Rapport d’Activité Professionnelle Mastère 1

Mon poste d'intégrateur et développeur chez Orange Business

Rapport d’alternance présentant le cadre professionnel, les missions réalisées, les compétences mobilisées et le bilan personnel sur l’année de Mastère 1 (2025–2026) au sein d’Orange Business.

Alexandre Noblet
3 juillet 2026

Table des matières

Table des illustrations

1 Introduction

Depuis octobre 2024 et jusqu'en octobre 2027, j'occupe le poste d'**intégrateur et développeur mobile** en alternance. J'évolue au sein de la Practice Smart Tracking, une entité spécialisée dans l'Internet des Objets (IoT) dirigée par Ghislaine LEHMANN. Mes interventions s'inscrivent également dans un cadre plus large lié aux activités industrielles pilotées par Cécile DELERY.

Mon quotidien se définit par une double compétence. D'un côté, j'agis en tant qu'intégrateur technique dans le monde industriel. De l'autre, je réalise du développement applicatif directement lié aux projets de suivi connecté. Cette configuration me pousse à gérer à la fois l'intégration des solutions au sein des systèmes existants (interopérabilité, déploiement, cohérence de l'écosystème) et la création des fonctionnalités applicatives nécessaires pour exploiter les données récoltées.

Ce rapport a pour objectif d'analyser mon parcours et l'évolution de mes compétences sur cette période. Il mettra en lumière comment cette double expertise est devenue un atout pour répondre aux enjeux de transformation numérique. J'aborderai particulièrement les problématiques d'alignement entre les contraintes du terrain et les exigences techniques, ainsi que mes axes d'amélioration sur la qualité logicielle, la maintenabilité et la capacité d'adaptation des solutions.

À travers l'analyse de ces trois missions, ce rapport ne se limite pas à une simple description de tâches. Il s'articulera autour d'une problématique centrale, véritable fil rouge de mon année : **Dans quelle mesure un développeur-intégrateur peut-il garantir la maintenabilité logicielle et la résilience des architectures face à l'urgence des déploiements en environnement industriel critique ?** Il s'agira d'explorer ces axes d'analyse pour démontrer comment la rigueur méthodologique, l'adoption d'une posture d'architecte et l'intégration de workflows IA permet de sécuriser la production technique tout en répondant aux hautes exigences opérationnelles de nos clients. Ce rapport est le fruit d’une démarche réflexive sur une année d’apprentissage continu. Il ne présente pas l’aboutissement d’une transition achevée, mais témoigne d'un cheminement itératif, marqué par la confrontation permanente entre la théorie architecturale et les réalités opérationnelles du terrain.

2 Présentation du cadre professionnel

Mon environnement de travail s'articule autour de deux pôles d'activités distincts mais complémentaires : le domaine du Smart Tracking (IoT) d'une part, et les activités de convergence industrielle d'autre part.

Sur le périmètre du **Smart Tracking**, l'objectif opérationnel est de rendre la donnée de suivi exploitable. Il s'agit de développer des solutions permettant à nos clients d'améliorer la traçabilité de leurs actifs, de fiabiliser le pilotage de leurs flux logistiques et d'optimiser leurs interventions sur le terrain.

Parallèlement, mon intervention sur les **activités industrielles** s'inscrit au cœur d'une mutation technologique majeure : l'Industrie 4.0. Aujourd'hui, le principal défi des usines n'est plus seulement de produire, mais de comprendre *comment* elles produisent en temps réel. Mon métier d'intégrateur sur ce pôle consiste à réaliser la convergence entre l'informatique (IT) et l'exploitation (OT), deux mondes historiquement cloisonnés. Il s'agit de remonter la donnée du terrain (automates, capteurs, puces RFID) vers des systèmes d'information globaux (Cloud, ERP, MES). Cette convergence soulève des défis critiques que je dois gérer au quotidien : garantir la cybersécurité des infrastructures désormais connectées, et assurer une haute disponibilité des systèmes, car un arrêt informatique implique aujourd'hui un arrêt direct de la chaîne de production physique.

Cette double casquette me conduit à collaborer avec des acteurs très variés au quotidien. Je dialogue d'un côté avec les experts IoT pour définir l'architecture des applications de traçabilité, et j'échange de l'autre avec les techniciens et responsables métiers de l'industrie pour valider les besoins d'intégration et les retours d'usage.

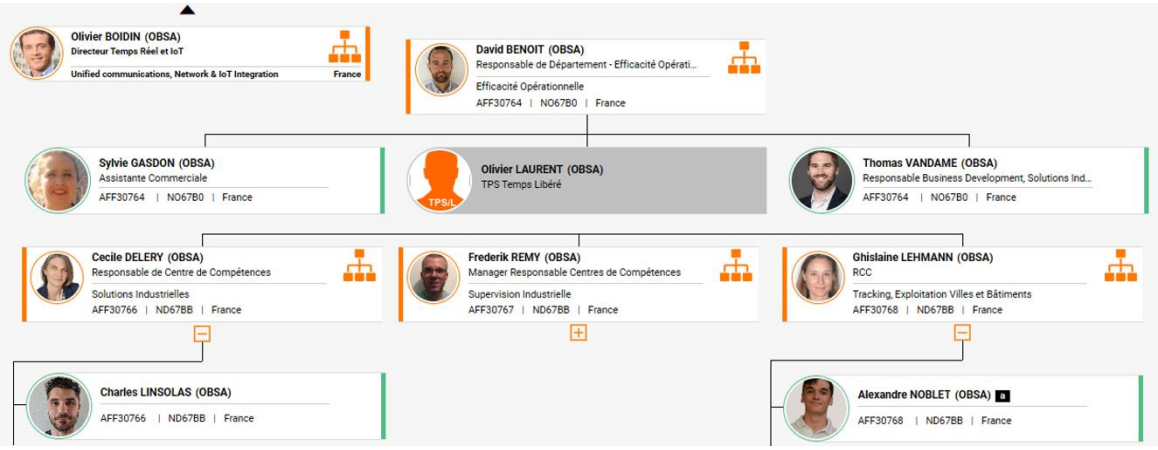


Figure 1. Diagramme hiérarchique du pôle Smart Tracking IoT et Solutions Industrielles

3 Missions réalisées et compétences mobilisées

3.1 Mission 1 : Intégration industrielle de PlanetTogether

3.1.1 Contexte stratégique : l'unification des systèmes de production

Avant d'aborder les aspects techniques de l'intégration, il est essentiel de comprendre l'enjeu métier pour ce client industriel. Historiquement, les différentes usines du groupe fonctionnaient en silos, s'appuyant sur des systèmes d'information hétérogènes allant d'ERP locaux fragmentés à des fichiers Excel complexes, avec des processus manuels isolés. Cette disparité empêchait toute vision globale et rendait impossible la consolidation fiable des données de production à l'échelle du groupe.

Le choix d'intégrer PlanetTogether s'inscrit donc dans un programme de transformation et de standardisation ambitieux. L'objectif stratégique du client est d'imposer un outil d'ordonnancement unifié (APS) pour l'ensemble de ses sites à travers le monde. Il s'agit de remplacer ces pratiques locales disparates par une plateforme unique et centralisée, capable de briser les silos pour alimenter le système global de l'entreprise avec des données standardisées, comparables et exploitables d'une usine à l'autre.

3.1.2 Un contexte de déploiement international et la maîtrise de l'anglais technique

La dimension internationale de ce projet a ajouté une couche de complexité organisationnelle et linguistique significative. Le client industriel déploie cette solution à l'échelle mondiale, avec des sites pilotes basés en Allemagne et en Italie, suivis de futures intégrations en Pologne et en Suisse. De plus, la gouvernance informatique de ce client (le département *Global IT*) est entièrement centralisée et basée en Inde.

Par conséquent, l'intégralité de mes échanges de coordination, qu'il s'agisse des réunions de suivi hebdomadaires, des ateliers de conception architecturale ou des sessions de débogage technique, s'est déroulée exclusivement en anglais. Ce contexte multiculturel m'a contraint à développer une maîtrise rigoureuse de l'anglais technique. Vulgariser un modèle de données complexe, argumenter des choix d'architecture logicielle ou rédiger la documentation des scripts d'extraction dans une langue étrangère est devenu une exigence quotidienne. Cette immersion m'a permis de franchir un cap dans ma communication professionnelle, validant ma capacité à piloter techniquement un projet de bout en bout dans un environnement résolument tourné vers l'international.

3.1.3 Découverte de l'outil PlanetTogether

La première étape de cette mission a été de prendre en main le logiciel PlanetTogether, une solution de planification et d'ordonnancement de production (APS) dont le rôle est d'optimiser les flux des usines en temps réel pour accroître leur efficacité opérationnelle.

Cette montée en compétence n'était pas une simple formalité d'accueil : un intégrateur qui ne comprend pas la logique interne de l'outil cible produit des flux de données syntaxiquement corrects mais fonctionnellement faux. Je ne pouvais pas modéliser fidèlement les contraintes de production du client sans comprendre d'abord comment PlanetTogether les interprète. Cette immersion a ainsi orienté mes choix d'intégration sur trois plans :

- **Logique métier complexe :** Comprendre comment traduire des contraintes de production en algorithmes de planification. C'est ce vocabulaire de domaine, assimilé dès cette phase, qui a ensuite servi de socle aux modèles de la refonte architecturale (section 3.1.6).
- **Intégration de données :** Gérer les flux d'informations entre les systèmes de production et l'outil d'ordonnancement. Comprendre comment l'outil consomme la donnée m'a permis de concevoir des flux respectant ses invariants, plutôt que de découvrir les incompatibilités en production.
- **Enjeux opérationnels :** Appréhender l'impact direct de nos solutions techniques sur la performance des usines clientes. Mesurer qu'une erreur d'ordonnancement peut arrêter une ligne de production a fixé, dès le départ, le niveau d'exigence de fiabilité qui a ensuite guidé ma stratégie de tests.

3.1.4 Architecture et flux de données

Le logiciel a été développé en C# et se base sur une architecture en base de données SQL Server pour l'import et l'export des données, comme illustré sur le schéma ci-dessous.

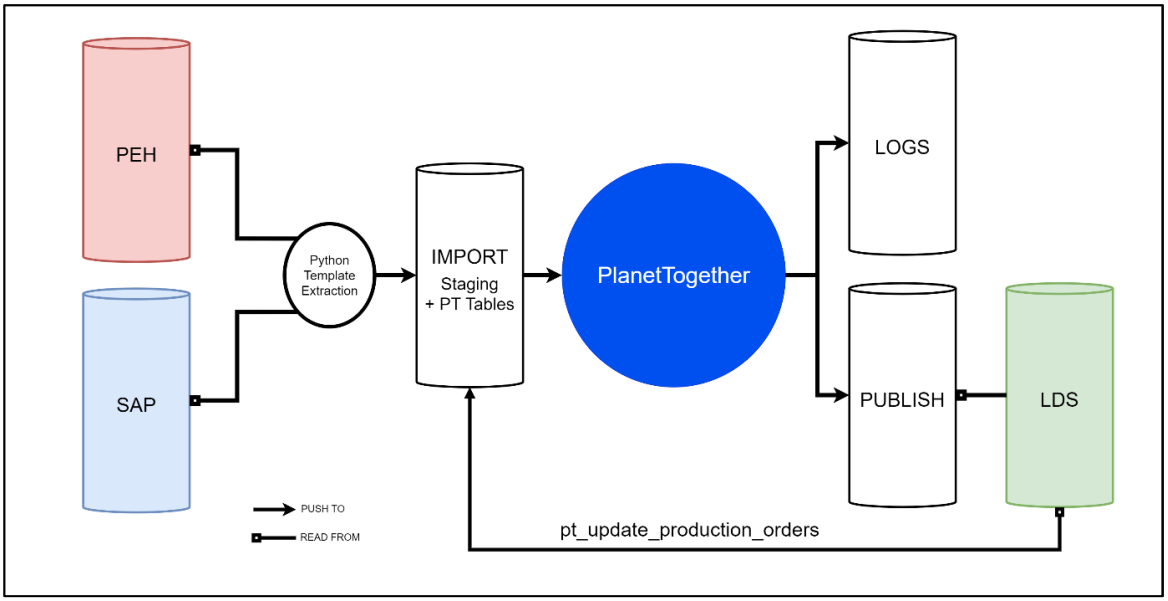


Figure 2. Architecture de l'intégration de PlanetTogether

L'écosystème autour de PlanetTogether s'articule autour de trois logiciels majeurs côté client :

- **PEH :** Permet de lister les machines, les maintenances planifiées et les horaires d'ouverture.
- **SAP :** Gère la liste des moules, les ordres de productions, les gammes et les approvisionnements en matières premières.
- **LDS :** Le logiciel de suivi de production (MES) du client.

Dans ce schéma, nous retrouvons un bloc central nommé «PythonTemplateExtraction». Il représente le cœur de ma mission d'intégration. Ce script a pour objectif de récolter les données des différentes sources, de les agréger et de les « traduire » dans un langage compréhensible par PlanetTogether afin de modéliser les contraintes émises par la production. Ce bloc gravite autour de l'agrégation et du traitement de multiples flux de données hétérogènes. Comme l'illustre le schéma de l'architecture ci-dessous, le module central *Python Template Extraction* agit comme un véritable carrefour d'intégration. Il collecte les informations préalablement extraites de systèmes sources variés (SAP, LDS, PEH, fichiers Excel de Master Data) via des connexions ODBC et formatées par une première couche de procédures stockées.

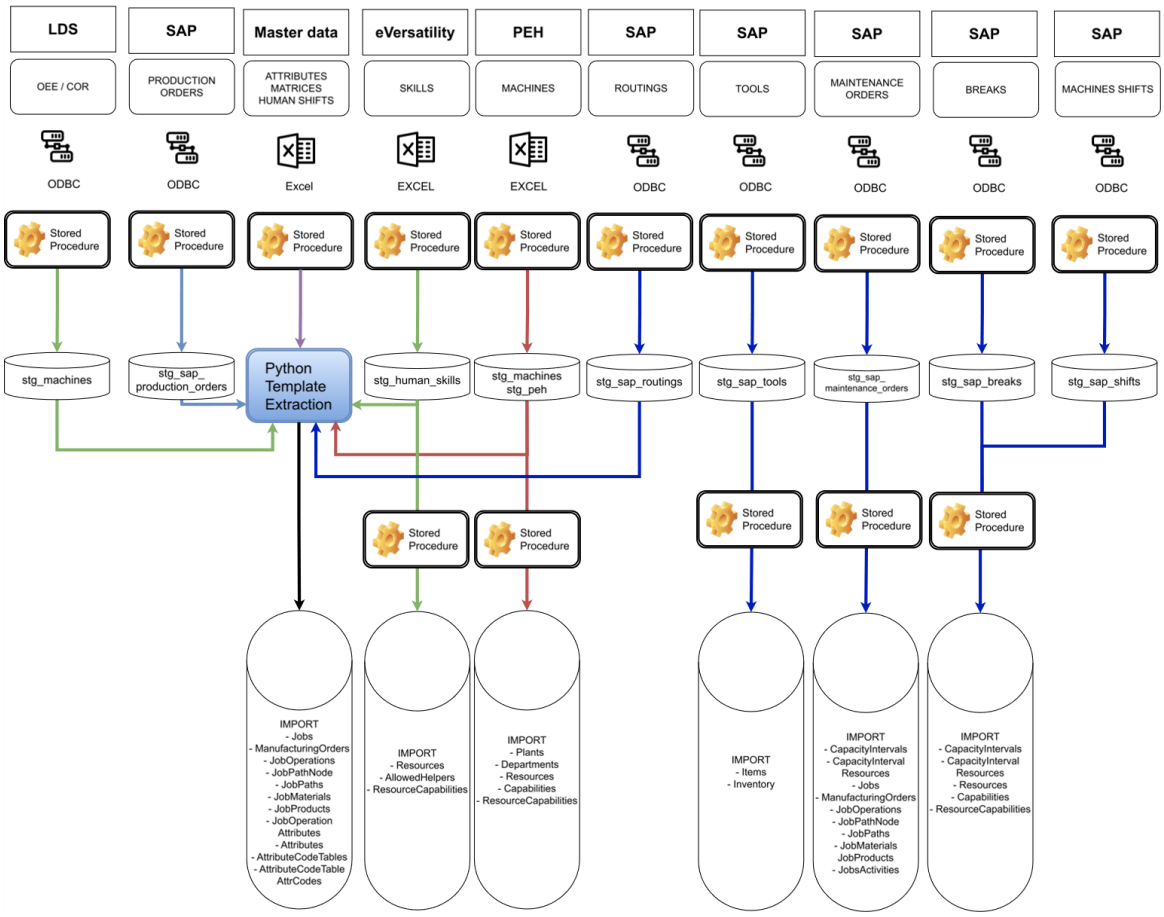


Figure 3. Flux des données autour du script PythonTemplateExtraction

Concrètement, le script Python vient puiser dans plusieurs tables de transition dites de *staging* (notamment stg_machines, stg_sap_production_orders, stg_human_skills, stg_peh et stg_sap_routings). Sa mission est de croiser et de "traduire" ces informations disparates pour générer une structure de données unifiée. Celle-ci vient ensuite alimenter directement la base de données cible finale d'**IMPORT**, peuplant ainsi les tables maîtresses nécessaires à la planification (telles que *Jobs*, *ManufacturingOrders*, *JobOperations*, *JobPaths*, etc.).

Ce choix d'architecture n'allait pas de soi. Un ETL du marché aurait pu remplir ce rôle, mais au prix d'une dépendance de licence et d'une boîte noire difficile à déboguer dans un contexte où chaque usine apporte ses spécificités. Le module Python sur mesure, adossé à des tables de staging SQL, offrait trois garanties : une transparence totale sur les transformations appliquées (indispensable pour diagnostiquer les écarts de données avec le Global IT), une testabilité fine de chaque règle de traduction et une évolutivité maîtrisée à mesure que de nouveaux sites rejoignent le programme. La couche de staging, elle, découple les systèmes sources du format cible : une évolution de SAP ou du MES n'impacte que la procédure d'extraction concernée, jamais le cœur du script.

3.1.5 L'enjeu de la maintenabilité face à la dette technique

Cette mission s'inscrit dans un programme de standardisation ambitieux. Je travaille sur un dépôt de code partagé (monorepo) avec deux autres développeurs. Notre objectif est de sécuriser ce pipeline critique ETL (Extraction, Transformation, Chargement) pour absorber l'arrivée de nouveaux sites industriels (Pologne, France, Suisse), après avoir d'abord signé pour des pilotes en Allemagne et en Italie.

Dans le domaine de l’intégration, la réutilisation de code existant est rare. Il a donc fallu livrer rapidement une première version du code pour les deux sites pilotes, en respectant des délais très serrés. Au fur et à mesure que le client affinait son besoin, de nouvelles fonctionnalités se sont greffées sur ce code initial. Résultat : le script s'est transformé en un fichier monolithique atteignant 1543 lignes.

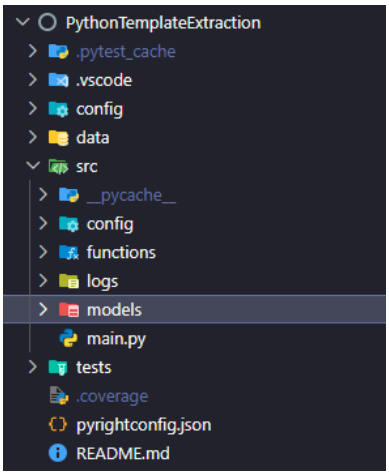
Plutôt que de me limiter à la création du code initial, j'ai dû évaluer la dette technique accumulée. Ce script était devenu une **dette à intérêt composé** : chaque ajout de fonctionnalité risquait de fragiliser l'existant. Le choix de la refonte n'était pas une préférence technique, mais un investissement stratégique. Il était indispensable pour garantir la scalabilité du système vers les sites polonais et suisses, évitant ainsi un 'effet tunnel' où la maintenance aurait fini par occulter toute capacité d'innovation.

3.1.6 Refonte architecturale et assurance qualité

J'ai mené un audit qualité complet pour réduire cette dette technique. Pour gagner en efficacité, je me suis appuyé sur l'intelligence artificielle afin d'analyser le code existant, ce qui m'a permis d'identifier et de prioriser une vingtaine de problèmes critiques liés à la sécurité et à la maintenabilité, au premier rang desquels l'absence de typage contrôlé (les 323 erreurs Pyright détaillées en 3.1.8) et un couplage monolithique rendant toute couverture de tests impossible. Il fallait également sécuriser de toute urgence une règle métier de synchronisation. Concrètement, l'ancien fonctionnement écrasait silencieusement les données lors de la mise à jour : si un planificateur modifiait manuellement un ordre dans PlanetTogether, le script supprimait cette modification lors de la synchronisation suivante.

Pour résoudre cela, j'ai découpé le fichier monolithique en modules à responsabilité unique, en m'appuyant sur des modèles de domaine homogènes issus du vocabulaire propre à PlanetTogether. La nouvelle architecture se structure ainsi :

- /config : paramètres propres à chaque usine.
- /data : données issues de chaque usine.
- /src : code source central.
  - /models : modèles de domaines issus de PT.
  - /config : paramétrage interne du script.
  - /functions : fonctions utilitaires.
- /tests : suite complète de tests automatisés.



Grâce à cette architecture découplée, l'ajout de fonctionnalités est devenu plus simple et ciblé : une évolution ne touche désormais qu'un module et ses tests associés, là où le monolithe imposait de relire l'ensemble du script pour en évaluer les effets de bord.

3.1.7 Impact métier, ROI et conduite du changement

La réussite de l'intégration de PlanetTogether ne se mesure pas uniquement à la qualité du code C# ou Python produit, mais à son adoption sur le terrain. Le passage d'outils locaux hétérogènes et de processus manuels à un système de planification automatisé et centralisé a constitué un bouleversement majeur pour les opérateurs des sites pilotes en Allemagne et en Italie. J'ai dû prendre en compte cette résistance au changement dans la conception de mes scripts d'intégration. En m'assurant que la remontée des données était parfaitement transparente et sans perte d'information, j'ai contribué à instaurer un climat de confiance technique. Sur le plan du retour sur investissement (ROI), la réduction de la dette technique et l'automatisation du pipeline d'extraction ont transformé radicalement le processus de planification : là où un superviseur pouvait consacrer jusqu'à **deux heures chaque matin** à construire manuellement le planning de la journée, l'ordonnancement est désormais généré en **moins de cinq minutes** et recalculé en continu au fil des aléas de production. Rapporté aux deux à trois planificateurs de chaque site pilote, ce gain représente, en moyenne, plusieurs centaines d'heures libérées par an et par usine, réinvesties dans l'analyse des plannings plutôt que dans leur saisie. Je précise la nature de ces chiffres : il s'agit d'ordres de grandeur constatés lors de la mise en service des pilotes, et non d'une mesure contractuelle validée

par le client — leur consolidation fera partie du bilan officiel des sites pilotes. Je distingue également ce qui relève de l'outil de ce qui relève de mon travail : le gain de planification est d'abord celui de l'APS lui-même ; la contribution propre de l'intégration se mesure à la fiabilité du pipeline de données, sans laquelle aucun de ces gains n'est atteignable. Surtout, la consolidation de ces données standardisées offre désormais à la direction industrielle une capacité d'analyse inter-usines inédite, transformant une contrainte technique en un véritable levier de performance financière.

3.1.8 Bilan et rigueur analytique : l'importance d'une base solide

Les résultats de cette refonte sont directement mesurables et toutes mes décisions techniques ont été guidées par l'analyse des données. L'utilisation d'outils d'analyse statique a révélé une dette technique importante : le compteur d'erreurs de typage (Pyright) affichait initialement **323 erreurs**. J'ai utilisé le suivi de cet indicateur jusqu'à zéro comme critère objectif d'achèvement. De plus, la taille du fichier orchestrateur est passée de **1543 lignes à environ 130 lignes**, témoignant objectivement de l'effort de refactoring.

La mise en place de l'environnement de test m'a permis de développer **688 tests unitaires et 28 tests** d'intégration, automatisés via GitLab CI. Là encore, la donnée a orienté ma stratégie : constatant un contraste flagrant de couverture (86 % sur certains modules de traitement contre 0 % sur les appels API), j'ai inversé mon approche initiale pour prioriser les tests sur les zones à fort risque métier. En imposant cette rigueur analytique et architecturale, j'ai pu transformer un script fragile en un socle industriel robuste.

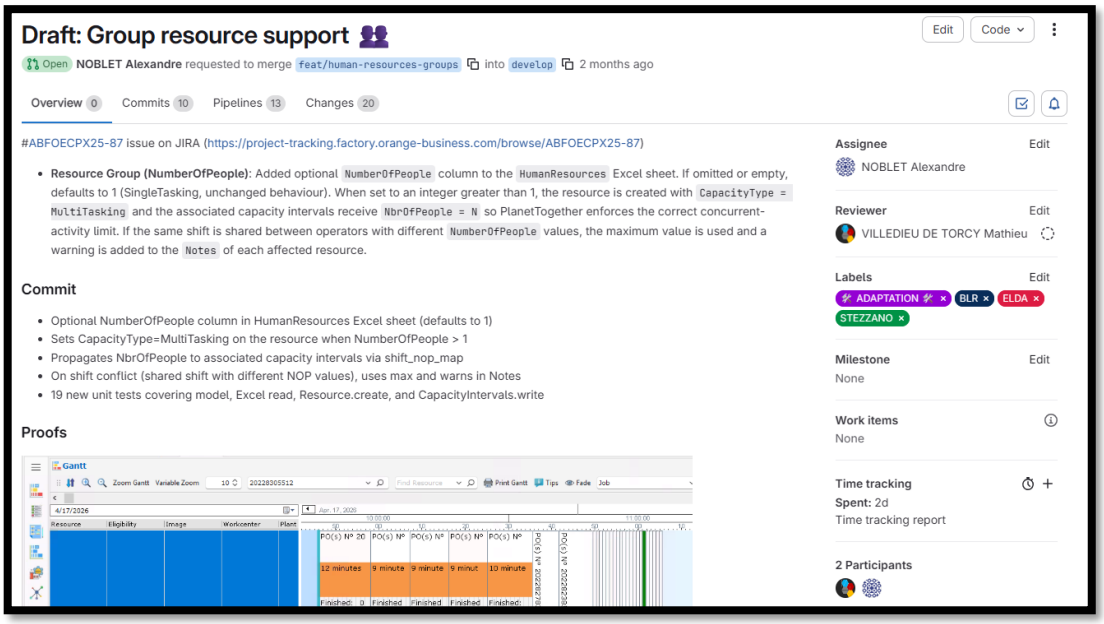


Figure 4. Merge Request sur GitLab pour une nouvelle fonctionnalité

Cette mission m'a fait prendre conscience qu'écrire du code qui fonctionne ne suffit pas. J'ai véritablement appris à concevoir un code maintenable et pérenne. Cette démarche d'architecture stricte et de tests systématiques fait directement écho aux enseignements d'architecture logicielle et de design patterns abordés en Mastère, confirmant que la théorie apprise en cours est le prérequis indispensable pour concevoir des solutions industrielles à grande échelle.

3.2 Mission 2 : Développement mobile multi-clients

3.2.1 Contexte et présentation de Smart Traceability

Cette seconde mission concerne un portefeuille d'applications mobiles développées en Flutter. J'interviens sur plusieurs projets distincts (Smart Traceability, LiveTag, LiveTick, Smart Search) avec des profils de contribution très variables. Le développement de l'application LiveTag ayant déjà été détaillé lors de mon rapport de stage (consultable sur mon portfolio : https://portfolio.alexandrenoblet.fr/blogs/stage-obs), je concentrerai cette section sur un autre projet majeur: **Smart Traceability**.

Smart Traceability est une application mobile de traçabilité conçue pour les environnements industriels et logistiques. Son rôle principal est de suivre des produits physiques et de valider leur conformité par rapport à un attendu théorique. Elle permet par exemple de vérifier le contenu d'un colis, de valider un bon de livraison ou de réaliser un inventaire en comparant les articles scannés à une liste préétablie. Pour cela, l'application s'intègre avec des équipements professionnels lourds, comme les terminaux RFID Zebra ou Axem. Elle gère la collecte de données sur le terrain, leur stockage local et leur synchronisation avec des systèmes distants. Ce flux physique est critique : la moindre erreur de synchronisation des identifiants ou des dates a un impact direct sur la chaîne logistique du client.

3.2.2 Architecture logicielle et Découplage (Clean Architecture)

L'architecture de l'application mobile repose sur une structuration en couches strictes, garantissant la maintenabilité et l'évolutivité du code source. Le projet est segmenté logiquement : une couche `data/` pour les entités et les contrats d'interface (repositories), une couche `modules/` isolant chaque fonctionnalité métier (boxes, inventories, items, authentication), et des couches transverses `services/` et `notifiers/` pour l'état applicatif.

La gestion d'état s'appuie sur le pattern Riverpod, intégré de manière transitive via notre package interne partagé `fl_sms_core`. Chaque fonctionnalité métier expose des notifiers (ex: `ClientConfigurationNotifier`, `BackupInventoryNotifier`) qui étendent `ChangeNotifier`. Le véritable découplage architectural se matérialise par l'injection de dépendances : les repositories sont définis par des classes abstraites (`IBoxesRepository`, `IItemsRepository`) injectées dans les notifiers via les providers Riverpod.

Cette abstraction permet de permuter dynamiquement l'implémentation sous-jacente selon le contexte de déploiement (défini par une variable d'environnement `Env.datasource`). L'application peut ainsi basculer de manière transparente entre une implémentation distante (`*.repository.impl.dart` s'appuyant sur Retrofit/Dio pour les appels API REST), une persistance locale (`*.repository.objectbox.dart`), ou un jeu de données factices pour les tests d'intégration (`*.repository.mock.dart`).

Ce découplage a un coût d'entrée réel : multiplication des interfaces et courbe d'apprentissage de Riverpod pour tout contributeur ponctuel. Je l'ai assumé pour deux raisons. D'une part, la cohérence avec le socle interne fl_sms_core, qui mutualise ces patterns sur l'ensemble du portefeuille d'applications de la Practice. D'autre part, le besoin structurel de permuter les sources de données selon les clients : c'est précisément cette abstraction qui a rendu possible la stratégie multi-clients décrite en 3.2.5 sans réécriture du cœur applicatif.

3.2.3 Résilience des Sessions et Persistance Locale (Offline)

La stratégie de gestion du mode déconnecté (Offline) a été pensée pour répondre au risque opérationnel majeur du terrain : la perte de données lors d'un inventaire en cours. Plutôt que de développer un système générique et complexe de synchronisation bidirectionnelle avec résolution de conflits algorithmiques, j'ai opté pour une approche asymétrique alliant réactivité et sécurité transactionnelle.

D'une part, les données de référence et de cache (items, métadonnées, événements historiques) sont stockées dans une base de données locale hautes performances, ObjectBox, qui a récemment remplacé Hive. Cette base alimente l'interface via des flux réactifs (`query().watch()`), garantissant un affichage instantané sans latence réseau. Les écritures transactionnelles lourdes (comme la validation définitive d'une boîte) contournent ce cache et interrogent directement le backend Spring Boot via le client HTTP Retrofit.

D'autre part, pour protéger le travail en cours, un filet de sécurité applicatif a été implémenté via le `BackupInventoryNotifier`. Ce module orchestre une sauvegarde automatique de la session de scan dans les `SharedPreferences` (au format JSON) exactement deux secondes après la détection du dernier tag RFID. En cas de crash inopiné de l'application ou de perte brutale de connexion, la session est gelée avec un délai d'expiration de deux heures. À la réouverture, l'opérateur se voit proposer une restauration automatique de son inventaire, avec un décompte précis des éléments récupérés. Ce mécanisme apporte une tolérance aux pannes ciblée sur le principal risque opérationnel du terrain : la perte d'un inventaire en cours.

3.2.4 Sécurité des données au repos et empreinte matérielle

Le déploiement d'applications mobiles en environnement logistique soulève une problématique critique : la sécurité des terminaux matériels. Les opérateurs utilisent des scanners durcis (Zebra, Axem) qui circulent physiquement dans les entrepôts et peuvent être égarés ou volés. Puisque j'ai mis en place une architecture permettant un stockage local via ObjectBox pour le mode hors-ligne, une quantité importante de données industrielles confidentielles (références clients, quantités, flux de production) réside directement sur l'appareil. Pour contrer ce risque, l'architecture a dû intégrer des principes de *Security by Design*. Il a fallu s'assurer que les bases de données locales soient chiffrées au repos et inaccessibles depuis l'extérieur du système d'exploitation Android. Par ailleurs, les choix d'architecture ont également pris en compte l'empreinte matérielle (Green IT). L'optimisation des requêtes locales au lieu de sollicitations réseau continues permet non seulement de préserver la batterie des terminaux industriels sur des cycles de travail de 8 heures, mais aussi de réduire la charge serveur globale, limitant ainsi l'empreinte carbone numérique de l'application.

3.2.5 Le défi du multi-clients : Cloud vs On-Premise

L'application étant très liée au métier, plusieurs versions existent. Mon objectif principal était de livrer une fonctionnalité complète de traçabilité RFID (appairage, association entre fiches, boîtes et engins, inventaires) pour un grand compte du secteur aéronautique. En parallèle, je devais adapter cette même base applicative pour un second client industriel, mais dans un contexte technologique radicalement opposé.

Le premier client utilise une architecture orientée Cloud multi-tenant. Le second exigeait un déploiement local (on-premise) avec une intégration complexe à son ERP Oracle. Pour gérer ce grand écart technique sans dupliquer inutilement le code, j'ai mis en place une stratégie stricte de gestion de versions.

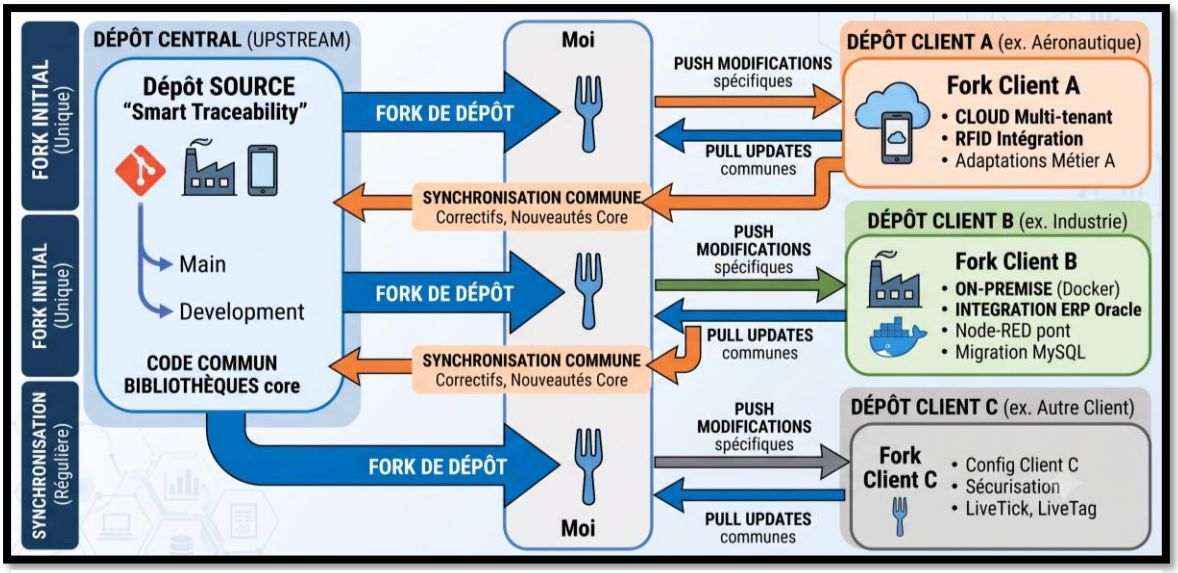


Figure 5. Stratégie de gitflow pour gérer les forks

J'ai géré **cinq dépôts Git** liés de manière simultanée. J'ai utilisé une stratégie de "fork" à partir d'un dépôt source commun (upstream) synchronisé régulièrement vers les dépôts clients. Cette méthode rigoureuse m'a permis de mutualiser les corrections de bugs tout en isolant les développements spécifiques et confidentiels de chaque usine.

Ce choix a été arbitré face aux alternatives classiques du multi-clients Flutter : les flavors (variantes de build au sein d'un même dépôt) ou un monorepo à compilation conditionnelle. Ces approches mutualisent davantage le code, mais chaque build embarque alors l'ensemble des sources, ce qui était incompatible avec le cloisonnement de confidentialité exigé entre clients ; elles introduisent en outre une complexité de configuration que rien ne justifiait à l'échelle de deux clients actifs. La stratégie de fork combinait donc simplicité et utilité : chaque dépôt reste un projet Flutter standard, immédiatement lisible par un contributeur ponctuel, tout en garantissant une isolation physique des développements spécifiques. J'en assume la limite : le coût de synchronisation avec l'upstream croît avec le nombre de clients, et au-delà d'une dizaine de dépôts, une bascule vers une architecture à flavors devrait être réévaluée.

3.2.6 Rigueur méthodologique et développement Full-Stack

Afin de sécuriser la production, la planification en amont a été primordiale. J'ai formalisé un **plan de refonte sur 11 semaines**. La première semaine a été exclusivement consacrée à la réalisation de trois preuves de concept (POC) techniques avant de lancer le moindre développement : le fonctionnement du scanner matériel, la structure de la table RFID en base de données et le flux événementiel. Ce choix de dé-risquer techniquement le projet a été décisif.

J'ai ensuite mené un développement full-stack coordonné en autonomie. Les fonctionnalités étaient livrées simultanément côté mobile en Flutter et côté backend en Spring Boot Java. Pour le second client industriel, j'ai dû piloter une migration technique très lourde. J'ai basculé l'architecture d'une base de données NoSQL (MongoDB) vers une base relationnelle (MySQL) en simplifiant le modèle de données. J'ai remplacé le pipeline Cloud habituel par une image Docker livrée directement chez le client. Enfin, j'ai intégré l'outil Node-RED pour faire le pont avec l'ERP Oracle existant.

3.2.7 Bilan factuel et qualification de la contribution

Afin de qualifier honnêtement mon niveau de responsabilité sur ces différents projets, je me suis appuyé sur l'historique des dépôts. Bien que le volume de commits ne soit pas l'unique mesure de la complexité d'un développement, il illustre fidèlement mon degré d'implication (ownership) selon les applications. Sur l'application mobile source, j'ai agi comme contributeur régulier, représentant **14 % des modifications** (63 commits sur 445). En revanche, j'ai assumé un rôle de référent sur les backends : je totalise **40 % des contributions** sur celui du client aéronautique (quasi-propriétaire des évolutions RFID) et 24 % sur celui du second client, où mes interventions portaient sur des tâches critiques comme la migration des bases de données. Côté qualité logicielle, la robustesse du nouveau module a été prouvée par la création de 14 fichiers de tests en Java (couches unitaires, d'intégration et contrôleurs) validant le code avant toute mise en production.

Cette mission s'est conclue par des phases d'ajustements itératifs basés sur les retours réels des utilisateurs sur le terrain, permettant par exemple de clarifier des confusions conceptuelles directement dans l'interface. Pour garantir la reprise du projet, j'ai réécrit l'intégralité de la documentation technique. J'ai ainsi démontré ma capacité à adapter un socle technique unique à des contraintes clients radicalement différents, validant le déploiement de solutions de traçabilité fiables en environnement industriel complexe.

3.3 Mission 3 : Audit industriel de modernisation

3.3.1 Contexte et enjeux d'une modernisation industrielle

Pour cette troisième mission, je suis volontairement sorti de ma zone de confort liée au développement logiciel pour mener un audit industriel en automatisme. J'ai accompagné un Client Industriel du secteur de la décoration industrielle qui exploitait un système de télémétrie vieillissant et fragile. La collecte de données reposait sur du matériel inadapté aux contraintes de l'usine et était gérée par un prestataire externe.

Le besoin initial était de remplacer cette installation obsolète par une architecture Edge Computing moderne. L'objectif était de remonter des indicateurs de production fiables en temps réel directement vers un système centralisé Cloud (MES). L'enjeu dépassait le simple remplacement matériel sur un site pilote. Il s'agissait de poser les fondations techniques d'une future usine en définissant une doctrine d'architecture standardisée, sécurisée et réplicable à grande échelle.

3.3.2 Méthodologie technique et traitement de la donnée

Ma mission a consisté à produire un document d'audit et de spécifications techniques servant de référence contractuelle. Pour cela, j'ai d'abord analysé le matériel existant sur le terrain. J'ai ensuite récupéré des fichiers de logs réels des machines pour en faire la rétro-ingénierie. Les données factuelles ont révélé une grande complexité : j'ai dû traiter trois formats de logs radicalement hétérogènes, représentant plus de **168 000 lignes de données** à analyser.

Sur cette base, j'ai conçu l'architecture cible de bout en bout. J'ai défini l'acquisition matérielle via des contrôleurs industriels spécifiques (Kunbus RevPi), la stratégie d'isolation électrique, puis le transport sécurisé des données via le protocole MQTT. Tout au long de ce processus, j'ai dû arbitrer des choix techniques complexes, comme la répartition de la puissance de calcul entre l'équipement local et le Cloud. Pour garantir la solidité de mes préconisations, j'ai systématiquement vérifié les capacités réelles du matériel dans les documentations des constructeurs au lieu de me reposer sur de simples suppositions.

3.3.3 Développement des compétences comportementales (Soft Skills)

Bien que le contexte soit très technique, cette mission a été un accélérateur pour mes compétences transverses. Elle a confirmé mon évolution vers une posture d'architecte fonctionnel. J'ai dû faire preuve d'une grande pédagogie pour vulgariser des concepts pointus (protocoles de transport, architecture événementielle) et les rendre accessibles à un client non spécialiste.

J'ai appris à naviguer entre des mondes distincts. Dialoguer avec le chef de production du client, le responsable commercial et les équipes d'intégration technique, chacun ayant des priorités souvent divergentes, a exigé de la diplomatie. J'ai notamment dû gérer un désaccord technique structurant sur la collecte des données du contrôleur Kunbus : je préconisais un flux poussé en MQTT depuis HiveMQ Edge vers la passerelle, tandis qu'une autre partie prenante défendait une interrogation en OPC-UA directement dans le contrôleur. Plutôt que d'imposer ma vue, j'ai assoupli ma préconisation en documentant objectivement les deux scénarios : avantages, limites et implications d'architecture.

3.3.4 Résultats, dimensionnement et approche par la donnée

La conduite de ce projet s'est faite de manière itérative : à la suite de la visite du site, six réunions de travail avec le client ont permis de recadrer progressivement le périmètre de l'audit et de faire valider chacune des versions intermédiaires (de la 1.0 à la 1.5), avant d'aboutir au livrable final d'une vingtaine de pages. Ce processus de validation itérative constitue en soi un indicateur de qualité : chaque version a été confrontée aux retours du client, garantissant que les préconisations finales reflètent la réalité opérationnelle du terrain plutôt qu'une vision théorique. Ce document d'audit encadre précisément le périmètre d'un Proof of Concept (PoC).

Au-delà du volume du document, c'est son contenu décisionnel qui en mesure la valeur. L'audit valide le déploiement sur six machines pilotes. Plus qu'un simple rapport technique, ce document fournit au client une matrice claire des responsabilités, un verdict de faisabilité net, et une véritable ligne directrice pour l'industrialisation de ses futurs sites de production.

À la date de rédaction de ce rapport, l'audit a été remis au client et est en cours d'instruction. J'assume cette absence de verdict immédiat, car elle fait partie intégrante de la réalité du métier de conseil : la valeur d'un audit ne se mesure pas au volume du livrable produit, mais à la décision qu'il permet. Le lancement effectif du PoC sur les six machines pilotes constituera donc le véritable indicateur de réussite de cette mission.

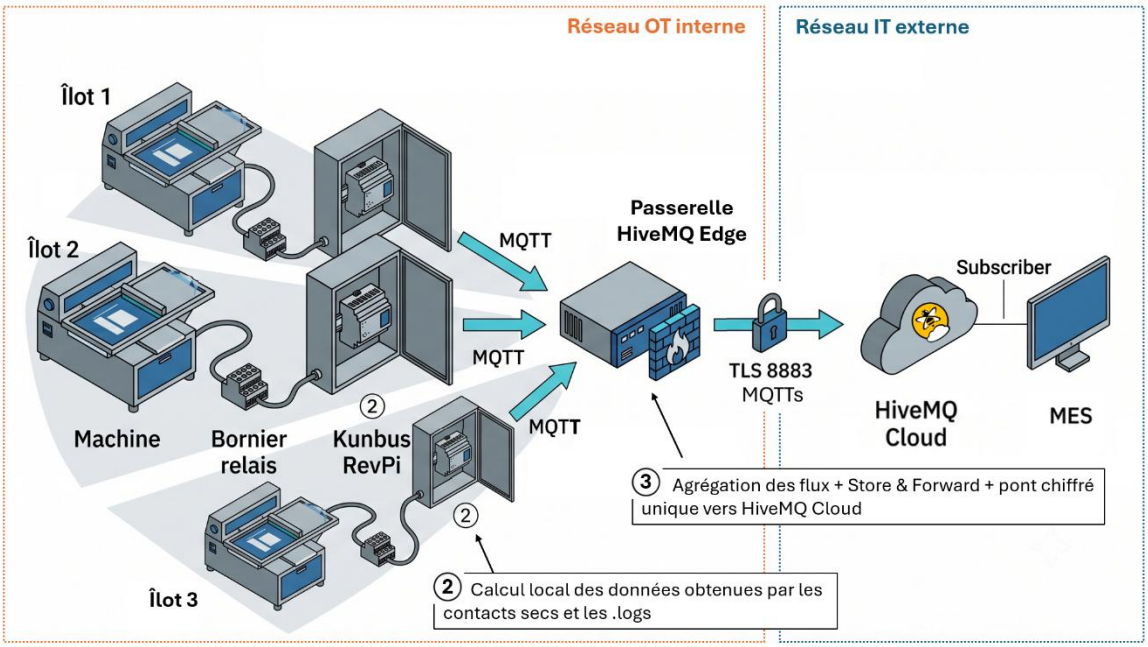


Figure 6. Architecture finale préconisée au client dans le cadre de l'audit

Cette mission m'a apporté des compétences qui dépassent le cadre strictement technique. À l'heure où l'intelligence artificielle générative automatise une part croissante de l'écriture de code, cette capacité d'analyse fonctionnelle, de vulgarisation et de dialogue inter-métiers constitue le véritable levier qui me permet de me démarquer et d'affirmer ma posture d'architecte.

4 Leadership et collaboration

4.1 Coordination technique sur le projet d'intégration

Sur le projet d'intégration pour le Client Industriel A, je me positionne comme le contact technique privilégié. Étant le seul référent technique permanent sur ce périmètre, je porte la responsabilité complète des décisions techniques et des choix d'architecture. Cette position comporte un revers que j'analyse avec lucidité : mes décisions ne rencontrent pas de contradiction naturelle au quotidien, faute de pair senior en mesure de les contester. Pour compenser cette absence de contre-pouvoir technique, je m'appuie sur des indicateurs objectifs (analyse statique, couverture de tests) plutôt que sur ma seule conviction, et je documente mes arbitrages structurants afin qu'ils restent auditables par un tiers. Ce rôle implique également de coordonner les collègues développeurs qui viennent prêter main-forte ponctuellement sur le code. Chaque contribution passe par une merge request dont je suis relecteur : j'y vérifie le respect des standards d'architecture mis en place, je formalise mes retours par des commentaires de revue, et je complète ce processus par des échanges oraux pour débloquer rapidement les points de friction. Je travaille en étroite collaboration avec mon chef de projet et tuteur, Charles LINSOLAS. Notre binôme est très complémentaire : il gère le cadrage stratégique et relationnel avec le client, tandis que je traduis ces besoins en solutions techniques tout en l'aidant à évaluer la charge de travail de façon optimale.

4.2 Autonomie et responsabilité totale sur le pôle mobile

La dynamique de collaboration est radicalement différente sur les projets mobiles. À la suite du changement de département de mon ancien tuteur, je suis devenu l'**unique référent sur ces technologies** au sein de la Practice. Je ne travaille plus sous la supervision directe d'un expert technique senior sur ce périmètre. J'assume seul la responsabilité de garantir la qualité du code, de maintenir les dépôts Git et d'assurer la livraison finale des applications (APKs) aux clients. Ce rôle d'ownership complet exige une rigueur organisationnelle constante pour respecter les engagements commerciaux tout en veillant au strict maintien des bonnes pratiques de développement logiciel.

Conscient que ce rôle d'expert unique représente un point de défaillance (SPOF) pour l'entreprise, j'ai mis en place une stratégie de transmission systématique. Cela se traduit par la mise à jour constante de la documentation technique (diagrammes d'architecture C4, READMEs) et par la création de guides d'onboarding pour tout collaborateur ponctuel. Cette documentation est le socle de ma gouvernance : elle assure que le système reste compréhensible et maintenable, même en cas d'indisponibilité de ma part.

4.3 Rituels et outils de communication

Pour assumer cette double posture (coordinateur d'un côté, référent autonome de l'autre), un système a été mis en place. J'utilise **Jira** au quotidien pour la gestion de projet. Cet outil me permet de découper les demandes complexes en tickets réalisables, de prioriser mon backlog et de donner une visibilité en temps réel sur l'avancement des développements à mon chef de projet.

En parallèle, la communication asynchrone via **Teams** est indispensable pour débloquer rapidement les collaborateurs ponctuels ou échanger avec les acteurs métiers. Ces échanges continus sont consolidés lors de nos **"Weekly"** (réunions hebdomadaires). Ce rituel de synchronisation est crucial : il me permet de remonter les points de blocage techniques à Charles LINSOLAS, d'ajuster les priorités de la semaine et de réaligner nos objectifs opérationnels avec les attentes du client.

4.4 Anticipation et gestion de la capacité de production

Être l'unique référent technique sur plusieurs projets fait de ma disponibilité une donnée critique de planification : chaque semaine de cours ou d'absence réduit mécaniquement la capacité de production du pôle mobile, sans relais possible à court terme. J'ai donc traité ma propre capacité comme une ressource projet à part entière. Concrètement, je projette mes absences plusieurs mois en amont dans notre outil de gestion des ressources, **TimeToClick**. Mon chef de projet peut ainsi positionner les jalons de livraison en dehors de mes périodes d'indisponibilité ou, à l'inverse, ajuster et renégocier une échéance client qui tomberait sur une semaine de cours. Cette discipline transforme une contrainte structurelle de l'alternance en paramètre maîtrisé du planning, plutôt qu'en risque subi.

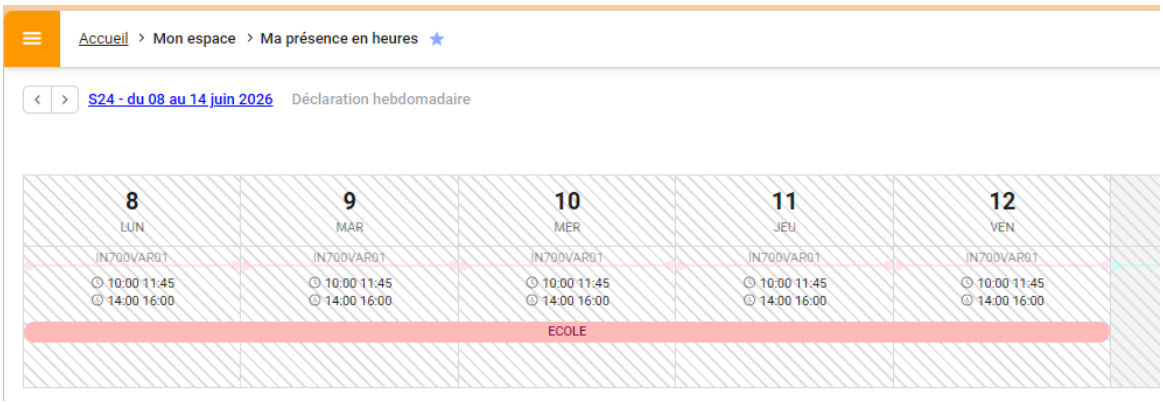


Figure 7. Extrait du logiciel TimeToClick

5 Résolution de problèmes complexes

5.1 Le défi structurel : charge de travail et échéances

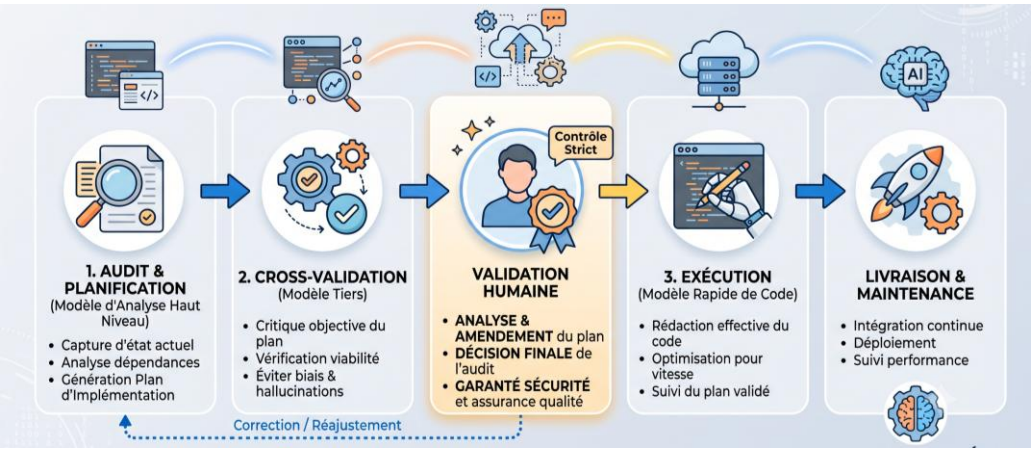
Le rythme inhérent à l'alternance impose des défis structurels importants. Mon principal problème complexe n'a pas été un bug technique isolé, mais la gestion systémique du temps et l'accumulation des échéances. À plusieurs reprises, les dates limites de livraison sur des projets distincts (par exemple, un jalon critique sur l'intégration industrielle et une livraison d'APK côté mobile) se sont rapprochées dangereusement, voire superposées. La contrainte de temps devenait un risque direct pour la qualité logicielle et le respect des engagements commerciaux de la Practice.

5.2 Cadrage managérial et sanctuarisation du temps

Face à ce goulot d'étranglement, la première étape de ma démarche a été organisationnelle. L'analyse de ma charge de travail a mis en évidence la nécessité d'optimiser mon temps de production effectif. J'ai instauré une communication accrue et transparente avec mon chef de projet. L'objectif était de prioriser drastiquement les tâches, de réévaluer ce qui pouvait être reporté sans impact majeur, et de réduire ma participation aux réunions non essentielles. Cette démarche a permis de sanctuariser des blocs de temps dédiés exclusivement au développement et à l'architecture.

5.3 Changement de paradigme : de développeur à architecte

Malgré cette optimisation calendaire, la charge de développement restait très lourde. J'ai donc opéré un changement de paradigme fondamental face à l'émergence des intelligences artificielles génératives. Au lieu de subir cette mutation technique, j'ai décidé d'assumer pleinement une fonction d'architecte-développeur. Plutôt que d'utiliser l'IA de manière rudimentaire (comme un simple générateur de fonctions), j'ai conçu un véritable flux de travail d'ingénierie logicielle basé sur un écosystème multi-modèles.



5.4 Une méthodologie stricte : Audit, Planification et Exécution

Ma méthode repose sur une séparation stricte des rôles assignés à différentes intelligences artificielles, sous mon contrôle exclusif. Mes requêtes initiales (prompts) ne sont jamais des commandes directes de génération de code. Le processus suit trois étapes inviolables :

1. **Audit et Planification (Modèle d'analyse haut niveau) :** Je sollicite d'abord un modèle d'IA reconnu pour ses capacités de raisonnement complexe. Je lui demande d'ingérer l'état actuel du code, d'identifier les dépendances et de générer un plan d'implémentation étape par étape.
2. **Cross-Validation (Modèle tiers) :** Pour garantir la viabilité de cette architecture et éviter tout biais cognitif ou "hallucination" algorithmique, je soumets ce plan à un second modèle de pointe, issu d'un écosystème technologique différent, afin d'obtenir une critique objective.
3. **Validation Humaine et Exécution (Modèle rapide) :** C'est ici qu'intervient ma valeur ajoutée d'architecte. Ma validation ne se limite pas à une relecture syntaxique. J'effectue une 'signature logique' : je vérifie que l'intention métier est préservée, que la sécurité est conforme aux standards internes, et que le code produit ne contrevient pas aux règles de gouvernance. Une fois validé par mes soins, je délègue la génération de l'implémentation finale à un modèle optimisé pour l'exécution, transformant ainsi l'IA en un simple moteur d'assemblage sous mon contrôle.

Pour mettre en œuvre cette méthodologie multi-modèles de manière sécurisée, je m'appuie sur **Dinootoo**, la plateforme d'intelligence artificielle interne d'Orange. Cet outil d'entreprise met à ma disposition la pluralité des modèles nécessaires pour effectuer mes phases d'audit, de validation croisée et d'exécution algorithmique. L'utilisation de cette plateforme interne est un prérequis indispensable : elle me permet de tirer parti de la puissance des dernières IA génératives tout en garantissant la stricte confidentialité du code et des données industrielles de nos clients, qui ne sont jamais exposés sur des services publics.

Cette méthode d'orchestration a **divisé mes temps de développement par un facteur de deux à trois** : une fonctionnalité standard qui exigeait auparavant deux à trois jours de travail est aujourd'hui livrée en une journée au maximum, conception, développement et tests inclus. Cet ordre de grandeur, constaté sur mes livraisons successives, reste une estimation personnelle et non une mesure instrumentée — je l'assume comme telle. C'est ce gain qui m'a permis de respecter les échéances, tout en conservant l'entière responsabilité de la sécurité et de la maintenabilité du code produit.

5.5 Résultats et limites du workflow multi-modèles

Si cette méthodologie a considérablement démultiplié ma vélocité de production, l'analyse de cette situation impose d'en souligner les résultats et limites opérationnelles. L'utilisation d'intelligences artificielles génératives, même orchestrée rigoureusement, introduit une part de non-déterminisme. À plusieurs reprises, les modèles ont généré des "hallucinations" algorithmiques ou proposé des bibliothèques obsolètes (un risque particulièrement élevé sur l'écosystème Flutter, qui évolue à un rythme soutenu).

De plus, la rédaction et la maintenance de "prompts" architecturaux complexes exigent un temps de préparation conséquent, ce qui réduit le retour sur investissement sur des tâches de développement mineures. Enfin, cette dépendance à la plateforme Dinootoo crée un point de défaillance unique (SPOF) : en cas d'indisponibilité ou de maintenance des serveurs internes d'Orange, la capacité de production chute brutalement. Ces limites confirment que l'IA reste un outil d'accélération, et que la validation technique humaine demeure le seul véritable rempart de sécurité.

Cette validation s'applique tout particulièrement à ma stratégie de tests, et la transparence impose ici une précision : la majorité des tests automatisés a été rédigée par les modèles. Pour neutraliser le risque de circularité qu'induirait un code validé par des tests issus du même outil, je conserve systématiquement la main sur ce qui fait la valeur d'un test : les cas métiers à couvrir, les valeurs limites et les scénarios d'échec sont spécifiés par mes soins avant toute génération. De même, aucun plan d'implémentation n'est exécuté sans avoir été relu et amendé par mes commentaires. C'est **cette relecture critique, et non la volumétrie de tests**, qui constitue la véritable garantie de qualité.

5.6 Enjeux légaux et propriété intellectuelle liés à l'IA

Au-delà des limites purement techniques des modèles d'intelligence artificielle, l'adoption de ce workflow a imposé une réflexion rigoureuse sur les risques juridiques. Le code généré par une IA pose la question épineuse de la propriété intellectuelle et de l'origine des données d'entraînement. Si un modèle génère un algorithme soumis à une licence open-source stricte et que je l'intègre directement dans le code propriétaire d'un de nos clients, j'expose l'entreprise à un risque légal majeur (contamination virale de la licence). C'est ici que ma fonction d'architecte prend tout son sens institutionnel. L'utilisation de la plateforme interne Dinootoo est une première barrière de sécurité pour la confidentialité, mais elle ne dispense pas d'une revue critique. Ma validation humaine, située à l'étape 3 de mon processus de développement, n'est pas uniquement fonctionnelle : elle est aussi légale. Je m'assure que le code exécutif généré reste cantonné à des algorithmes standards ou à l'implémentation de modèles de domaine dont la logique intellectuelle m'appartient, garantissant ainsi à nos clients la pleine et entière propriété de la solution livrée.

6 Bilan personnel et professionnel

6.1 De l'exécution à l'orchestration architecturale

Ce rapport met en évidence une évolution charnière dans mon parcours. Au fil de cette année, ma posture s'est radicalement transformée : je suis passé d'un profil de **développeur exécutant** à celui d'**architecte fonctionnel et technique**. J'ai pris conscience que le cœur de mon métier n'est plus circonscrit à la simple écriture de lignes de code, mais réside dans la capacité à concevoir des systèmes viables, robustes et maintenables.

Ce bilan professionnel intègre nécessairement l'analyse de mes difficultés ou échecs, qui ont été de puissants moteurs d'apprentissage. Lors de la phase initiale de la mission d'intégration PlanetTogether, j'ai sciemment accepté d'accumuler de la dette technique pour respecter les délais commerciaux extrêmement serrés des pilotes en Allemagne et en Italie. Le script d'extraction s'est ainsi transformé en un monolithe fragile.

L'erreur ne résidait pas dans ce choix initial, mais dans ma mauvaise évaluation de l'effort de remédiation ultérieur. Lorsque le besoin s'est complexifié avec l'ajout de nouveaux sites, chaque modification entraînait des régressions inattendues, menaçant la stabilité de la production. J'ai compris à mes dépens que repousser le *refactoring* génère une charge de travail exponentielle. Cette situation critique m'a contraint à bloquer les déploiements pour réécrire l'architecture de zéro. Cette difficulté a été un tournant dans mon évolution professionnelle: elle a forgé ma posture actuelle d'architecte, me poussant à refuser catégoriquement tout développement non pérenne si une doctrine de standardisation n'est pas clairement définie et validée en amont.

Une seconde difficulté, à plus petite échelle, relève du même travers. Sur le projet Smart Traceability, une fonctionnalité m'avait semblé suffisamment claire pour que, par impatience, je me lance directement dans le développement sans reformuler le besoin auprès du métier. J'étais en réalité parti sur une mauvaise compréhension : ce que j'avais développé ne répondait pas au besoin réel, et environ deux jours de travail ont dû être abandonnés. La perte est modeste en volume, mais la leçon est structurante : une heure de cadrage en amont coûte toujours moins cher qu'un développement à refaire. Depuis, je m'impose de reformuler systématiquement le besoin et d'obtenir une validation explicite avant d'écrire la première ligne de code. Là encore, c'est un réflexe d'architecte qui s'est substitué à celui du développeur pressé de produire.

6.2 Le rebond face à l'IA : d'une menace à un avantage compétitif

Cette prise de conscience a été catalysée par la démocratisation fulgurante de l'intelligence artificielle générative. Face à cette mutation technologique, j'ai acquis une conviction, pessimiste peut-être, mais réaliste : le métier de développeur "traditionnel", réduit à la seule production de code, est voué non pas à être supprimé, mais à se transformer en profondeur. Au lieu de percevoir l'IA comme une menace, j'ai choisi d'opérer un rebond stratégique pour maintenir et accroître ma compétitivité. J'ai délégué la production brute de code à la machine pour me hisser au niveau de superviseur. En concevant des workflows multi-modèles stricts, j'utilise aujourd'hui l'IA comme un puissant moteur d'exécution, tandis que je conserve le rôle exclusif d'architecte, garant de la sécurité et de la logique métier.

6.3 La valeur ajoutée humaine : l'intelligence de situation

Cette nouvelle dynamique m'a prouvé que ma véritable plus-value se situe désormais là où l'IA s'arrête. Comme l'a particulièrement démontré ma mission d'audit en automatisme industriel, la différence se fait sur l'intelligence de situation, la diplomatie, le dialogue inter-métiers et la compréhension profonde des enjeux terrain d'un client. Traduire un besoin métier complexe et parfois ambigu en une architecture logicielle précise est une compétence où l'humain conserve, à ce jour, un avantage décisif que les modèles génératifs ne parviennent pas à égaler.

6.4 La synergie avec les enseignements du Mastère

Dans ce contexte de forte évolution de notre industrie, les enseignements dispensés au sein du Mastère 1 ont pris tout leur sens. L'apprentissage de l'autonomie, exigé au travers des différents modules de l'école, a été le levier principal de ma réussite en entreprise. Les enseignements d'architecture logicielle et de design patterns ont été directement mobilisés sur le terrain : le pattern Repository et l'injection de dépendances structurent le découplage de Smart Traceability, la modélisation C4 documente l'architecture du pôle mobile, et le module CI/CD a directement inspiré l'automatisation de mes 716 tests via GitLab CI. C'est ce bagage théorique qui me donne aujourd'hui la hauteur nécessaire pour encadrer efficacement les IA génératives. J'ai constaté sur le terrain que l'important n'est plus d'être une encyclopédie de la programmation, mais de maîtriser la pensée systémique. C'est cette prise de recul qui me permet aujourd'hui d'assumer avec confiance le rôle de référent technique sur mes projets industriels et mobiles, et d'aborder la suite de ma carrière avec une longueur d'avance.

Au terme de cette année, je suis en mesure de répondre à la problématique posée en introduction. **Oui, un développeur-intégrateur peut garantir la maintenabilité logicielle et la résilience des architectures face à l'urgence des déploiements industriels, mais à trois conditions que mes missions ont éprouvées**. D'abord, traiter la dette technique comme un engagement explicite : l'accepter consciemment pour tenir un délai est un choix légitime, à condition d'en chiffrer la remédiation dès le départ. Ensuite, piloter la qualité par des indicateurs objectifs (analyse statique, couverture de tests, revues systématiques), au même rang que les jalons commerciaux. Enfin, adopter une posture d'architecte qui délègue l'exécution pour concentrer la valeur humaine sur la conception, l'arbitrage et la validation. En définitive, ce n'est pas l'urgence qui menace la maintenabilité : c'est l'**absence de doctrine architecturale**. C'est cette doctrine que j'ai construite cette année, et c'est elle que je continuerai d'appliquer en Mastère 2.

6.5 Perspectives professionnelles : Cap sur le Mastère 2 et l'intégration chez Orange Business

L'entrée en classe de Mastère 2, dès le mois d'octobre prochain, marquera l'ultime étape de mon cursus académique. Cette dernière année d'alternance a pour vocation de consolider définitivement ma posture d'architecte logiciel et d'expert en convergence OT/IT. Fort des responsabilités acquises sur les projets industriels complexes (comme le déploiement de PlanetTogether ou l'audit Edge Computing) et du niveau d'autonomie atteint dans la gestion du pôle mobile, mon ambition pour les mois à venir est très claire : m'imposer comme un élément moteur et stratégique de la *Practice Smart Tracking*.

Au-delà de la validation de mon diplôme, mon objectif principal est de transformer cette alternance en une intégration pérenne au sein d'Orange Business. Je souhaite continuer à concevoir des architectures logicielles résilientes, piloter la transition de nos clients vers des solutions hybrides (Edge/Cloud), et industrialiser nos méthodes de développement grâce à l'intelligence artificielle. Cette dernière année sera donc entièrement dédiée à la création de valeur ajoutée pour l'entreprise et à la sécurisation de nos livraisons, avec pour ligne de mire de m'ancrer durablement et de me faire une place de choix au sein des équipes de Cécile DELERY et Ghislaine LEHMANN.