du message d'erreur, j'ai découvert qu'il fallait télécharger symfony/runtime.

composer require symfony/runtime

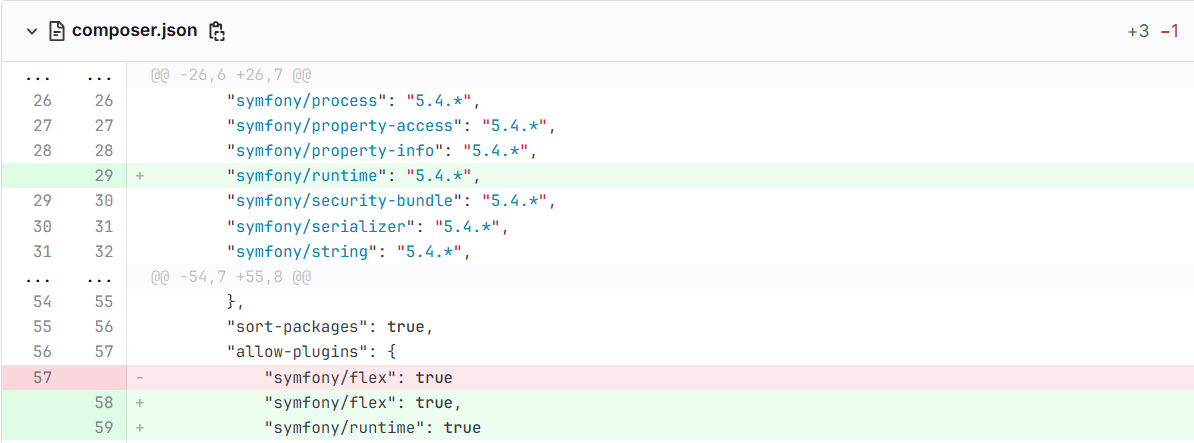
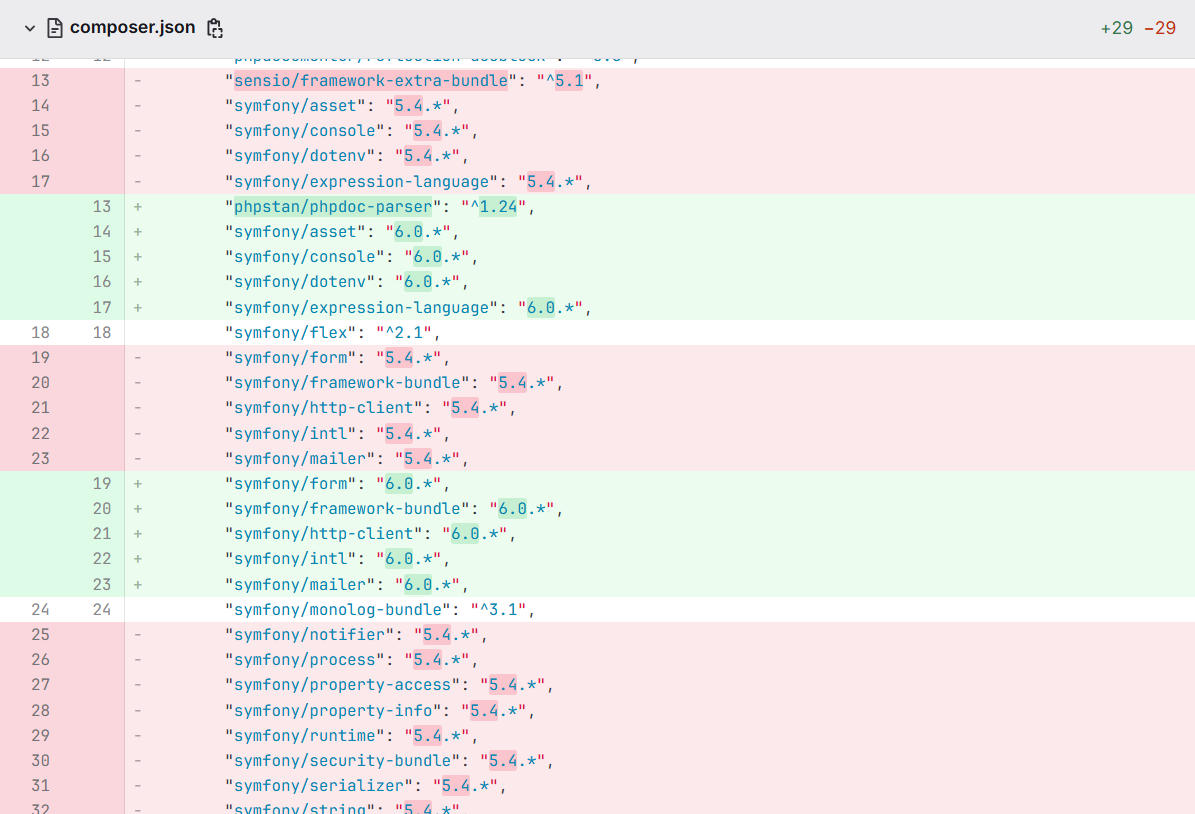


Figure 53: le details de commit de l'installation de symfony/runtime

Après cela, les versions obsolètes ont toutes disparu. Je savais donc qu'il était possible de passer à Symfony 6.4 en toute sécurité.

// SS Dépreciations sont disparu

Dans le fichier composer, j'assurerai de changer toutes les versions 5.4.\* en 6.4.\* pour les packages qui débutent par symfony/



Puis j’ai exécuté la commande suivante :

symfony composer up 'symfony/\*' ou symfony composer up

//SS DE COMMANDE

Et voilà, c'est fait.

//SS DE PAGE WEB

Résultat :

Suppression de toutes les dépréciations nécessaires et passage à Symfony 6.4

**2 ETAPE :**

Après avoir migré avec succès le nouveau projet, j'ai remarqué qu'avant, les dépréciations disparaissaient toutes soudainement après la mise à jour de toutes les recettes. J'ai donc décidé de tout recommencer avec une nouvelle branche, cette fois appelée « développement ». Au lieu de corriger manuellement les dépréciations, pourquoi ne pas simplement mettre à jour les recettes et les laisser faire tout le travail ?

Voici ce que j'ai fait :

// SS COMMITS

// SS

Résultat :

En ce qui concerne les dépréciations au niveau de la configuration, la plupart du temps, les recettes les corrigent immédiatement.

## Étapes de migration du projet Autossimo de Symfony 5.4 vers 6.4

### Suppression des dépréciations

La première étape essentielle dans la migration d’un projet de Symfony 5.4 vers 6.4 est de rendre le code "dépréciation libre" (Deprecation Free ). Cela signifie qu’il faut éliminer toutes les méthodes, classes ou interfaces marquées comme obsolètes avant de passer à la nouvelle version.

#### 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.

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*

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

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.

//////SS

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.

#### Liste des dépréciations identifiées dans Autossimo

La liste des dépréciations que j’ai identifiées dans Autossimo :

##### [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 :

$request = $this->requestStack->getCurrentRequest();

        if ($request === null || !$request->hasSession()) {

            return;

        }

        $session = $request->getSession();

(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. Afin de ne pas alourdir le corps du rapport, la liste complète des fichiers concernés est disponible dans l'annexe A.

##### [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. Afin de ne pas alourdir le corps du rapport, la liste complète des fichiers concernés est disponible dans l'annexe B.

##### [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 :

* src/Handler/LogoutHandler.php

##### [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 :**

Le seul fichier impacté est :

* src/Entity/User.php

##### [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 :**

Le seul fichier impacté est :

* src\Security\PunchOutAuthenticato.php

##### [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 :**

Le seul fichier impacté est :

* config/packages/security.yaml

##### [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 :

1. src\ApiAdCatalog\Process\UseCases\Model\CatalogApiUser.php
2. src\Entity\User.php

##### [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 :**

Les changements doivent être faites au niveau des fichiers suivants :

1. config/services.yaml
2. config/packages/framework.yaml
3. config/packages/snc\_redis.yaml

##### [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 :

* src\Repository\AdvertisingBannerRepository.php

##### [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 :**

Le changement doit être fait au niveau de :

* config/packages/framework.yaml

##### [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 :**

Le seul fichier qui doit être changer est :

* config/packages/framework.yaml

##### PHP Intl Extension :

L’installation du PHP Intl

* **Solution :**

RUN docker-php-ext-install intl

* **Impact sur le code :**

Le fichier qui doit être changer (dans la production doit être change par l’équipe d’exploitation) est :

* docker/php/Dockerfile

### 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

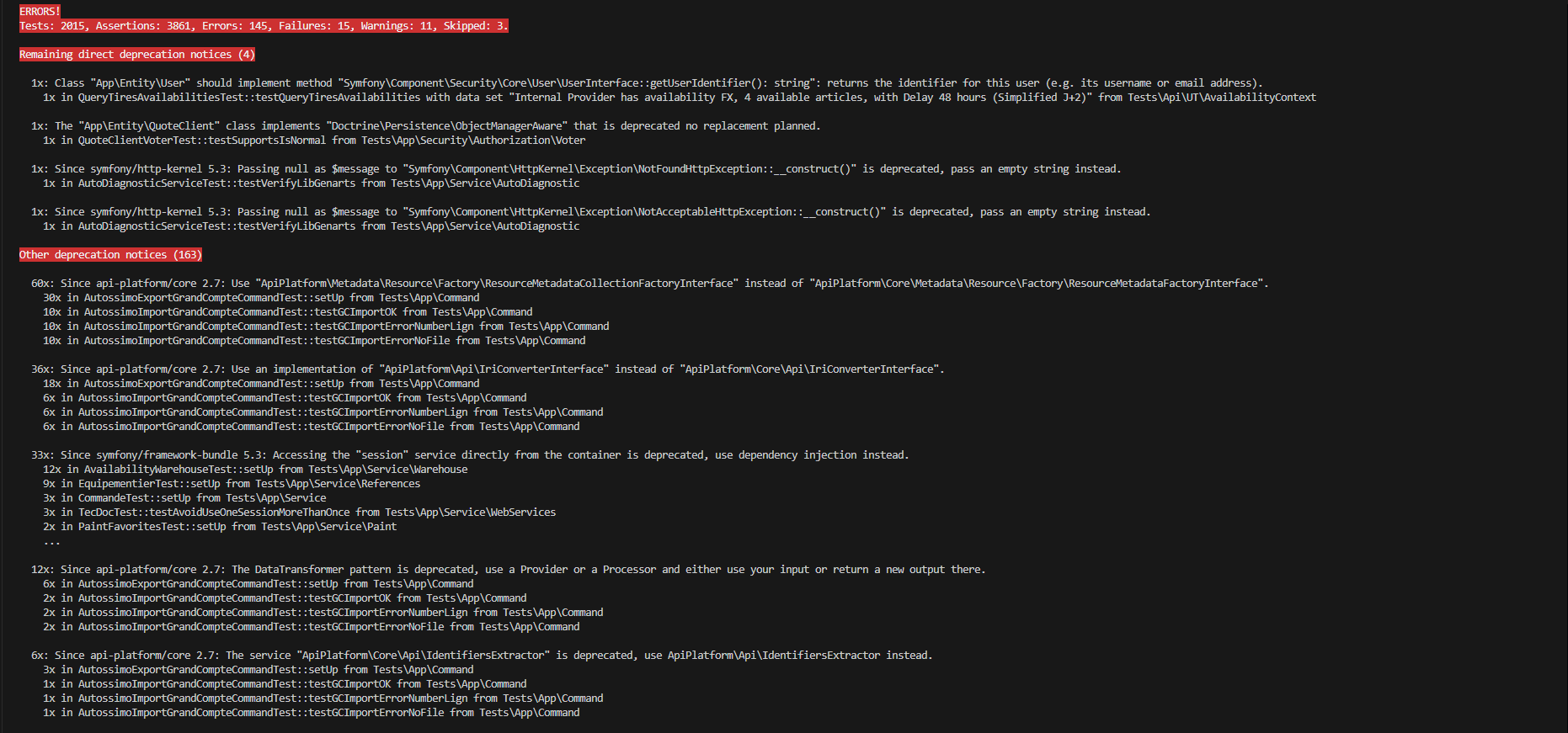


Figure 54: Dépréciations au niveau de PHPUnit

### Automatisation partielle avec Rector

Un outil puissant pour automatiser la correction de nombreuses dépréciations est Rector. J’ai tenté son utilisation, bien que je l’aie trouvé moins efficace dans certains cas spécifiques, car il peut générer des modifications inattendues ou incorrectes.

#### L’installation de Rector

Pour installer Rector, il suffit d’exécuter la commande suivante :

composer require --dev rector/rector

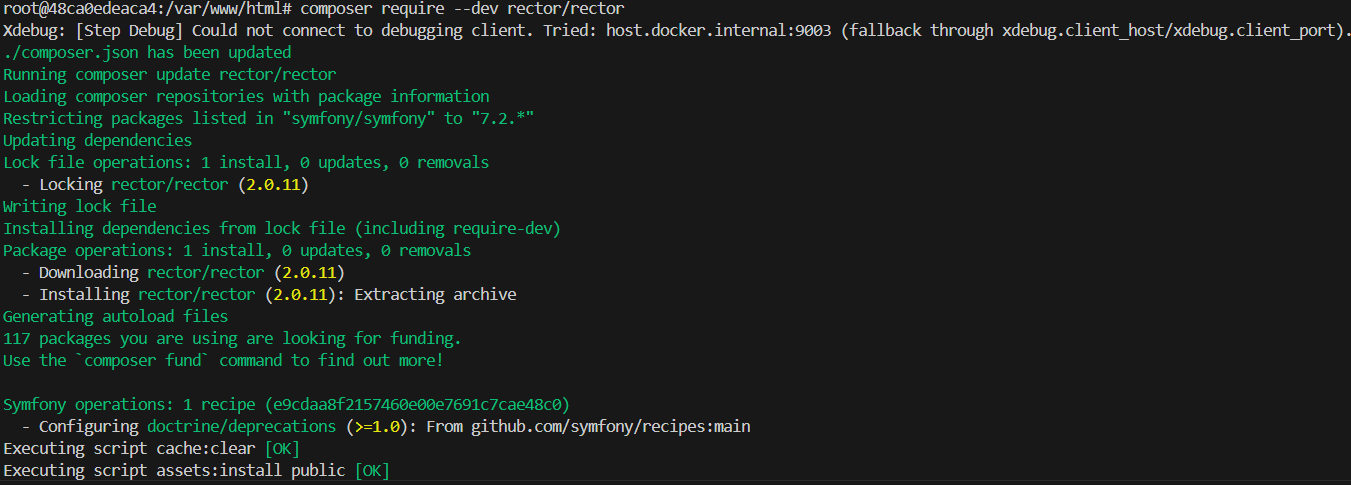


Figure 55: L'installation du Rector (2)

<?php

declare(strict\_types=1);

use Rector\Config\RectorConfig;

use Rector\Symfony\Set\SymfonySetList;

return RectorConfig::configure()

    ->withPaths([

        \_\_DIR\_\_ . '/ci',

        \_\_DIR\_\_ . '/config',

        \_\_DIR\_\_ . '/public',

        \_\_DIR\_\_ . '/src',

        \_\_DIR\_\_ . '/tests',

    ])

    ->withSymfonyContainerXml(\_\_DIR\_\_ . '/var/cache/dev/App\_KernelDevDebugContainer.xml')

    ->withSets([

        SymfonySetList::SYMFONY\_54,

        SymfonySetList::SYMFONY\_CODE\_QUALITY,

        SymfonySetList::SYMFONY\_CONSTRUCTOR\_INJECTION,

    ]);

#### Mode d’emploi

Deux commandes principales :

* ./vendor/bin/rector process --dry-run

**//SS Résultat de commande**

→ Simule les changements sans les appliquer

* ./vendor/bin/rector process

**//SS Résultat de commande**

Bien que Rector soit utile pour certaines corrections, j’ai préféré corriger manuellement certaines dépréciations afin d’avoir un contrôle total sur les modifications apportées au code.

### 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-noytsymfony/framework-bundle ^8.2...), consulté les journaux et la page officielle GitHub de chaque package, j'ai constaté ce qui suit :

// SS of the commands

##### [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. Afin de ne pas alourdir le corps du rapport, la liste complète des fichiers concernés est disponible dans l'annexe C.

##### [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 les fichiers suivants :

1. config/bundles.php
2. config/parameters\_erp.yaml
3. config/packages/swiftmailer.yaml
4. config/packages/test/swiftmailer.yaml
5. infra/.env

##### [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 les fichiers suivants :

1. config/bundles.php
2. config/parameters\_erp.yaml
3. config/packages/swiftmailer.yaml
4. config/packages/test/swiftmailer.yaml
5. infra/.env

##### [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.
* Remplacement : EnglishInflector de [Symfony/string](https://packagist.org/packages/symfony/string?type=application&page=12)
* **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. Afin de ne pas alourdir le corps du rapport, la liste complète des fichiers concernés est disponible dans l'annexe D.

##### [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 :**

*J'ai choisi la version minimale compatible. Ainsi, plus la version est ancienne, moins il y a d'opérations nécessaires pour que la nouvelle version fonctionne.*

### Passage à PHP 8.2

Symfony 6.4 nécessite au minimum PHP 8.1, cela nous a permis de bénéficier des améliorations de performance et des nouvelles fonctionnalités comme les attributs natifs PHP et les constructeurs promus. Cependant, pour des raisons techniques, nous avons décidé de passer à PHP 8.2.

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

Pour migrer vers PHP 8.2, nous devons vérifier les problèmes d'incompatibilité. À l'étape précédente, nous avons listé les packages incompatibles avec PHP 8.2, ainsi que la version à mettre à jour pour réussir la mise à niveau. Voici comment procéder :

Il suffit d'accéder à composer.json et de modifier les versions mentionnées pour obtenir la version compatible cible. Il est à noter que pour csa/guzzle-bundle, il convient de le remplacer comme indiqué. Cependant, si nous procédons étape par étape, nous pourrions le conserver et simplement le mettre à niveau pour prendre en charge PHP 8.2, qui est "dev-master" :

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",

#### Modification de l’environnement Docker

Après avoir mis à jour les packages nécessaires vers une version compatible, la mise à jour vers PHP 8.2 peut être effectuée immédiatement.

On peut modifier l’image utilisée dans le Dockerfile pour utiliser PHP 8.2 :

ARG PHP\_VERSION=8.2

### PHP

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

Puis reconstruit l’environnement :

docker compose up php -d –build

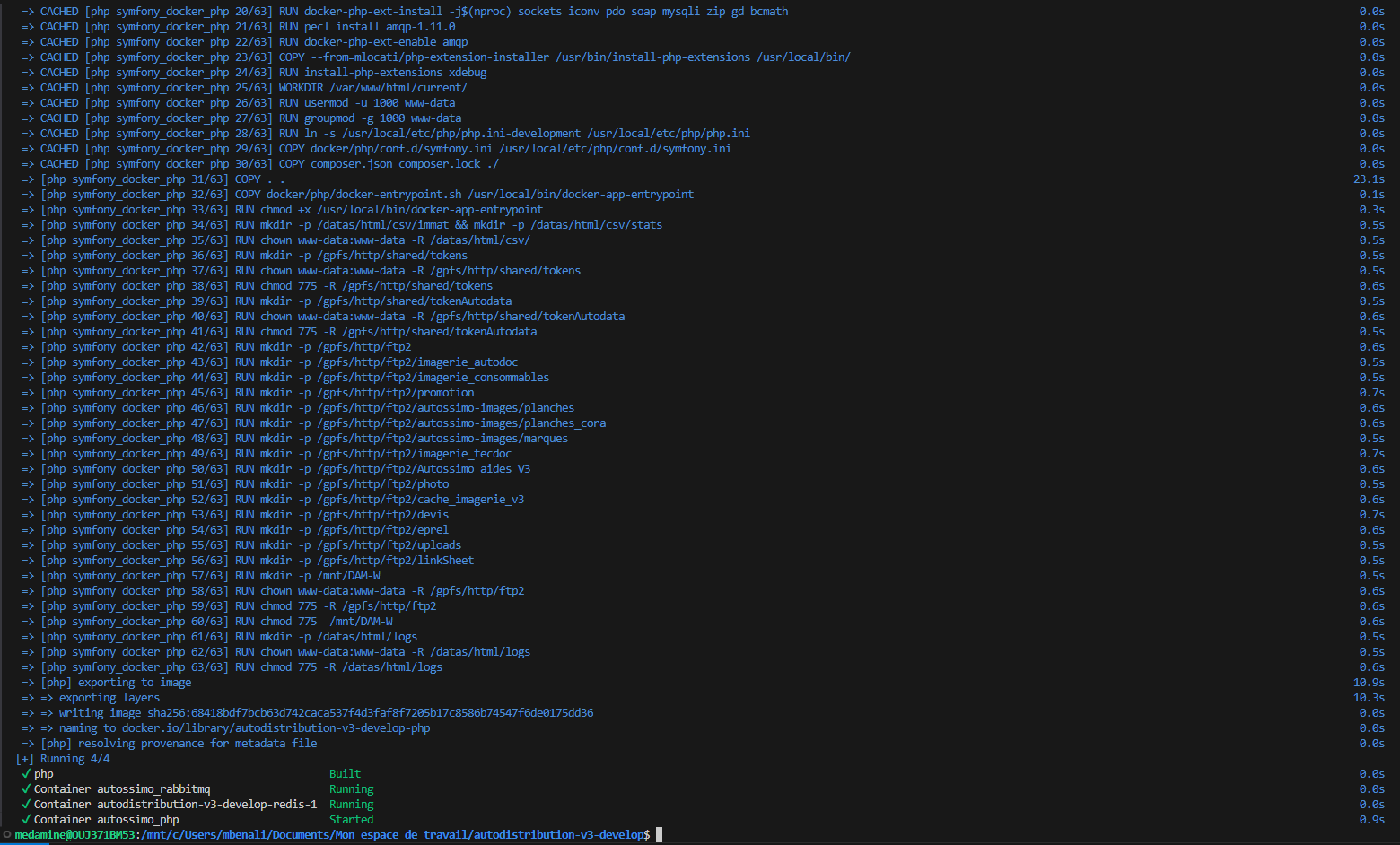


Figure 56: L'exécution de la commande "docker compose up php -d --build"

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

Modification dans composer.json :

  "require": {

    "php": "^8.2",

    "ext-json": "\*",

    "ext-zlib": "\*",

Ensuite, exécution de la commande suivante pour entrer dans le conteneur :

make enter

Puis l’exécution de la commande :

composer update

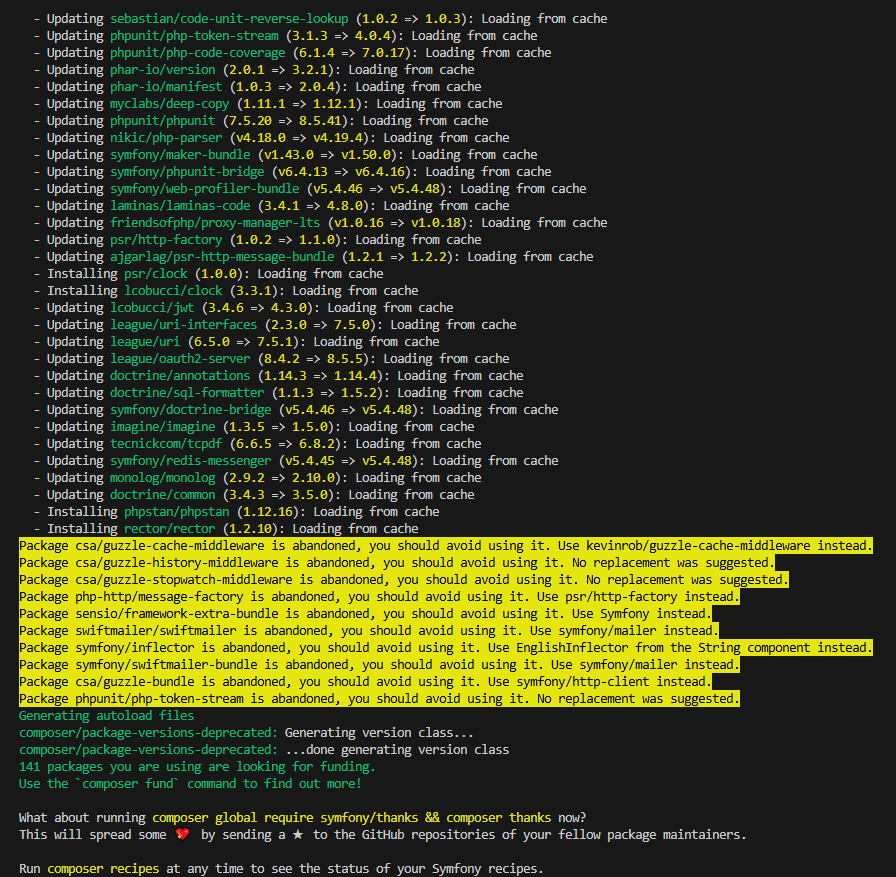


Figure 57: L'exécution de la commande "composer update"

La mise à jour devra être faite avec succès ✓

### Utilisations des nouvelles fonctionnalités de PHP 8

Avec le passage à PHP 8.2, plusieurs fonctionnalités peuvent être exploitées pour moderniser le code :

* Attributs natifs au lieu des annotations (Doctrine / Sensio)
* Constructeur promu pour simplifier l’injection de dépendances
* Meilleure gestion des types stricts
* Meilleures performances

Par exemple, voici comment convertir une annotation Doctrine en attribut :

<?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 { }

* **Nettoyage complet des annotations**

On peut utiliser cette configuration dans rector.php (le fichier de configuration de [Rector](#_Rector)), pour automatisuer certaines migrations :

<?php

use Rector\Doctrine\Set\DoctrineSetList;

use Rector\Set\ValueObject\LevelSetList;

$rectorConfig->withSets([

    LevelSetList::UP\_TO\_PHP\_82,

    DoctrineSetList::ANNOTATIONS\_TO\_ATTRIBUTES,

]);

Puis lancé la commande : ./vendor/bin/rector process

### Mise à jour de Symfony 5.4 vers 6.4

Une fois toutes les dépréciations corrigées et les dépendances mises à jour, il était temps de passer à Symfony 6.4.

#### 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 dépenadances

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

**// SS OF composer update erreur**

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 paquets

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) » 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

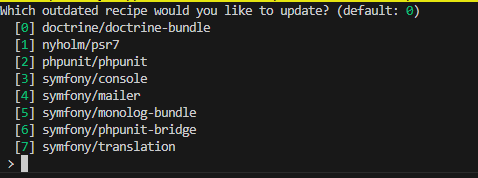


Figure 58: Sortie de la commande "compose recipes:update"

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.

// SS RECETTE CONFLIT



### Mettez 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.

### Les outils utilisés

|  |  |
| --- | --- |
| Outil | Rôle |
| [Composer](#_Composer) | Gestion des dépendances |
| [Rector](#_Rector) | Refactorisation automatique |
| Symfony CLI / Flex | Aide à la mise à jour |
| [PHPUnit](#_PHPUnit) | Validation des tests unitaires |
| [Git](#_Git) | Versioning des modifications |

### Etudes préliminaires réalisées

Afin de valider la faisabilité de la migration, j’ai effectué les actions suivantes :

#### Migration réussie d’un projet test

J’ai migré un projet simple contenant quelques contrôleurs, formulaires et services. Cela m’a permis de :

* Valider les étapes clés de la migration
* Identifier les points bloquants classiques
* Tester les outils comme [Rector](#_Rector)

#### Réutilisation du composer.json du projet réel

J’ai remplacé le composer.json du projet test par celui du projet Autossimo puis j’ai résolu les dépréciations en mettant à jour les recettes, remplaçant les bundles non-compatible et j’ai lancé la commande :

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

* **Résultat** : Le projet est passé à PHP 8.2 et Symfony 6.4 sans erreur critique.

### Conclusion

La migration de Symfony 5.4 à 6.4 nécessite une restructuration profonde du projet, notamment au niveau de la sécurité, des services, et des dépendances externes. Bien que cette analyse montre qu’une telle migration est réalisable, elle requiert une planification rigoureuse pour gérer les incompatibilités et moderniser le code source.

# CHAPITRE 3 : TECHNOLOGIES ET OUTILS UTILISES ET RENCONTRES



## Introduction

Ce chapitre présente les différents Langages, Frameworks, Technologies et outils utilisées ou rencontrés au cours de ma mission.

## Conclusion

Ce chapitre décrit l'ensemble des outils que j'ai rencontrés ou utilisés au cours de ma mission. Il existe de nombreux autres packages et bibliothèques dont nous n'avons pas parlé et qu'Autossimo utilise.

# CHAPITRE 4 : REALISATION DU PROJET

## Introduction

Introduction

Type de migration 🡺Synthèse sensiolab

Le choix type 3

Roadmap

Chantier pour les Inteface dépricié 🡺 SessionInterface , container, ……

Création d’un spike pour analyser des dépriciation existant et des ticket en sortie pour les différents Dépriciation ainsi leurs estimation ….

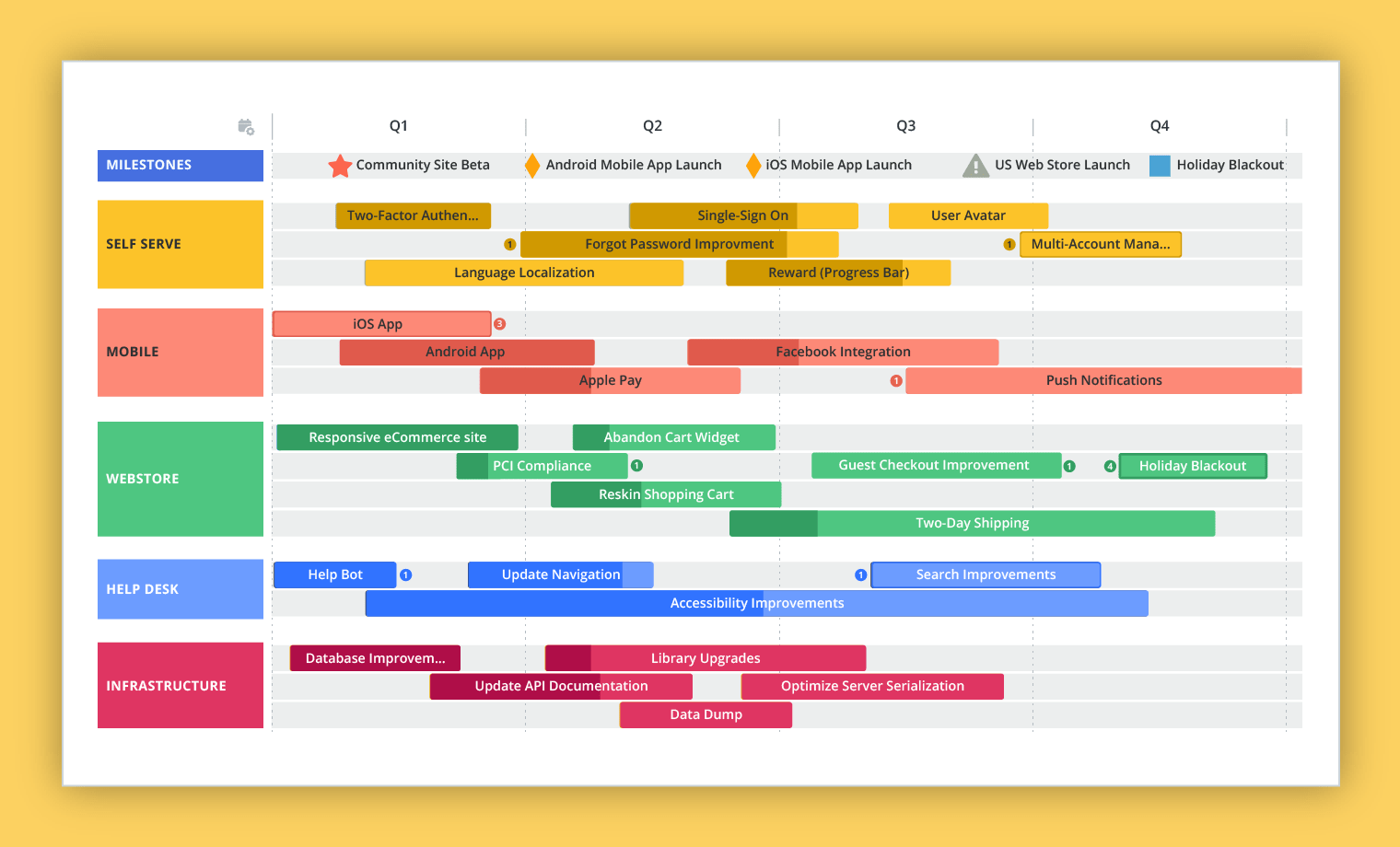
Chantier pour chaque package obsolère : Guzzle, Trikoder ….

Création d’un spike pour analyser des dépriciation existant et des ticket en sortie pour les différents Dépriciation ainsi leurs estimation ….

Enfin appliqué 3eme analyse (après garantir la résolution des problème potentiel)

Résoudre toutes problème après la migration

TNR manuelle de l’application avant déploiement



Une image contenant capture d’écran, texte, conception

Le contenu généré par l’IA peut être incorrect.

Cette section décrit les étapes concrètes entreprises pour mettre en œuvre la migration du projet Autossimo de Symfony 5.4 vers 6.4. Malgré des obstacles techniques liés aux dépendances obsolètes, une partie de la migration a pu être réalisée avec succès, notamment le passage à PHP 8.2.

### Migration réussie d’un projet test

Afin de mieux comprendre les étapes nécessaires à la migration, j’ai commencé par travailler sur un projet simple, contenant quelques contrôleurs, formulaires et services basiques.

Les actions suivantes ont été effectuées :

* Mise à jour du fichier composer.json pour passer à Symfony 6.4
* Passage à PHP 8.2
* Correction des dépréciations grâce à Rector et le travail manuel
* Mise à jour du composant Security (security.yaml)
* Tests unitaires passés avec succès

### Réutilisation du composer.json du projet réel

Pour simuler une migration réaliste, j’ai remplacé le fichier composer.json du projet test par celui du projet réel (Autossimo) .

Avant d’exécuter la mise à jour, j’ai effectué plusieurs modifications essentielles :

* Changement de "php": "^7.3" en "php": "^8.2"
* Mise à jour ou remplacement des dépendances incompatibles avec Symfony 6.4
* Ajout ou modification des configurations nécessaires pour prendre en charge les nouvelles structures requises par Symfony 6.4

Toutes ces opérations ont été réalisées au sein d’un environnement Docker, ce qui m’a permis de tester la migration dans des conditions proches de celles de production sans impacter le projet réel.

* **Résultat :**
* Le projet est passé à PHP 8.2
* Il est également passé à Symfony 6.4 sans erreur critique bloquante

Cela montre que, dans un environnement isolé bien préparé, la migration est possible même pour un projet complexe.

### Tentative de migration du projet réel

Malgré le succès partiel dans l’étape précédente, la tentative de migration directe du projet réel n’a pas abouti entièrement.

* **Étapes tentées :**
* Modification du composer.json du projet réel pour passer à Symfony 6.4
* Exécution de composer update symfony/\* --with-all-dependencies
* Correction automatique via Rector
* Vérification des fichiers critiques comme security.yaml, services.yaml, etc.
* **Résultat :**
* Seule la mise à jour vers PHP 8.2 a réussi
* Mais plusieurs erreurs ont empêché le passage complet à Symfony 6.4 :
  + Incompatibilité de certains bundles tiers (csa/guzzle-bundle, trikoder/oauth2-bundle)
  + Problèmes liés au nouveau format de security.yaml
  + Dépendances obsolètes ou non maintenues
  + Erreurs dans les templates Twig et services mal configurés

### Obstacles rencontrés

|  |  |
| --- | --- |
| Obstacle | Description |
| csa/guzzle-bundle | Abandonné, incompatible avec Symfony 6 |
| symfony/swiftmailer-bundle | Abandonné, remplacé par symfony/mailer |
| trikoder/oauth2-bundle | Ne supporte pas Symfony > 5.4 |
| sensio/framework-extra-bundle | Abandonné, doit être supprimé ou remplacé |
| symfony/inflector | Abandonné, remplacé par EnglishInflector du composant symfony/string |
| Anciens services | Certains services utilisent des interfaces dépréciées (SessionInterface, UserPasswordEncoderInterface) |

### Solutions envisagées

Devant ces obstacles, voici les actions qui peuvent permettre de finaliser la migration :

1. Remplacer les bundles abandonnés :

* Remplacer csa/guzzle-bundle par symfony/http-client
* Remplacer trikoder/oauth2-bundle par league/oauth2-server-bundle
* Supprimer et remplacer sensio/framework-extra-bundle

1. Corriger les interfaces dépréciées :

* Remplacer SessionInterface par RequestStack
* Passer de UserPasswordEncoderInterface à UserPasswordHasherInterface
* Modifier security.yaml pour utiliser password\_hashers au lieu de encoders

1. Utiliser Rector pour :

* Corriger les dépréciations grâce aux sets SymfonySetList::SYMFONY\_54 et LevelSetList::UP\_TO\_PHP\_82

1. Mettre à jour les tests :

* Adapter les tests unitaires et fonctionnels pour refléter les nouvelles structures

### Conclusion de la réalisation

Bien que la migration complète du projet réel n’ait pas encore abouti, cette tentative a permis d’identifier les principaux blocages techniques. J’ai réussi à passer le projet à PHP 8.2, ce qui constitue une étape essentielle avant la mise à jour complète vers Symfony 6.4.

La réussite finale dépendra de la correction manuelle de certains modules critiques et du remplacement des dépendances obsolètes. Cela nécessitera un temps supplémentaire et une approche méthodique pour garantir la stabilité de l’application après migration.