INSTITUT MÉDITERRANÉEN
D'ÉTUDES ET DE RECHERCHE EN
INFORMATIQUE ET ROBOTIQUE

Qualification et industrialisation d'une solution d'API Gateway

Mémoire de fin d'études

Diplôme de niveau VII (BAC+5) reconnu par l'État de
"Manager de systèmes Informatique et Robotique"

Réalisé par : Alexandre NEGREL

Maître d'apprentissage : Laurent SCALISI

Tuteur école : Éric SALVAT

Promotion : Milles 2020/2023

Validation par l'entreprise :

Résumé

Ce mémoire constitue une investigation approfondie de la mission solution d'API Gateway au sein de l'entreprise Française Des Jeux (FDJ) et plus particulièrement son entité technologique, le Tech-Lab. Les objectifs de ce document sont multiples et chaque chapitre met en lumière des aspects essentiels de la mission.

L'introduction offre une vue d'ensemble de l'entreprise, de ses opérations et du contexte industriel qui la régit, tout en soulignant les contraintes métiers qui en résultent. De plus, elle établit le rôle du Tech-Lab, sa position dans l'entreprise, expose son mode de fonctionnement et ses activités. L'objectif de ce chapitre est de créer un cadre contextuel pour la FDJ et le Tech-Lab, tout en situant ma propre intégration au sein de l'entreprise.

Le second chapitre est consacré à la présentation de la mission solution d'API Gateway. Il aborde premièrement le besoin à l'origine de cette mission suivi d'une description technique et fonctionnelle. Par la suite, les contraintes spécifiques qui émergent du contexte industriel présenté précédemment y sont abordées.

Le troisième chapitre présente la méthodologie de projet suivie lors de la mission. Les contraintes méthodologiques du Tech-Lab qui découlent de sa position sont examinées pour apporter une perspective approfondie des défis qui doivent être relevés. Chaque étape, de l'expression du besoin à la livraison en passant par la mise à niveau et l'industrialisation, est exposée et justifiée en détails.

Le quatrième chapitre met en lumière quelques-unes de mes contributions spécifiques à la mission. Certains de mes travaux, notamment sur les tests fonctionnels, d'exigences non fonctionnelles et de performance, sont présentés en détail, montrant mon acculturation à d'autres aspects de l'ingénierie logicielle et ma capacité à travailler dans un milieu industriel.

En conclusion, ce mémoire offre une analyse approfondie et structurée de la mission solution d'API Gateway au sein de la FDJ. Il démontre la démarche méthodique suivie pour atteindre les objectifs fixés et met en évidence mes contributions spécifiques à cette entreprise.

Remerciements

À l’heure de conclure cette étape importante de mon parcours académique, je tiens à remercier chaleureusement toutes les personnes qui ont contribué à la réalisation de mon mémoire de fin d’étude.

Premièrement, je tiens à exprimer ma profonde gratitude à Laurent Scalisi, mon tuteur, pour son implication dans ma formation et son engagement sans faille. Ses relectures attentives et ses précieux commentaires ont grandement enrichi mon travail. Sa disponibilité et ses précieux conseils m'ont permis de bénéficier de périodes de stages complémentaires visites enrichissantes au Centre d'Ingénierie de Performance (CIP) et autres services de l’entreprise. Nos réunions régulières ont été l'occasion de partager nos idées et d'approfondir certains aspects essentiels de mon mémoire.

Je souhaite également exprimer toute ma reconnaissance envers Christian Boitel, expert technique, qui a généreusement partagé son savoir et consacré de nombreuses heures à répondre à mes questions. Sa disponibilité sans faille m'a été d'une aide précieuse et m'a permis de progresser sur de nombreux aspects.

Un grand merci à Romain Guerin pour ses relectures attentives, ainsi que pour le partage de ses connaissances approfondies sur Git, les tests fonctionnels et d’exigence non fonctionnelles. Ses relectures ont été d’une grande aide dans la finalisation de mon mémoire.

Je tiens à remercier tout particulièrement Florian Torres pour sa contribution à mon mémoire. Sa vision totalement différente de l'informatique et son approche pour résoudre les problèmes ont été particulièrement pertinentes et ont permis d'élargir mes horizons.

Je remercie également Cyril Dupont pour ses relectures attentives et son partage de connaissances.

Un merci tout particulier à Yannick Tison pour sa relecture, qui a permis d'améliorer la qualité de mon mémoire.

Enfin, je souhaite exprimer ma plus profonde gratitude à Paola, ma compagne, et à ma mère pour leur soutien et leurs relectures attentives, leurs corrections minutieuses et leurs conseils avisés ont été d'une aide précieuse dans la rédaction de mon mémoire.

Je tiens à souligner que ces remerciements ne suffisent pas à exprimer toute ma reconnaissance envers chacun de vous. Votre implication, votre soutien et votre expertise ont été essentiels dans la réalisation de mon mémoire et dans mon parcours académique. Vous avez largement contribué à mon développement personnel et professionnel.

Je souhaite aussi adresser mes remerciements à toutes les personnes qui m'ont soutenu de près ou de loin tout au long de mon parcours, ainsi qu'à tous ceux qui ont participé à mon apprentissage.

Merci infiniment à vous tous.

Table des matières

1 - Introduction

1.1 - La Française Des Jeux

Ce chapitre traite de la Française Des Jeux, ou **FDJ**, une entreprise française historique née au XXème siècle. Il aspire à offrir une rétrospective des origines et de l'évolution de l'entreprise, mettant en lumière les moments clés qui ont façonné son parcours jusqu'à sa position de leader incontesté de l'industrie du jeu en France. Sa raison d'être, son impact et sa contribution à la société contemporaine seront détaillés afin de fournir un cadre complet de ce qu'est l'entreprise de nos jours. Enfin, les objectifs à court terme, l'expertise technologique et l'organisation interne de l'entreprise permettront de présenter la direction vers laquelle elle évolue.

1.1.1 - Histoire

L'histoire débute en 1931 lorsque l'Union des Blessés de la Face et de la Tête (**UBFT**) crée une souscription nationale assortie d'une tombola nommée "La Dette".

Figure 1 : Billet de souscription nationale "la dette"

L'association fondée en 1921 par Bienaimé Jourdain, Albert Jugon et le Colonel Picot est dévouée à aider et soigner les blessés de guerre. Leur objectif était d'apporter un soutien moral mais aussi

financier afin de pallier le défaut d'aide sociale. En effet, les Gueules Cassées étant atteintes de séquelles à la tête, l'État les considérait aptes à travailler. Néanmoins, malgré les dons et les legs, les revenus restent insuffisants car non récurrents. C'est ainsi qu'en 1931, l'UBFT ainsi que trois autres associations dédiées aux blessés de guerres lance "La Dette". C'est le succès considérable de cette tombola qui va inciter l'État à créer, par une loi de finance, une véritable loterie nationale. Le premier tirage a eu lieu le 7 novembre 1933 en présence de cinq mille personnes. Néanmoins, ce n'est pas le succès commercial escompté, avec un prix jugé trop élevé, les billets valant cent francs de l'époque ne se vendent pas. L'UBFT a alors l'idée d'acheter ces billets entiers à l'État, de les fractionner en dixièmes et de les vendre. Ces derniers sont vendus par des veuves ou des mutilés de guerre. Les bénéfices permettent à l'association de financer l'aide aux blessés de tous les conflits. Suite à cela, les tirages se multiplient, il y en a un chaque mois et pour les occasions particulières tel qu'un vendredi 13, la Saint-Valentin, etc.

Jusqu'en 1945, il y avait une forme de symbiose entre l'État qui apporte une sécurité juridique des tirages et l'association qui fractionne, popularise et vend les billets. Après la Seconde Guerre mondiale, le gouvernement de Gaulle institue le régime général de la sécurité sociale par ordonnance. Ce nouveau régime social dont bénéficient les Gueules Cassées perturbe l'équilibre et remet en cause la légitimité de l'UBFT à participer au monopole de la loterie. L'UBFT qui ne cesse de se réinventer, se lance alors dans la création de centres de loisirs, de solidarité et de maisons de retraites pour des veufs et invalides de guerre.

En 1954, le Pari Mutuel Urbain (**PMU**), un service de pari accessible en dehors de champs de courses, lance un nouveau type de pari : le Tiercé. Celui-ci se joue à trois chevaux et s'avère gagnant si l'on trouve les trois premiers de la course, et cela, peu importe l'ordre. Devant le succès fulgurant de ce nouveau loisir, la popularité de la loterie nationale se voit périclité.

Une fois de plus, l'UBFT doit se repenser et décide de parcourir, avec d'autres associations d'anciens combattants, l'Europe et les États-Unis pour dénicher des nouveaux jeux qu'ils pourraient importer. C'est de cette manière que le Loto fût découvert en Allemagne et importé en France. Dès le début, les joueurs sont séduits, car, contrairement à la loterie, on choisit ses numéros et donc son destin.

En 1976, la Société de la Loterie Nationale et du Loto National (SLNLN) est créée à la suite du lancement du Loto national. L'État et les émetteurs détenaient respectivement 51 % et 49 % du capital. L'association fût et demeure encore aujourd'hui l'actionnaire privé le plus important.

En 1989, la société est renommée France Loto. La participation de l'État passe de 51 % à 72 % tandis que celle des émetteurs est réduite à 20 %. Deux ans plus tard, la société change à nouveau de nom et devient la Française Des Jeux.

Figure 2 : Logo de la Française Des Jeux depuis 2009

De sa création jusqu'en 2010, la Française Des Jeux avait le monopole des jeux d'argent et des paris sportifs (hors paris hippiques) en France. Cela incluait, en plus de la France métropolitaine, les DROM-COM (anciennement DOM-TOM) et la principauté de Monaco. Depuis le 12 mai 2010, la loi 2010-476 a ouvert à la concurrence les jeux d'argent et paris sportif **en ligne**, néanmoins, les jeux de hasard sont toujours en monopole.

En 2017, notre président Emmanuel Macron et son gouvernement amorce la privatisation de la Française Des Jeux. Il faudra attendre deux ans et la loi relative à la croissance et la transformation des entreprises pour acter l'entrée en bourse de FDJ, le 7 novembre 2019. L'État qui détenait jusqu'alors 72 % du capital, ne conserva que 20 % des actions.

Suite à cette privatisation, la Française Des Jeux s'est réorganisée autour de cinq piliers : l'offre de jeux, le modèle responsable, l'engagement sociétal, l'ancrage territorial et la durabilité. Sa nouvelle raison d'être, approuvée par l'assemblée générale en 2020, reflète ses ambitions :
"Le jeu est notre métier, la contribution à la société notre moteur et la responsabilité notre exigence."

1.1.2 - Raison d'être

Aujourd'hui, la Française Des Jeux est le premier opérateur de jeux en France avec plus de 25 millions de joueurs chaque année. C'est plus de 20 milliards d'euros qui ont été misés en 2022 et 2,5 milliards d'euros de chiffre d'affaires. Le jeu est et restera le métier de l'entreprise qui a renouvelé son monopole en 2019 pour les 25 prochaines années.

Le groupe FDJ, qui comprend l'entreprise et ses filiales, est reconnu internationalement pour ses jeux. Il a su diversifier ses activités et vend ses services à des entreprises étrangères (Loterie Romande en Suisse, loterie canadienne, etc). À ce jour, FDJ est la seconde loterie d'Europe et la quatrième mondiale. Le groupe est composé de huit filiales directes, dont certaines basées à l'étranger, qui servent son activité principale : le jeu.

Malgré sa croissance, l'entreprise a conservé ses valeurs éthiques et sociales et redistribue la majorité de son chiffre d'affaires. En effet, 68,4 % des mises sont redistribuées aux gagnants, plus de 4,4 milliards d'euros contribuent au budget de l'État et les détaillants sont rémunérés à hauteur de 965 millions d'euros. De plus, l'entreprise crée ou pérennise plus de 50 000 emplois, elle contribue significativement à l'économie française.

Entreprise populaire, enracinée dans les territoires, FDJ se mobilise aussi au service de la préservation du patrimoine en France. C'est plus de 125 millions d'euros qui ont été collectés depuis 2018 et 450 sites ont d'ores et déjà été restaurés ou sont en cours de restauration. L'entreprise, soucieuse de l'environnement, investit également pour préserver plus de 100 hectares de forêts en France.

La fondation FDJ, créée il y a exactement 30 ans, marque l'engagement solidaire du groupe. Lors des cinq dernières années, la fondation a financé, accompagné et soutenu plus de 400 associations dans les territoires, partout en France, pour trouver avec elles des solutions adaptées aux enjeux locaux et aux besoins des populations. Pour les cinq prochaines années, c'est une enveloppe de 25 millions d'euros qui va être investie dans l'égalité des chances et, plus précisément, dans l'éducation.

Enfin, pour promouvoir une pratique récréative du jeu d'argent, l'entreprise place au cœur de ses préoccupations l'accompagnement de ses clients, l'intégrité de ses jeux et la réduction des risques et des conséquences liés à son activité.

La lutte contre le jeu des mineurs est le premier pilier de la politique de jeu responsable de FDJ. Le rappel de l'interdiction du jeu des mineurs s'appuie sur plusieurs vecteurs d'information tels que des affiches, logos et films diffusés sur l'écran caisse et brochures. Environ 10 % du budget média est consacré à des campagnes sur la prévention du jeu des mineurs. Par ailleurs, tous les détaillants sont formés à la prévention du jeu des mineurs et testés régulièrement.

Le second pilier est la lutte contre le jeu excessif. Avant d'être lancés sur le marché, tous les jeux sont analysés afin de mesurer leur niveau d'attractivité, que ce soit au niveau du design ou de la mécanique de jeu. FDJ met aussi à la disposition des joueurs des outils leur permettant de se fixer des limites, évaluer et contrôler leur pratique de jeu.

Par ailleurs, des organismes externes, tel l'Autorité Nationale des Jeux (ANJ), font des audits pour contrôler les actions mises en place par FDJ et les autres acteurs de jeux d'argent. L'ANJ a pour missions de prévenir le jeu excessif, les activités frauduleuses et assurer l'intégrité des opérations de jeu.

La raison d'être que nous venons de voir est le socle de la stratégie du groupe à horizon 2025.

1.1.3 - Objectifs horizon 2025

Cette section décrit les quatre axes sur lesquels le groupe souhaite progresser pour l'horizon 2025.

Le premier axe est la digitalisation de la loterie en développant l'omnicanalité afin de proposer une expérience de jeu adaptée à l'évolution des usages. FDJ distribue ses jeux dans une approche omnicanale et poursuit sa stratégie de fidélisation fondée sur une politique relationnelle enrichie grâce à une meilleure connaissance client.

Le second axe est de conquérir des parts de marché sur les paris sportifs en ligne, tout en maintenant une dynamique de croissance en point de vente. L'entreprise accélère le développement d'une offre distinctive et compétitive et la mise en œuvre d'une relation client plus personnalisée. Dans le même temps, FDJ souhaite renforcer l'attractivité des paris sportifs en point de vente par l'élargissement de l'offre, l'amélioration des produits et services et le renforcement du rôle prescripteur des détaillants.

Le troisième axe est la construction d'une relation client créatrice de valeur, tout en renforçant sa politique de jeu responsable, via l'identification et la connaissance des clients. L'entreprise souhaite mettre en place des offres et des services exclusifs, des parcours optimisés et des bénéfices relationnels. La meilleure connaissance des clients servira à leur offrir une expérience personnalisée, tout en favorisant le développement d'une approche de jeu responsable, différenciée et proportionnée, en fonction du comportement de chacun.

Enfin, le dernier axe vise à renforcer la résilience de son modèle économique en développant de nouvelles activités. Le Groupe a pour ambition de capitaliser sur ses actifs technologiques et sur sa notoriété, ainsi que sur son réseau de distribution pour développer de nouvelles activités et rendre son modèle plus résilient et durable. D'une part sur le marché B2B à l'international dans lequel il souhaite construire une position d'acteur de référence, avec une offre de services pour les opérateurs de loterie ou de paris sportifs, et d'autre part, sur le paiement et les services en point de vente. FDJ a par ailleurs pour objectif de se développer dans le secteur du divertissement, en dehors des jeux d'argent.

Voyons maintenant l'organisation opérationnelle mise en place par FDJ afin de mettre en œuvre sa stratégie.

1.1.4 - Organisation interne et expertise technologique

L'entreprise effectue chaque année des réorganisations, dans le but de, progressivement, tendre vers une organisation adaptée à ses objectifs actuels.

Aujourd'hui, l'entreprise est en partie organisée autour d'unités opérationnelles (BU ou business unit en anglais) qui conduisent des activités relevant de la régulation des jeux d'argent et de hasard par l'ANJ. Il y a deux BU, une pour la loterie et une pour les paris sportifs.

Il y a aussi trois unités opérationnelles en développement (ABU ou acceleration business unit en anglais), qui ont la responsabilité d'activités encore peu matures, hors régulation des jeux d'argent et de hasard par l'ANJ, et disposant d'enjeux spécifiques. Ce sont les ABUs "international", "divertissement" et "paiement et services".

L'entreprise est aussi construite autour de trois fonctions transverses qui pilotent la déclinaison opérationnelle de la stratégie dans une logique d'optimisation des ressources. Elles ont pour rôle de faciliter la mise en œuvre de la stratégie. Ce sont les fonctions clients, commerciales et technologie.

Enfin, les fonctions *corporate* ont pour rôle de définir la politique générale du Groupe et de garantir sa cohérence globale. Au sein de ces fonctions *corporate*, se trouvent notamment la direction de la Communication et la direction de l'Innovation.

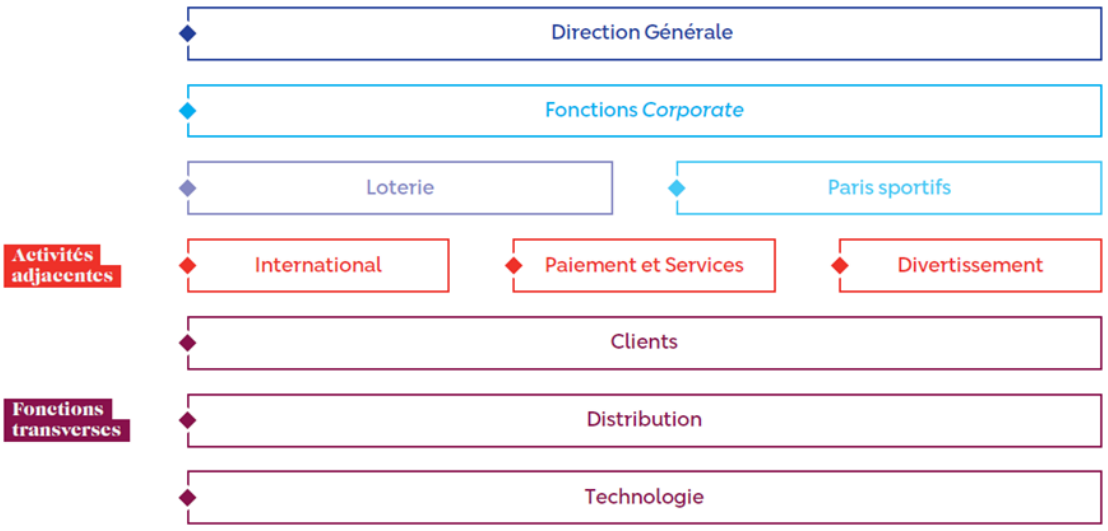


Figure 3 : Schéma organisation opérationnelle FDJ

L'organisation opérationnelle est retranscrite géographiquement sur les différents sites de la maison mère.

Le siège social se situe à Boulogne-Billancourt, proche de Paris. On y retrouve la direction générale et les fonctions corporate. La fonction transverse clients ainsi que les activités adjacentes (BU et ABU).

Le site de Saint-Witz et Saint-Mard sont des entrepôts logistiques. Ils concentrent plusieurs chaînes de production logistique de qualité industrielle. Ces sites se partagent la fonction transverse distribution avec le siège social.

Enfin, nous avons le site de Vitrolles, proche de Marseille, qui concentre le pôle technologique de FDJ. La quasi-totalité de l'expertise technique est à Vitrolles. Le datacenter privé composé de deux salles redondantes de FDJ est sur ce site.

Étant originaire et habitant Marseille, mon alternance se déroule à Vitrolles. Le service que j'ai intégré fait partie de la fonction transverse Technologie. On retrouve, au sein de cette fonction, le groupe cybersécurité, la Direction des Systèmes d'Information (DSI) et la Direction Technique (DT).

Le groupe cybersécurité garantit la protection de toutes les activités métiers du Groupe FDJ en s'appuyant sur l'ensemble des compétences sécurité du Groupe. En plus d'assurer le recensement des risques cyber à la direction, ils mettent en place de campagnes de sensibilisation à l'hameçonnage (phishing en anglais), de communication sur les bonnes pratiques, etc.

La DSI répond aux besoins applicatifs de tout le groupe ainsi que d'elle-même. Que ce soit en ligne ou en point de vente, la DSI développe ou achète des applications nécessaires à la création de nouveaux jeux, mais assure aussi les évolutions des applications déjà en place.

La DT assure le maintien en condition opérationnelle des systèmes applicatifs développés ou achetés par la DSI. Elle fournit l'ensemble des services d'ingénierie nécessaires à l'homologation fonctionnelle, technique et de performance, et à l'industrialisation des solutions SI au service des

projets de la roadmap et des systèmes en production. L'équipe que j'ai intégrée, le Tech-Lab, dépend directement de la direction technique.

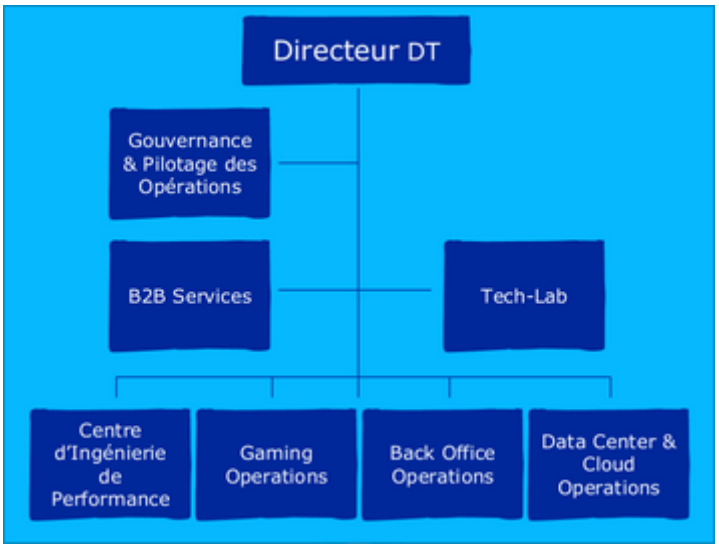


Figure 4 : Diagramme de la direction technique

Avant de s'intéresser à l'équipe qui m'a accueilli pendant ces trois années, il est important de souligner les enjeux et contraintes du pôle Technologie.

Aujourd'hui, la Française Des Jeux est également une entreprise technologique. L'ensemble du trafic généré par son réseau de détaillants, ses applications et sites web, est traité à Vitrolles.

Pour rappel, FDJ dispose du plus grand réseau de distribution français, avec un point de vente accessible en moins de dix minutes partout en France. Il n'est pas rare qu'une application doive traiter un pic de charges à plus 1000 transactions par seconde.

Au-delà du fort taux transactionnel, l'entreprise a de multiples contraintes métier régulées par l'Autorité Nationale des Jeux, la Commission Nationale de l'Informatique et des Libertés (CNIL) et la World Lottery Association (WLA).

Par exemple, l'ANJ contraint les entreprises de jeux d'argent, dont FDJ, à écrire l'ensemble des inscriptions, transactions et actions utilisateurs dans un coffre-fort électronique auquel eux seuls ont accès. De cette manière, ils peuvent surveiller l'activité de FDJ. Néanmoins, cela impacte grandement l'architecture logicielle des applications et l'architecture du système informatique.

Les moteurs de jeux doivent, en plus d'implémenter le protocole du coffre, produire des événements au format imposé par l'ANJ. Cela affecte également les applications qui, en plus de stocker les transactions, doivent maintenant écrire dans le coffre-fort et assurer l'intégrité et la cohérence avec leur copie.

L'architecture du système informatique est pareillement influencée par cette contrainte, car le coffre ainsi que les transactions nécessitent une infrastructure particulière. En effet, pour pouvoir traiter et stocker une quantité importante d'événements, environ 5 milliards par an, le coffre-fort ainsi que

l’infrastructure l'hébergeant doivent être résilients et sécurisés. Une panne réseau, une coupure de courant, le crash d’une application ne doivent pas impacter l’intégrité du coffre-fort.

Pour répondre à cette seule exigence et ne pas se faire sanctionner lors d’audits, il est nécessaire d’avoir des experts réseau, cybersécurité, architecte solution, conception et développement au pôle technologie et, plus particulièrement, à la direction technique.

Au-delà des contraintes imposées par les organismes de régulation de jeux d’argent et de hasard, l’entreprise a intérêt à investir dans des systèmes (applicatifs ou d’infrastructure) observables et auditables pour pouvoir détecter une anomalie au plus tôt. Il en va de sa réputation, une prise de jeux EuroMillions gagnante égarée pourrait fortement nuire à l’image publique de la Française Des Jeux.

Maintenant que le contexte FDJ est posé, voyons ce qu’est le Tech-Lab et comment il s’insère dans l’organisation de l’entreprise.

1.2 - Le Tech-Lab

Ce chapitre a pour objectif de présenter l'équipe qui m'a accueilli lors de mes trois années d'alternance. Nous verrons dans un premier temps la raison d'être du Tech-Lab. Il conviendra ensuite de présenter sa gouvernance ainsi que son fonctionnement au sein de la direction technique. Enfin, nous détaillerons les types de missions traitées, les membres de l'équipe et l'environnement dans lequel ils travaillent.

1.2.1 - Raison d'être

La raison d'être du Tech-Lab est de qualifier des solutions technologiques émergentes et de les industrialiser pour faciliter leur utilisation.

Il y a premièrement, un besoin de qualifier des solutions, c'est-à-dire vérifier et prouver qu'une solution fonctionne et répond aux besoins. Externaliser la qualification de solution dans un service à part, le Tech-Lab, permet :

- d'avoir et d'investir du temps dédié à la qualification de la solution;
- de standardiser une solution à un besoin.

Enfin, il y a un besoin d'industrialiser une solution pour la mettre à niveau, pour qu'elle réponde pleinement au besoin. Comme nous l'avons vu lors de la présentation de l'entreprise, la DT évolue dans un contexte fortement contraint et il est rare qu'une solution soit prête à être déployée. Le Tech-Lab industrialise une solution en :

- mettant en place des chaînes automatisées de test fonctionnel et d'exigences non fonctionnelles;
- la configurant pour être sécurisée par défaut;
- la documentant;
- la livrant dans un format prête à être déployée.

L'objectif étant de pérenniser et faciliter l'utilisation d'une solution ainsi que son déploiement.

Pour résumer, afin d'éviter que chaque entité étudie et mette en place des solutions différentes, le Tech-Lab est mandaté pour réaliser cette mission prenant en compte l'ensemble des besoins. Ceci permet de consommer moins de ressources et d'harmoniser les composants du SI. Les ressources du Tech-Lab étant dédiées aux missions, il est possible d'appliquer une méthodologie éprouvée prenant en compte l'ensemble des contraintes du Groupe FDJ afin de faciliter le futur maintien en condition opérationnelle de la solution choisie.

1.2.2 - Gouvernance et types de missions

Sans connaissance de l'ensemble des besoins et contraintes du SI, le Tech-Lab dépend d'un conseil d'administration ou board en anglais.

Le conseil d'administration, composé de responsables du pôle technologie de l'entreprise et ses filiales, constitue la gouvernance du Tech-Lab. Tous les deux mois et demi, le conseil ainsi que le responsable du Tech-Lab se réunissent pour discuter de nouveaux besoins identifiés ainsi que l'avancement des missions. Les sujets identifiés sont priorisés si nécessaire.

Lorsqu'un nouveau besoin est identifié, il est ajouté à la liste des missions potentielles. Cette dernière recense l'ensemble des sujets à traiter ainsi que leur priorité. Dès que le Tech-Lab finit une mission, il passe au sujet prioritaire suivant.

Il existe différents types de missions, car chaque sujet ne nécessite pas le même niveau d'approfondissement :

- recommandations d'une solution;
- qualification d'une solution;
- qualification et industrialisation d'une solution.

Pour chaque type de mission, une méthodologie de projet similaire est appliquée.

Lorsqu'un client a besoin d'une recommandation, nous analysons en détail son besoin ainsi que les différentes solutions existantes. L'objectif est d'étudier, évaluer et documenter les différentes options. À partir de nos résultats **purement théoriques**, nous faisons une recommandation au client qui est libre de la suivre ou non.

Les missions de qualification vont encore plus loin. Au-delà d'une recommandation, nous qualifions les fonctions et performances des solutions. L'objectif est de documenter et **prouver** qu'une ou plusieurs solutions répondent ou pas aux besoins du client.

Enfin, lorsque le client a besoin d'une solution prête à être utilisée, nous l'industrialisons. Après la qualification d'une solution, il se peut que celle-ci ne soit pas au niveau des attentes fonctionnelles, performances et opérationnelles du client. Parfois, en plus d'industrialiser la solution, une mise à niveau est nécessaire.

1.2.3 - L'équipe

Afin de mener à bien ses missions, le Tech-Lab dispose de divers types de compétences.

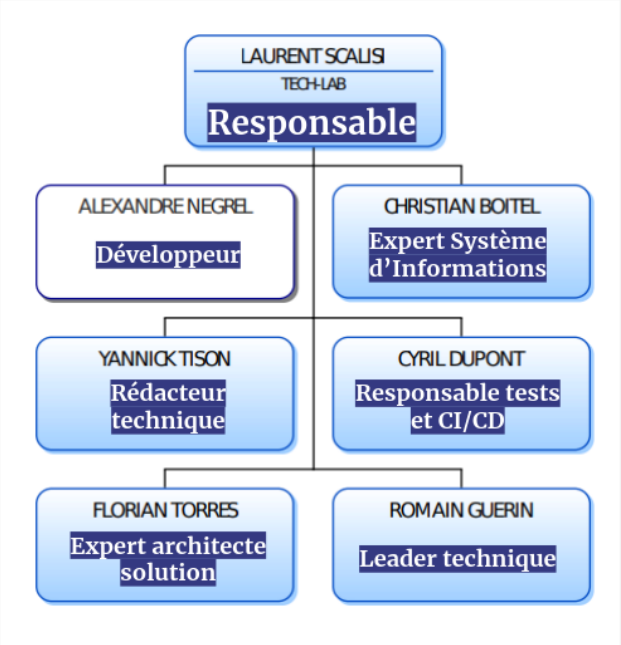


Figure 5 : Organigramme du Tech-Lab août 2023.

Le responsable du Tech-Lab, Laurent Scalisi, est le garant de la mise en place et du pilotage d'une équipe pluridisciplinaire permettant de qualifier, industrialiser des solutions technologiques innovantes identifiées dans le cadre de l'évolution de l'architecture du SI et des produits commercialisés par le groupe FDJ.

Christian Boitel, expert SI, veille à élaborer et proposer des solutions les plus efficaces possibles, notamment en termes d'architecture technique, de technologies, de conformité, de risques et de sécurité des SI. Il incarne la technologie et dans ce cadre, il assure un rôle de conseil, d'assistance, d'alerte, d'information et de formation sur tout ou partie des projets de son périmètre. Il effectue une veille technologique, il participe aux études techniques générales et à son évolution ainsi qu'à la qualification des plates-formes informatiques. Enfin, il définit la vision moyenne et long terme dans son domaine de compétence.

Florian Torres, architecte solution confirmé, définit des solutions d'architecture et garantit, pour le périmètre de son entité, la conformité des choix avec la stratégie technologique de la FDJ et avec les exigences et contraintes des projets en appliquant les standards d'architecture et d'urbanisme.

Romain Guerin, leader technique, pilote la conception applicative et le développement de l'ensemble des projets de son périmètre. Dans ce cadre, il définit les méthodes et outils de travail, il pilote les développeurs et est garant de la qualité des livrables et de la performance des applications ou des services en sortie de développement. Il est le référent technique et le point d'entrée principal de son périmètre pour tous les enjeux de performance, de la qualité des applications et du maintien en condition opérationnelle.

Cyril Dupont, responsable tests et CI/CD (Continuous Integration / Continuous Deployment), met en place les conditions permettant d'industrialiser et d'automatiser les livrables dans le respect de la qualité, du coût et des délais.

Yannick Tison, rédacteur technique, met en place la stratégie documentaire de communication de l'entité afin de vulgariser les nouvelles technologies pour l'ensemble du groupe au travers de l'organisation d'événements ludiques et innovants. Il est également garant de la qualité des guides et documentations associés aux livrables de l'entité qui doivent être au niveau d'un éditeur de logiciels.

En ce qui me concerne, j'intègre l'équipe en tant que développeur. Je contribue aux projets et aux outils via le développement de fonctionnalités, tests fonctionnels et de performances. Mes contributions ainsi que mon rôle dans l'équipe sont approfondies dans la section 4.

En plus des membres permanents, des contributeurs internes ou externes peuvent être ajoutés à l'équipe selon les besoins de la mission en cours ou de l'évolution de notre infrastructure dédiée.

Voyons maintenant les ressources matérielles dont dispose le Tech-Lab.

1.2.4 - Environnement de travail

Les contraintes fortes de l'entreprise ne sont pas compatibles avec les missions qui nécessitent de tester et de déployer des composants non approuvés par les équipes de sécurité. C'est pourquoi le Tech-Lab dispose de sa propre infrastructure informatique isolée du reste du SI.

Le Tech-Lab est entièrement autonome et ne dépend quasiment pas du SI FDJ. Les équipements du Tech-Lab (serveur, poste de travail) n'ont pas d'accès aux services et réseaux internes de l'entreprise (suite Microsoft, GitLab, etc). C'est pourquoi, l'équipe maintient ses propres services pour ses propres besoins.

De manière générale, le Tech-Lab utilise les mêmes outils qu'en interne, mais déployés sur son infrastructure. À date, GitLab et Sonatype Nexus Repository Manager sont respectivement utilisés comme plateforme de développement et CI/CD (intégration et déploiement continue) et pour gérer les fichiers binaires, artefacts sur l'ensemble de la chaîne d'approvisionnement logicielle. Pour la bureautique, l'équipe utilise les versions gratuites de la suite Microsoft avec Outlook et Teams.

Pour ce qui est du poste de travail, des portables HP avec Windows et Linux installés sont utilisés. Pour être précis, c'est la distribution Linux Ubuntu, dans sa dernière version, qui est utilisée. Le groupe FDJ a récemment fait le choix d'Ubuntu pour ses serveurs et, naturellement, le Tech-Lab a fait de même pour les siens et les postes de travail.

Au Tech-Lab, les membres de l'équipe sont responsabilisés. Chaque membre est administrateur de sa propre station de travail. Il en est responsable et s'occupe du maintien en condition opérationnelle de sa machine. Cela inclut les mises à jour logicielles, le durcissement du système d'exploitation, etc. Néanmoins, il est évident que des règles doivent être respectées, seulement, elles ne sont pas garanties par des sécurités informatiques, mais par les utilisateurs eux-mêmes. L'ensemble des bonnes pratiques et règles à suivre sont regroupés dans une charte, validée par le groupe cybersécurité, sur le GitLab du Tech-Lab.

Dans le cadre des missions, il est légitime que des composants et des outils soient installés. Il est donc naturel que chacun ait les droits d'administrateur sur sa machine.

La présentation de l'équipe étant faite, voyons concrètement ce qui est fait au Tech-Lab.

2 - La mission solution d'API Gateway

Au cours de mes trois années d'alternance, le Tech-Lab a travaillé sur les missions suivantes :

- qualification et industrialisation d'une solution API Gateway;
- qualification d'un produit Identity Provider;
- recommandation d'un produit d'APM (Application and Performance Management).

Ces trois missions, toutes riches en apprentissage, concernent des solutions technologiques essentielles pour assurer le bon fonctionnement et la performance des infrastructures informatiques. J'ai eu l'opportunité d'approfondir mes connaissances en méthodologie de projet, ingénierie logicielle, protocoles web et systèmes basés sur UNIX lors de chaque mission.

Le choix de concentrer mon récit sur la mission solution d'API Gateway, la seule terminée à ce jour, a été fait afin de présenter la méthodologie de projet du Tech-Lab dans son ensemble. Ce chapitre est une introduction à la mission qui explicite le besoin initial, le positionnement technique d'une API Gateway ainsi que ses rôles et les contraintes associées.

La méthodologie de projet et mes contributions lors de cette mission sont détaillées dans leur chapitre respectif.

2.1 - Le besoin

Dans le cadre de l'évolution du SI de la FDJ et de la mise en place de micro-services, un besoin grandissant d'exposition de services publics a été identifié.

Ne possédant pas de solution robuste d'API Gateway au sein du groupe, une mission autour de ce besoin a été confiée au Tech-Lab. Cette mission ne se limitait pas à une recommandation mais bien à la livraison d'une solution industrielle, intégrant toutes les fonctionnalités utiles au groupe FDJ.

Premièrement, il est nécessaire de définir l'acronyme API qui signifie littéralement une interface de programmation d'une application. De la même manière, que nous, humains, interagissons quotidiennement via des interfaces, les applications font de même. Un humain interagit avec sa voiture via les pédales, le levier de vitesse et le volant. Une application interagit avec d'autres applications via des interfaces programmables. C'est-à-dire sans l'intervention d'un humain. Une API expose un service, et ce, peu importe la manière (requête IPC, HTTP, etc).

Les API sont accompagnées d'une spécification, un contrat d'interface, documentant l'usage de cette dernière. Par exemple, pour démarrer une voiture, il faut la clé de cette dernière, pour tourner à droite, il faut tourner le volant dans le sens horaire. Plusieurs voitures d'un même modèle peuvent implémenter le même contrat d'interface.

Si l'on poursuit la métaphore, les voitures d'un même modèle sont des instances de services. Différents modèles n'implémentent pas le même contrat ni les mêmes fonctionnalités. Les véhicules utilitaires et les formules 1 n'offrent pas le même service.

Une API Gateway a pour rôle de contrôler et réguler le trafic d'accès dit « Nord-Sud » mettant en relation les consommateurs externes avec les services du SI, qu'il s'agisse des équipements en points de vente, du joueur en mobilité ou dans un point de vente.

Comment cela fonctionne dans les faits et quel est son rôle ?

Afin de bien comprendre le positionnement de l'API Gateway, nous utiliserons ici la métaphore de la maison pour tenter de positionner son rôle d'un point de vue général.

Une maison est généralement constituée d'une porte d'entrée ouvrant généralement sur un couloir d'accès à des pièces, chacune disposant d'une porte d'accès. Chaque pièce dispose d'une utilité bien précise (difficile d'envisager de mélanger les toilettes et la cuisine) et les pièces disposent habituellement de porte d'accès adaptée à la fonction dévouée à cette pièce.

Si l'on projette ces concepts à un système informatique au lieu d'une maison, nous allons donc retrouver les similitudes suivantes :

Dans une maison	Dans le SI
Porte d'entrée de la maison	API Gateway
Une pièce	Un service
Porte d'accès à une pièce	Micro Gateway
Couloir reliant les pièces	Service Mesh

Tableau 1 : Traduction lexicale de la métaphore de la maison.

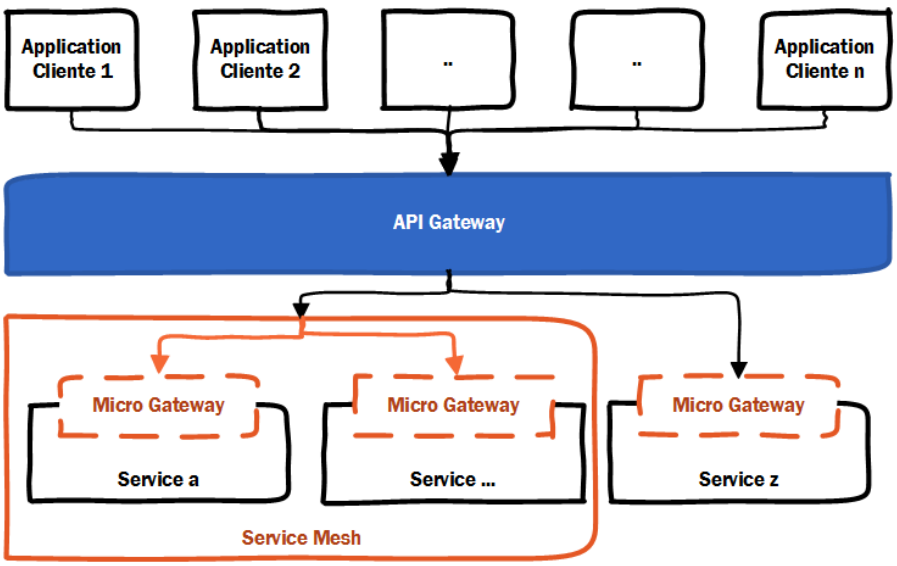


Figure 6 : Positionnement d'une API Gateway dans un SI.

2.2 - Positionnement technique

La Gateway d'API est positionnée derrière une infrastructure technique chargée de l'exposition technique du point d'accès. Cette infrastructure est chargée de sécuriser et optimiser les accès techniques depuis le réseau jusqu'au protocole HTTP/S : routeurs, pares-feux et ADC sont utilisés à cet effet.

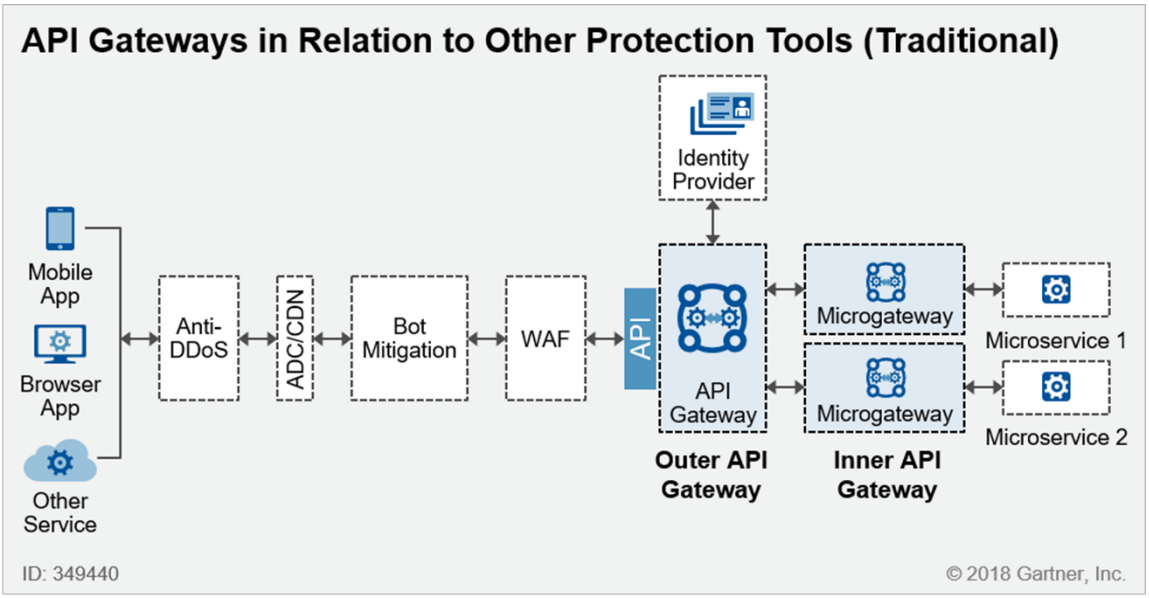


Figure 7: Positionnement technique d'une API Gateway dans un SI.

L'objectif n'est donc pas de répliquer les fonctionnalités déjà couvertes par des composants disposés en amont de l'API Gateway mais de plutôt s'intégrer dans ce dispositif et de se concentrer sur la mise à disposition des API.

2.3 - Rôles d'une Gateway

Une API est décrite dans un contrat d'interface sous la forme d'une représentation formelle : FDJ a fait le choix logique d'OpenAPI pour cette description.

Conformément à ce qui a été défini dans ces contrats d'interface, l'API Gateway est responsable de la mise en œuvre du point d'accès externe à ces API. Ainsi, il lui appartient de :

- gérer les points d'accès définis dans les contrats d'interface;
- contrôler que les appels sont conformes aux contrats d'interface et aux règles opérationnelles de déploiement des API;
- localiser les services chargés d'implémenter les API et router l'appel vers ces derniers.

Il est parfois tout aussi important de définir également ce qui n'est pas du ressort d'une API Gateway.

L'API Gateway n'est pas une solution d'API Management, elle n'en est qu'un élément. Dans ce cadre, elle utilise un référentiel des API, mais n'est pas elle-même le référentiel. De la même manière, elle n'a pas pour rôle d'exposer la documentation des API (portail d'API), ou encore de gérer le cycle de vie de l'API, des consommateurs et de leurs permissions (Identity Provider, IdP).

Elle ne connaît pas l'organisation interne des services. La Gateway est une cliente parmi tant d'autres du service : elle n'implémente donc pas une logique de routage entre les composants internes d'un service. Autrement dit, pour un service donné, la Gateway route vers une unique destination sur laquelle sera exposée l'intégralité de l'API. L'utilisation d'une Micro Gateway ou encore d'un annuaire de services commun peuvent être utilisés pour répondre à cette problématique.

Elle n'est pas responsable de l'optimisation des performances ou de la disponibilité d'un service. Les moyens d'assurer le niveau de performance et de disponibilité attendus par un service est de la responsabilité de ce dernier. L'utilisation de caches ou de multiples instances d'un même composant doit rester transparente du point de vue des clients du service, donc de la Gateway. À noter que l'utilisation de caches de réponses publiques en vue d'une exposition externe est déjà couverte dans la couche technique.

Enfin, elle ne connaît pas la nature des traitements à réaliser. Elle n'effectue de ce fait aucun traitement relatif à un appel d'API en dehors de la mise en relation du consommateur avec un service chargé de réaliser la tâche demandée. Elle n'a donc nullement la charge d'adapter (modifier, enrichir) les appels ou réponses.

Maintenant que le besoin est identifié, nous allons expliciter les contraintes associées.

2.4 - Les contraintes

Dans le cadre de la mission, plusieurs types de contraintes ont été identifiés.

Premièrement, il y a des contraintes inhérentes aux activités de la FDJ comme mentionnées lors de la présentation de l'entreprise :

- sécurité;
- performance;
- intégrité;
- traçabilité et observabilité.

Des contraintes liées à la future utilisation de ce composant, son intégration dans des solutions proposées à nos entreprises clientes, ont milité pour l'utilisation de composants de type Open Source.

L'Open Source Initiative énonce les critères suivants pour définir l'Open Source : redistribution libre, accès au code source, autorisation de modifications et de distributions dérivées (à usage commercial inclus), absence de discriminations et de restrictions. Elle promeut l'accessibilité, la collaboration et l'innovation dans la communauté du logiciel.

Le besoin et ses contraintes étant explicités, nous pouvons à présent aborder la méthodologie conçue pour y répondre.

3 - Méthodologie de la mission API Gateway

Ce chapitre traite de la méthodologie créée lors de la première mission du Tech-Lab. Cette dernière a été mise en place pour répondre à la problématique suivante :

*Comment **étudier** et **industrialiser** une solution d'API Gateway Open Source **répondant au besoin fonctionnel**, de **performance** et **sécurité** du groupe FDJ tout en étant **efficace** et en garantissant l'**objectivité**, la **traçabilité** et la **reproductibilité** des choix ?*

Au-delà du besoin premier, celui du demandeur, et des contraintes (technique, performance et sécurité) présentés dans le chapitre précédent, des contraintes organisationnelles découlent de la position du Tech-Lab au sein de la direction technique.

Le Tech-Lab met en place des solutions pour d'autres services. Il se doit de répondre au besoin du demandeur, c'est pourquoi, il est impératif que les décisions soient **objectives**, **traçables** et **documentées**. Le demandeur doit pouvoir suivre la mission et arriver aux mêmes conclusions afin qu'elles ne puissent pas être remises en cause.

Enfin, comme pour toute méthodologie, l'objectif est d'être le plus **efficace** : le besoin n'attend pas et les ressources du Tech-Lab sont limitées. La philosophie derrière la méthodologie est de partir d'une liste de solutions, les évaluer jusqu'à déterminer la meilleure solution, et ce, en étant le plus efficace possible.

Pour remplir ces objectifs, la macro planification comporte les étapes suivantes :

- Expression du besoin;
- Recensement des produits répondant potentiellement au besoin;
- Document des principes d'évaluation;
- Qualification des produits;
    - Évaluation théorique;
    - Évaluation pratique;
- Industrialisation et mise à niveau de la solution sélectionnée;
- Livraison et support.

Les motifs sous-jacents à la présence de chaque étape sont exposés de manière détaillée dans les chapitres suivants.

3.1 - Démarrage de la mission

Une fois le sujet sélectionné, le Tech-Lab initialise la mission via diverses tâches réalisées en parallèle.

Premièrement, il est nécessaire de déterminer les acteurs et contributeurs du projet qui participeront à l'expression du besoin. Cela inclut les différents demandeurs, mais aussi toute entité ayant des contraintes à ajouter comme la sécurité par exemple.

Une fois le recensement réalisé, une réunion de lancement de la mission a lieu afin de réunir l'ensemble des acteurs identifiés précédemment. La méthodologie et la macro planification de la mission y sont alors présentées. De même, des réunions pour suivre l'avancement du projet sont programmées.

La méthodologie du Tech-Lab pourrait être qualifiée de "semi-agile" ou de "méthodologie hybride". Elle combine des éléments de méthodologie traditionnelle avec des pratiques agiles.

La première phase du projet est davantage orientée vers une planification approfondie, une analyse des besoins et une mise en place des bases du projet. Une fois cette phase initiale terminée, la méthodologie passe à une approche agile avec des itérations de développement (sprints) comprenant des démonstrations (démo) et des rétrospectives pour favoriser l'adaptation continue et l'amélioration de la solution, de la méthodologie ou autre.

Cette méthodologie hybride permet de combiner les avantages des approches traditionnelles et agiles, adaptant ainsi le processus en fonction des spécificités du projet et des disponibilités du demandeur externe. De plus, elle facilite le chevauchement de deux missions à deux stades différents.

En parallèle des premières réunions, un membre du Tech-Lab est sélectionné pour être le Product Owner (PO) de la mission. Ce dernier est le responsable du produit au sein d'une équipe de développement agile. C'est lui qui définit les exigences, priorise les tâches et prend les décisions concernant le développement du produit. Il est le garant de la vision du produit et assure la communication avec les parties prenantes.

Dès lors qu'il est désigné, le Product Owner commence à se documenter et s'informer sur le sujet de la mission. Les sujets traités par le Tech-Lab étant assez vastes, une mise à niveau est souvent nécessaire. Une fois ces premières étapes réalisées, l'expression du besoin peut commencer.

3.2 - Expression du besoin

Les demandeurs ne pouvant pas être présents quotidiennement aux côtés du Tech-Lab pendant la mission, l'expression du besoin doit être faite en amont.

Cette phase qui s'étend sur environ deux mois est principalement dédiée à la rédaction du cahier des charges (CDC). Traditionnellement, le cahier des charges est un document formel et détaillé qui décrit précisément les spécifications, les exigences, les fonctionnalités, les contraintes techniques, les objectifs et les attentes liées au projet ou à la solution. Il sert de référence commune entre les parties prenantes impliquées dans le projet, facilitant la définition du périmètre du travail à réaliser et des résultats attendus. Le cahier des charges est essentiel pour assurer une compréhension commune des besoins, faciliter la planification et la réalisation du projet, et veiller à la conformité aux spécifications établies.

Pour remplir efficacement son rôle, le cahier des charges doit être :

- correct : une erreur ou une omission peut détourner le projet de ses objectifs;
- formel : une rédaction claire et précise évite les interprétations divergentes et les malentendus;
- accessible : il doit être compréhensible pour tous les acteurs et contributeurs, quel que soit leur domaine;
- réalisable : les besoins et les contraintes définis doivent être réalisables avec les ressources disponibles, qu'elles soient humaines ou matérielles.

Un cahier des charges ayant ces caractéristiques contribue fortement à l'efficacité et à la réussite du projet. À présent, intéressons-nous à la rédaction pratique du document.

Après la réunion de lancement de la phase d'initialisation, d'autres réunions sont programmées. Les premières sont destinées à l'expression du besoin. Les acteurs et contributeurs de la mission, le responsable du Tech-Lab et le Product Owner se réunissent pour discuter et définir le besoin.

Le Tech-Lab ayant aussi un rôle de conseil, les besoins sont challengés afin de ne pas dénaturer les fonctions intrinsèques d'une API Gateway.

La rédaction est un processus itératif, le document est amélioré et corrigé au fil des réunions. Tant que l'ensemble des acteurs et contributeurs ne valident pas le document, une autre réunion est programmée.

Le rédacteur du cahier des charges n'est personne d'autre que le Product Owner de la mission. La rédaction étant une activité assez chronophage, les parties prenantes n'ont pas forcément le temps ni les compétences nécessaires. Ainsi, il a été choisi de confier l'écriture à un membre de l'équipe. Étant responsable de la qualité fonctionnelle de la solution, le PO est le plus à même d'en rédiger la spécification.

Le besoin de la mission API Gateway présenté précédemment est un résumé de l'expression du besoin.

En parallèle de l'expression du besoin, le Tech-Lab commence à recenser les produits sur le marché.

3.3 - Recensement des produits

Le recensement des produits est la création d'une liste des produits existants sur le marché entrant potentiellement dans le périmètre de la mission. Le besoin n'étant pas encore défini dans les détails, les produits sont ajoutés sans réels critères.

Au fur et à mesure que l'expression du besoin s'affine, des critères sont ajoutés ou précisés. Certains critères sont implicites et non demandés par le client. À chaque réunion avec les parties prenantes, la liste et ses critères sont partagés. Par ailleurs, elles aussi peuvent proposer des produits.

L'ensemble de l'équipe, y compris moi, participe à la recherche de produits. Pour la mission API Gateway, une liste de 18 produits fût construite. L'objectif des prochaines phases du projet sera de réduire la taille de cette liste jusqu'à sélectionner un produit final.

	Company	Product	Environnment	Licence O.S		Documentation/fonctions	Vie	mots clefs	Fonctionnalités	Célébri
1	Kong	The Cloud-Native API Gateway		apache-2.0		https://github.com/Kong/kong	Open Issues : 339 Commit : 3 days	cloud-native	SSL: Setup a Specific SSL annotations	25036
2	Ambassador	gateway for microservices built on the	Kubernetes	apache-2.0		https://github.com/datawire/ambassador	Open Issues : 109 Commit : 3 days	microservice cloud-native	Support for gRPC	2580
3	Krakend	Ultra performant API Gateway with middlewares	cloud, k8s, bare metal, ...	apache-2.0		https://github.com/devopsfaith/krakend	Open Issues : 21 Commit : 2 days	middleware router	failure : Stateless. No databases. - Better user	2417
4	Gloo	Gloo is a feature-rich, Kubernetes-native ingress controller, and next-generation API gateway.	K8S	apache-2.0		https://github.com/solo-io/gloo	Open Issues : 280 Commit : a day	microservices grpc cloud-native api-gateway	projects, including gRPC, GraphQL, OpenTracing, NATS	2140
5	Express Gateway	A microservices API Gateway built on top of Express.js	npm docker	apache-2.0		https://www.lunchbadger.com/api-gateway	Open Issues : 38 Commit : 8 days	expressjs api-gateway	Centralized Config API Consumer and Credentials	2015
6	Incubator Apisix	Cloud-Native Microservices API Gateway		apache-2.0		https://github.com/apache/incubator-apisix	Open Issues : 143 Commit : a day	cloud-native api-gateway	operators for	1986
7	Gravitee Gateway	Gravitee.io - API Management - OpenSource API Gateway		apache-2.0		https://github.com/gravitee-io/gravitee-gateway	Open Issues : 10 Commit : 3 days	api-gateway	MongoDB, Redis, Elasticsearch, ...	943
8	Fusio	Open source API management platform		agpl-3.0		https://github.com/apioo/fusio	Open Issues : 62 Commit : 8 days	rest	Subscription :	815
9	Reactive Interaction Gateway	Create low-latency, interactive user experiences for stateless microservices.	K8S	apache-2.0		https://github.com/Accenture/reactive-interaction-gateway	Open Issues : 50 Commit : 6 days	reactive api-gateway server-sent-event	implementation (browser) keeps track of connection	393
10	Krakend CE	KrakenD Community Edition. Make your binary of KrakenD API Gateway	linux, docker	apache-2.0		https://github.com/devopsfaith/krakend-ce	Open Issues : 14 Commit : 3 days	microservice cloud-native	Bursting on	159
11	Product Microgateway	The **WSO2 API Microgateway** is a Cloud		apache-2.0	do ck	https://github.com/wso2/product-microgateway	Open Issues : 81 Commit : 3 days	microservices cloud-native	Analytics : Publish data to streams	157
12	Tree Gateway	This is a full featured and free API Gateway	docker	mit		https://github.com/TreeGateway/tree-gateway	Open Issues : 17 Commit : 2 months	microservice opensource free	circuitbreaker open circuit or an authentication	148
13	Apigility by Zend Framework	open-source framework to consider for API management. Apigility is an API Builder, designed to				https://apigility.org/documentation		PHP 5.6	versioning; Normalisation and Validation; Authentication	

Figure 8 : 13 des 18 produits recensés et leurs critères.

À la fin de l'expression du besoin, il est déjà possible de réduire la taille de la liste sur des critères rédhibitoires. Par exemple, tous les produits non Open Source tels que définis par l'Open Source Initiative sont supprimés.

Après sélection, les 10 produits suivants ont été retenus:

- Amazon API Gateway
- Apigee Cloud API Gateway
- Envoy
- Gloo
- Gravitee API Gateway
- Kong
- Krakend Community Edition
- Traefik
- Spring Cloud Gateway
- WSO2 API Gateway

Comme pour le reste de la méthodologie, la liste avant et après sélection sur critères rédhibitoires est documentée et partagée lors des réunions avec les acteurs et contributeurs.

Une fois les produits recensés et l'expression du besoin terminée, l'équipe a une bonne idée des compétences requises et de l'ampleur de la mission. Le responsable du Tech-Lab, Laurent Scalisi, commence donc à recruter des ressources externes si nécessaire.

3.4 - Document des principes d’évaluation

Après avoir clairement exprimé le besoin et construit une liste de produits, il faut déterminer lequel répond le mieux au besoin. Pour ce faire, il faut classer les produits et donc, les évaluer et les noter. Le Tech-Lab étant soumis à des contraintes d'objectivité, de traçabilité et de reproductibilité des choix, un document de notation a été mis en place pour adresser ces contraintes.

Premièrement, la traçabilité, un élément clé de la mission. Le document de notation doit s’appuyer sur le besoin du demandeur. On doit y retrouver l’ensemble des spécifications fonctionnelles, de performance et de sécurité du CDC sous formes de critères.

Ensuite, l’objectivité du document et de la notation. La notation des produits étant faite par des évaluateurs différents, il est important d’objectiver les principes de notation de chaque critère afin de ne pas avoir des résultats divergents et subjectifs.

Enfin, la reproductibilité des choix. Celle-ci n’est pas complètement garantie, car la phase d’évaluation décrite dans le chapitre suivant est purement théorique et basée sur des sources externes telles que la documentation et les sites web des produits. Ces derniers ne sont pas archivés. Néanmoins, le maximum est fait en documentant les sources, leur date de consultation et la version des produits.

Le document des principes d’évaluation est, lui aussi, sous la responsabilité du Product Owner mais est rédigé avec l’aide du responsable du Tech-Lab et du rédacteur technique. Enfin, comme pour le cahier des charges, ce document est validé par les parties prenantes.

Examinons maintenant le document de notation de la mission API Gateway. Ce dernier est découpé en deux parties. La première se concentre sur la méthodologie globale et les notions tandis que la seconde est dédiée à la notation.

L’objectif de la première partie est de permettre au lecteur d’utiliser le document. On y retrouve le principe général de la phase d’évaluation, la méthodologie de notation et son application. L’exploitation des résultats y est également décrite.

La seconde partie liste les axes de notation et leurs critères. On y retrouve, pour chaque critère, un paragraphe sur son objectif et la méthode d’évaluation. Le premier lie le critère au besoin tandis que le second décrit formellement comment noter ce critère. Certains critères ont une note seuil et entraînent une élimination si cette dernière n’est pas atteinte. Quelques critères de la mission sont en annexe 10.2.

Une grille de notation, un tableur avec les critères et leur pondération, est issue du document de notation. Abordons maintenant l’usage pratique de ces documents.

3.5 - Évaluation théorique

La phase d'évaluation théorique est la première où tous les membres de l'équipe sont impliqués et applique la méthodologie du document de notation.

Premièrement, les évaluations se font avec un binôme d'évaluateurs et un porteur par produit. Le porteur d'un produit ne l'évalue pas pour objectiver l'évaluation. Les évaluateurs peuvent tout de même consulter le porteur en cas de besoin. Chaque évaluateur effectue la notation seul, puis confronte son travail avec son binôme afin de définir une évaluation commune, présentée et validée à son tour par le porteur. Ces principes seront appliqués quelle que soit l'issue de la notation. En cas d'élimination, le porteur du produit aura validé la notation.

La notation doit être réalisée en utilisant le matériel mis à disposition sur le produit, que ce soit sur le site de l'éditeur, le dépôt de source ou encore des sites relatant des expériences. Le but n'est pas d'installer une plateforme et de réaliser des tests, mais bien d'évaluer le produit sur son déclaratif. En l'absence d'information disponible pour noter un critère et sans autre indication dans la méthode d'évaluation, la note retenue sera de zéro.

Les notes de chaque évaluateur et celle retenue sont justifiées dans un document et dans la grille de notation. Les évaluateurs de produits différents discutent aussi ensemble lors de la saisie des notes. Une fois toutes les notes saisies, la grille de notation calcule la note finale de chaque produit en prenant en compte la pondération des critères et des axes. Le classement des produits sera alors simplement établi, les notes les plus élevées représentent les produits les plus à même de réaliser la mission. En cas d'égalité stricte entre plusieurs produits, l'équipe attribuera des points bonus.

Les grands principes de notation étant établis, intéressons-nous aux axes de notations.

Pour la mission API Gateway, les axes suivants ont été retenus :

- Gateway Provider
- Operations ready by design
- Security by design
- Resilient by design
- Roadmap, support et licenses

Le premier axe, Gateway Provider, est spécifique à la mission. Il regroupe l'ensemble des critères nécessaires pour répondre au besoin fonctionnel. Voici une liste non exhaustive des critères de cet axe d'analyse :

- Support d'OpenAPI
- Extensibilité (développement d'extension)
- Autorisation d'accès
- Support des IT Providers
- Performance brute

Les axes restants sont présents à chaque mission et découlent du contexte FDJ. Dans l'axe "Operations ready by design", on retrouve des critères tels que le déploiement à chaud, la présence de journaux d'accès et d'erreur, la métrologie technique, etc.

L’axe “security by design” évalue la sécurité du produit avec des critères tels que la surface d’exposition minimale, la protection des accès d’administration, la confidentialité de la persistance des données sensibles, etc.

L’axe “resilient by design” évalue la résilience du produit. La résistance sur sollicitation excessive en cas de défaillances et la limitation de l’utilisation des ressources y sont évaluées.

Enfin, l’axe “roadmap, support et licenses” évalue la pérennité du produit, son support communautaire et d’entreprise.

La liste complète des critères de chaque axe et leur pondération est disponible à l’annexe 10.1. Il a fallu un peu moins d’un mois pour évaluer les 33 critères des 10 produits restants dans la liste.

Après avoir entré les résultats dans la grille de notation, le graphique radar suivant ressort.

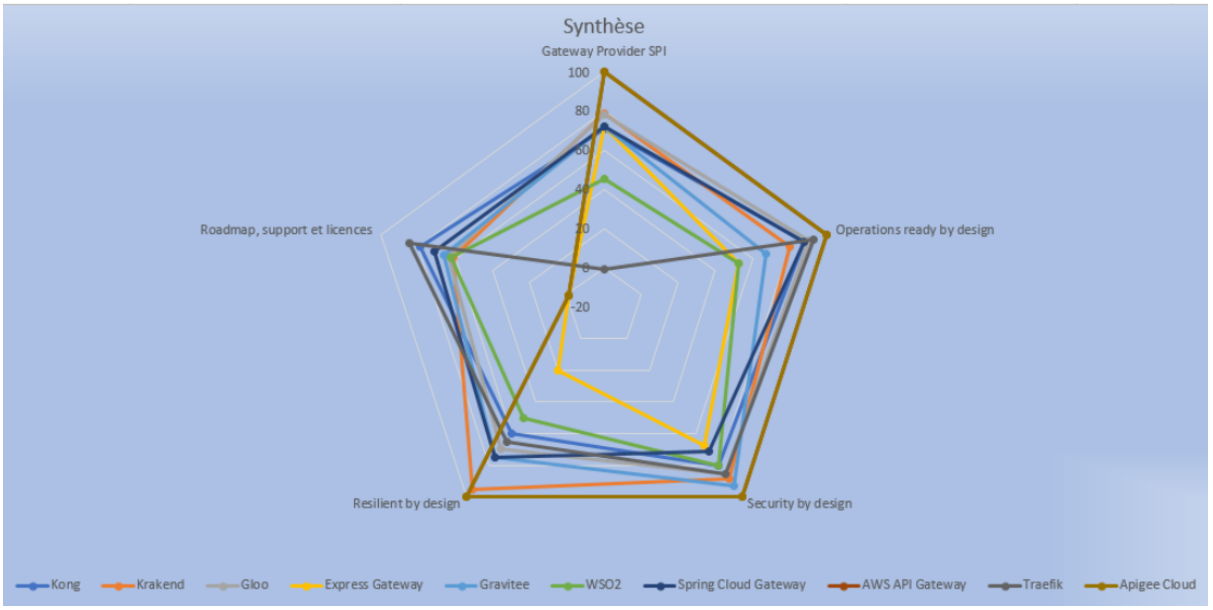


Figure 9 : Graphique radar synthèse des notes des produits par axe.

La grille de notation de chaque produit est disponible en annexe 10.3 - 10.13 . Afin qu’ils soient traçables, les résultats et les documents de justifications sont conservés et présentés aux parties prenantes.

Après la notation, voici le top 3 des produits retenus:

Classement	Note	Produit	Plus	Moins
1	407.0	Envoy	Adaptive concurrency CNCF/AWS/GCP/...	
2	403.8	Krakend	Aggregation (composition) Extensible avec peu de "code"	Petite entreprise de conseil

3	388.0	Gloo	Envoy	Une glue d'Envoy qui filtre les fonctions d'Envoy. Legacy annexe: K8 targeted
4	377.3	Kong	Historique fort	Lua everywhere Coquille vide en mode open source

Gloo est troisième en termes de score mais n’a pas été retenu car basé sur Envoy qui est premier. Ainsi le top 3 est Envoy, Krakend et Kong Community Edition. Nous pouvons maintenant passer à la phase d’évaluation pratique ou “preuve de concept”.

3.6 - Preuve de concept

Après avoir exprimé le besoin, recensé les potentiels produits qui y répondent, évalué puis sélectionné trois d'entre eux, vient le moment de vérifier qu'ils fonctionnent comme attendu. C'est la phase de preuve de concept : le POC (Proof Of Concept en anglais).

D'après Wikipédia :

*Une **preuve de concept** ou **validation de principe**, ou encore **démonstration de faisabilité**, est une réalisation ayant pour vocation de montrer la faisabilité d'un procédé ou d'une innovation.*

Dans le cadre de la mission, l'objectif du POC est surtout de déterminer quel produit répond le mieux au besoin client, et ce, à l'aide d'évaluation **pratique**. C'est la première fois que les produits vont être déployés pour **qualifier** leurs fonctions et performances.

Pour ce faire, un sous-ensemble des critères de l'évaluation théorique est évalué en condition réelle. Par souci d'efficacité, de possibilité et de pertinence, il n'est pas nécessaire d'évaluer tous les critères à ce stade. Les critères à tester sont sélectionnés avec les parties prenantes. C'est ce qu'on appelle une **stratégie de tests**. C'est un compromis entre ce qu'on veut tester et le temps investi. Généralement, les critères avec la plus grande pondération sont choisis et regroupés par phases.

Le POC de la mission API Gateway se découpe en quatre phases. Tout comme l'évaluation théorique, chaque phase a un document expliquant son objectif, les critères évalués et la méthode d'évaluation.

La première phase est l'installation des produits, on évalue et documente la facilité à les configurer. Chaque produit est configuré comme passerelle de 50 services, avec 5 ressources chacun et 6 opérations par ressource. Une opération est un couple unique ressource et méthode HTTP. Les nombres n'ont pas été choisis au hasard, mais sont inspirés des services déployés en production. Pour déployer les produits, un environnement est mis en place à l'aide de Terraform, un outil d'*infrastructure as code*. Ce dernier permet de décrire un environnement complet (machine virtuelle, réseaux virtuels, équilibreur de charge) avec du code.

La seconde phase est le développement d'extension. En effet, comme les produits n'implémentent pas tout le fonctionnel voulu, il faut s'assurer que leurs mécanismes d'extension nous permettent d'implémenter les fonctionnalités manquantes. Pour ce faire, une extension permettant de rejeter et d'altérer une requête est développée pour chaque produit.

Ensuite vient la phase de test dans laquelle on évalue un ensemble de fonctionnalités qui sont déjà implémentées. Une suite de tests commune est alors développée et exécutée sur tous les produits.

La phase finale est l'évaluation des performances des produits. Chaque produit est déployé dans le même environnement, celui de la phase d'installation, avec l'ensemble des fonctionnalités précédemment testé (extensions incluses). Plusieurs types de tests sont faits afin de déterminer les performances pures, la résilience au stress et l'utilisation des ressources de la machine (mémoire et CPU).

Le POC marque un virage organisationnel dans la méthodologie du projet. Désormais, l'équipe évolue en suivant un modèle agile. Ce dernier est basé sur des itérations de développement plus courtes qui s'accompagnent d'une vélocité de développement accrue. Au Tech-Lab, les itérations de développement misent en place ont une taille variable de deux à quatre semaines. À la fin de chaque itération, une réunion de démonstration suivie d'une rétrospective sont programmées. Enfin, le "daily", une réunion quotidienne, à lieux chaque jour à 9 h 30.

Il y a plusieurs raisons à ce changement organisationnel.

L'agilité est l'idéal pour les processus itératif tel que le développement. Chaque phase du POC contient du développement, que ce soit du Terraform, des fichiers de configuration ou bien des tests automatisés. Des courtes itérations de développement permettent de se concentrer sur un sujet, le traiter, le démontrer et apporter des correctifs rapidement.

Ensuite, l'équipe de développement consacre 100 % de son temps au POC. Le développement en agilité favorise l'engagement des membres de l'équipe avec des retours rapides et une collaboration accrue. Les méthodologies plus classiques ont des procédures plus lourdes et séparent davantage les équipes.

Enfin, les méthodes agiles privilégient la valeur et l'efficacité, c'est en parfait accord avec les objectifs du Tech-Lab.

Voyons maintenant comment une itération de développement s'organise.

La première chose à faire est de définir les objectifs de cette itération. Généralement, on veut implémenter des fonctionnalités ou des exigences non fonctionnelles qui sont exprimées sous la forme de "user stories" par le PO :

D'après Atlassian :

Une user story est une explication non formelle, générale d'une fonctionnalité logicielle écrite du point de vue de l'utilisateur final. Son but est d'expliquer comment une fonctionnalité logicielle apportera de la valeur au client.

Le leader technique découpe les "user stories" en tâches techniques. Ensuite, l'équipe de développement estime le temps nécessaire pour réaliser chacune de ces tâches.

Ces dernières sont ensuite ajoutées dans l'itération afin d'occuper tout le temps disponible. Celui-ci est calculé à partir de la formule suivante :
nombre de développeurs × *heure de travail dans une journée* × *nombre de jours dans l'itération* − *heures d'indisponibilité*
Les heures d'indisponibilité correspondent aux congés, arrêts maladie, mais aussi au temps alloué aux réunions, à la démonstration de fin d'itération, etc.

De manière générale, il est préférable d'embarquer les tâches d'une même "user story" afin de la clôturer en fin d'itération.

Une fois les tâches sélectionnées, l'itération peut commencer. Lors de la première réunion quotidienne, le "daily", les tâches sont attribuées dans le but d'occuper les développeurs toute la journée voir plus. Ces derniers travaillent ensuite sur leurs tâches respectives jusqu'à la réunion du lendemain.

Ces réunions quotidiennes ont plusieurs objectifs. Premièrement, c'est l'occasion de partager l'avancement des tâches, les points bloquants ou toutes informations pertinentes. Chaque personne passe à tour de rôle et liste les sujets sur lesquels il a travaillé la veille, le temps passé sur chacun d'eux et les objectifs de la journée. Les "daily" se doivent d'être courts et peu approfondis, si un sujet nécessite du temps, il faut programmer une réunion dédiée.

Dans la journée, si un développeur finit l'une de ces tâches, il créer une "merge request" dans le dépôt de notre GitLab. Une "merge request" est, comme son nom l'indique, une demande de fusion entre les fichiers présents dans le dépôt GitLab et ceux du développeur. Cette dernière est revue par au moins deux personnes de l'équipe, généralement le PO et le leader technique, avant d'être acceptée ou refusée. En cas de refus, des commentaires justifiant la raison ou expliquant les modifications à faire sont ajoutés. Une fois de plus, c'est un processus itératif. Lorsqu'une demande de fusion est acceptée, les changements sont incorporés dans le dépôt distant et la tâche associée sera clôturée au prochain "daily".

Enfin, à la fin de chaque itération, il y a une démonstration. Pendant cette réunion, l'ensemble des tâches clôturées sont démontrées au reste de l'équipe. L'objectif est de montrer ce qui a été fait et comment. Au Tech-Lab, la rétrospective est faite juste après la démonstration. Cette dernière ouvre la discussion sur les points d'amélioration pour les prochaines itérations.

Le virage organisationnel s'accompagne d'un changement de méthode de travail. Désormais, les livrables ne sont plus des documents ou des tableurs, mais bien des fichiers texte (code, configuration, etc). C'est pourquoi l'utilisation de GitLab et Nexus Repository Manager est privilégiée. Nous livrons un ensemble de dépôt de source contenant l'explication des phases du POC, le matériel pour le reproduire et les résultats. Les fichiers sont versionnés à l'aide de l'outil git sur des dépôts GitLab. Enfin Nexus Repository Manager est utilisé pour sauvegarder des artefacts tels que les dépendances externes, nos livrables et les résultats des tests automatisés.

Après de multiples itérations, évaluation et démonstration, le produit Envoy a été sélectionné pour la mise à niveau et l'industrialisation. Dès la phase évaluation théorique, le besoin d'une mise à niveau a été identifié car aucun produit ne vérifie la conformité des requêtes par rapport au contrat d'interface.

3.7 - Industrialisation et mise à niveau

Après avoir choisi le produit le plus adapté aux besoins client, il convient de le mettre à niveau et de l'industrialiser afin de livrer une solution complète de qualité industrielle. Pour ce faire, il est impératif d'étendre les fonctionnalités du produit tout en s'assurant qu'il réponde aux exigences non fonctionnelles du contexte FDJ.

En pratique, l'industrialisation et la mise à niveau se font en même temps, mais par souci de clarté, elles seront abordées successivement.

3.7.1 - Industrialisation

L'industrialisation est un processus de configuration, d'optimisation et d'automatisation visant à garantir un niveau de qualité.

Dans le cadre de la mission, le niveau de qualité est mesuré dans les catégories suivantes :

- fonctionnel;
- sécurité;
- performance;
- résilience;
- maintenance.

La solution étant destinée à être utilisée au sein du groupe FDJ et par des clients externes (loteries étrangères), le Tech-Lab se positionne comme un éditeur de logiciel. De fait, contrairement à l'évaluation pratique, l'ensemble des exigences sont évaluées ici.

Afin de s'assurer du bon fonctionnement de l'ensemble des fonctionnalités, des tests passant et cassant sont développés. Ces derniers sont automatisés dans le dépôt du produit et exécutés à chaque mise à jour du dépôt, c'est de l'intégration continue (Continuous Integration, CI). Il en va de même pour les tests de performance. De cette manière, une mise à jour de la solution, un changement de configuration ou bien du code source ne peuvent introduire une régression fonctionnelle ou de performance.

Pour garantir l'opérationnalité et l'exploitabilité de la solution, de multiples exigences non fonctionnelles sont testées. Premièrement, la présence, la qualité et la justesse des métriques qui permettent d'évaluer, de mesurer et surveiller différents aspects de la solution. Ensuite, les journaux d'événements ou les traces permettant d'observer les actions de la solution et la raison de ces dernières. Enfin, tout autre mécanisme d'auditabilité permettant de suivre ou tracer les actions, les changements dans le système.

La pérennité de la solution dépend grandement de la facilité à la maintenir. L'ensemble de ce qui est produit doit être documenté. Un guide d'administration, un guide d'utilisation et un guide développeur sont rédigés et livrés avec la solution.

Industrialiser un produit n'est pas une tâche facile, cela requiert une infrastructure de projet, choisir les bons outils, mettre en place une plateforme d'injection pour les tests de performance, revoir le code et la documentation. Plusieurs itérations sont nécessaires pour garantir un bon niveau de qualité.

Par exemple, pour pouvoir tester la solution en conditions réelles, il faut parfois développer des serveurs bouchons qui sont eux-mêmes partiellement industrialisés (tests, documentations).

Au Tech-Lab, nous considérons qu'une tâche est finie lorsque la fonctionnalité demandée est implémentée, testée et documentée. Cela allonge le temps nécessaire à la réalisation des tâches mais est indispensable pour maintenir une source unique de vérité. De cette manière, il n'y a pas de version avec une documentation incomplète, chaque version du dépôt est entièrement documentée, testée et fonctionnelle.

Lors de la mission API Gateway, l'industrialisation et notamment l'intégration continue a permis de détecter une régression des performances d'Envoy après une mise à jour. Un ralentissement des connexions TLS a été détecté et remonté aux développeurs qui ont pu corriger le problème.

La suite de test a aussi permis de détecter l'absence ou la non conformité à nos attentes de certaines fonctionnalités. Ces dernières doivent alors être corrigées lors de la mise à niveau.

3.7.2 - Mise à niveau

La mise à niveau consiste à améliorer le produit pour qu'il réponde à toutes les exigences fonctionnelles des demandeurs.

L'ajout ou l'amélioration de fonctionnalités se fait en suivant le processus de développement présenté précédemment : attribution d'une tâche, développement, test fonctionnel et d'exigences non fonctionnelles, documentation dans les différents guides, revues et intégration continue.

Pour la mission API Gateway, le développement de la validation des requêtes conformément à un contrat fût identifié dès l'évaluation théorique. D'autres demandes, non écrites lors de l'expression du besoin, se sont ajoutées au fil de la mission et nécessitent développement et tests.

Comme les solutions fournies par le Tech-Lab se basent sur des solutions Open Source, l'ajout de fonctionnalités dépend des mécanismes d'extension disponibles. Une mise à niveau ne se ressemble pas d'un projet à un autre.

L'évaluation pratique a permis d'identifier les différents mécanismes d'extension de chaque produit ainsi que leurs limitations respectives. Envoy possède trois mécanismes d'extensions. Il faut faire un choix.

Premièrement, il est possible de développer des extensions (ou filtres) en C/C++. Ces dernières ont l'avantage de bénéficier de performances natives, mais nécessitent de recompiler le produit. Envoy ne permet pas de charger dynamiquement des extensions natives.

La seconde option est d'utiliser une extension permettant d'exécuter un script Lua pour chaque requête traitée. Lua étant un langage interprété, les scripts sont chargés dynamiquement et ne nécessitent pas de recompiler le produit.

Enfin, le filtre "External Processing" d'Envoy permet d'externaliser le traitement d'une requête via des appels gRPC. Google Remote Procedure Call ou gRPC est un protocole d'appel de procédure distante basé sur HTTP 2. Un programme implémentant un serveur gRPC supportant les messages

d'Envoy, son protocole, peut agir comme une extension native sur le flux de requêtes traité. Les appels et retours de fonctions sont remplacés par des appels et réponses gRPC.

À la manière de la sélection du produit, le choix du mécanisme d'extension est fait par élimination. La solution choisie doit respecter les contraintes suivantes. Premièrement, la performance ne doit pas être impactée par l'extension. Le développement doit être aisé, facilement maintenable et testable. Enfin, le langage doit être relativement populaire, car des développeurs externes sont recrutés pour compléter l'équipe jusqu'à la fin de la mission.

Le mécanisme d'extension à l'aide de script Lua est le premier à être éliminé car son écosystème est pauvre en développeurs et bibliothèques logicielles. Étant un langage conçu pour être embarqué par un programme hôte, l'environnement d'exécution varie d'un programme à l'autre. Cette diversité d'environnements rend le langage impraticable dans le contexte de la mission.

Il reste deux choix, l'extension native et l'external processor émergent comme les solutions potentielles. L'external processor semble être la meilleure option, car le projet gRPC supporte officiellement plus de dix langages et ne demande pas de recompiler le produit. Néanmoins, la question de la performance se pose.

Don't guess, measure

La célèbre citation d'Éric Ries, qui est aujourd'hui un principe fondamental du développement logiciel, incite à ne pas deviner les performances d'une entité, mais de les tester et mesurer.

C'est ce qui a été fait lors de la phase précédente. En effet, les différents tests de performance de l'évaluation pratique ont montré que l'extension native et le serveur gRPC avaient des performances équivalentes, et ce, dans les différents scénarios.

En raison de ses bonnes performances, son expérience de développement et la richesse de son écosystème, la décision d'implémenter les fonctionnalités via un serveur gRPC écrit Go a été prise.

Le chapitre 4.2 montre concrètement comment une fonctionnalité, la rotation à chaud des fichiers de journalisation, est développée.

À mesure que les itérations progressent, de nouvelles fonctionnalités sont intégrées jusqu'à la finalisation de la solution. Dans les dernières phases, les premières versions sont développées et remises à l'entité réceptrice.

3.8 - Livraisons et support

Une fois la solution terminée ou presque, celle-ci est livrée à l'entité réceptrice. Une fois de plus, le Tech-Lab se positionne comme un éditeur logiciel et doit livrer une solution de qualité industrielle.

La livraison doit inclure une note de publication, un historique des modifications, le code source, les tests ainsi que les différentes documentations. Les dépôts de source sont livrés ainsi que les artéfacts de la version livrée : image OCI/Docker, binaire, etc.

Lorsque le produit est livré, le Tech-Lab a déjà créé de multiples versions de la solution, presque une par itération de développement.

La livraison et le support se font en parallèle. Lorsque l'entité réceptrice reçoit la première version, la quasi-totalité des fonctionnalités demandés sont implémentés. Ce premier lot n'a pas vocation à être déployé en production mais permet à l'entité de se familiariser avec la solution.

Dans la continuité du développement agile, la livraison et le support sont agiles. Des livraisons sont régulièrement faites et accompagnées de support. Ainsi, lorsqu'un problème survient, que ce soit une anomalie ou une quelconque remarque, il est communiqué au Tech-Lab qui le corrige pour le prochain lot.

L'objectif de ces multiples livraisons et, que petit à petit, l'entité réceptrice devienne propriétaire de la solution et ne dépende plus du Tech-Lab.

Pour faciliter le transfert de connaissances, un développeur externe prestataire rejoint l'entité réceptrice.

Une fois la livraison finale faite, le Tech-Lab passe à la mission suivante. Néanmoins, il reste disponible pour du support opérationnel et pédagogique. Cela inclut, mais ne se limite pas à :

- former des professionnels;
- aider lors de la mise en production;
- aider à résoudre un incident en production.

4 - Mes contributions

Cette section met en lumière mes contributions, en tant que développeur, lors des différentes phases du projet. Le besoin initial, la solution produite en autonomie, le travail fourni ainsi que le raisonnement et la méthodologie sous-jacente sont explicités pour présenter quelques-unes de mes tâches.

Avant tout, il est important d'avoir une vue d'ensemble de la solution finale pour comprendre son fonctionnement et le besoin à l'origine de mes contributions.

La solution finale se nomme FDJ-Envoy, elle est livrée sous forme d'image OCI/Docker (Open Container Initiative) et contient deux composants. Le premier composant est celui sélectionné lors de l'évaluation pratique, Envoy, tandis que le second est notre serveur gRPC d'extension : l'external processor.

Le schéma suivant montre le flux de données de la réception de la requête à l'envoi de la réponse.

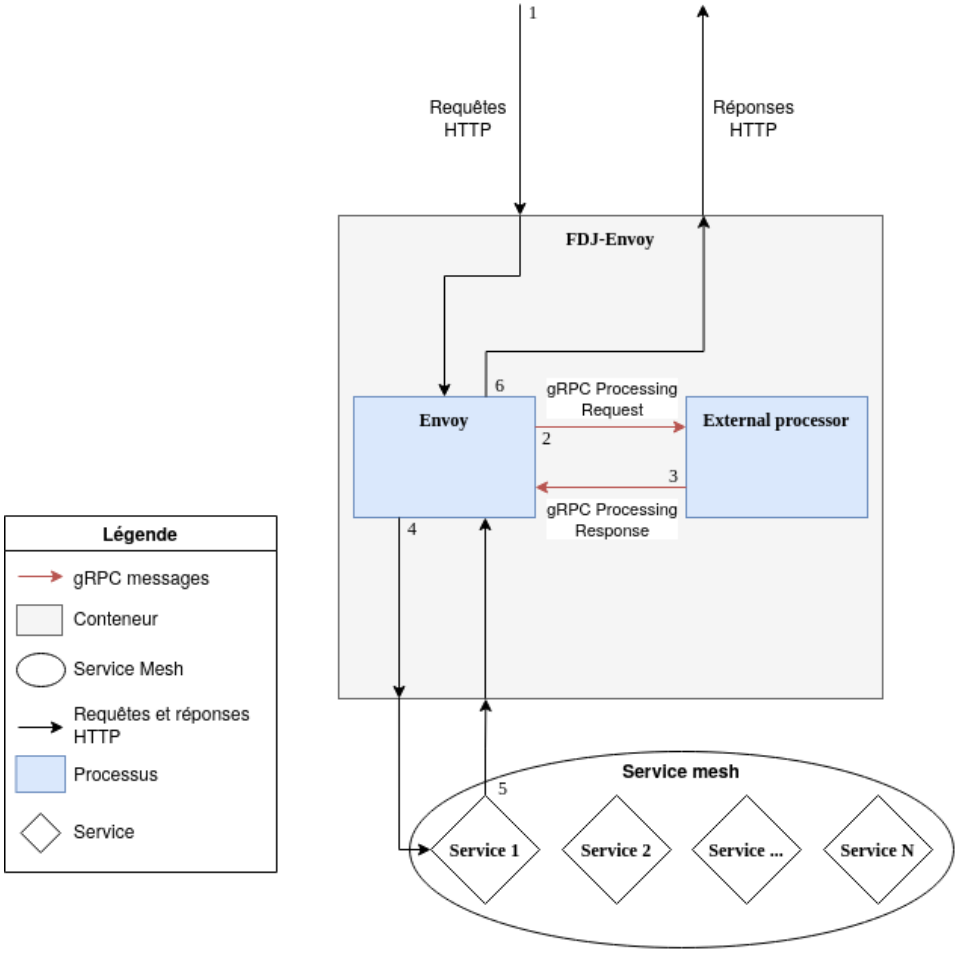


Figure 10 : Vue d'ensemble de FDJ-Envoy.

Les diagrammes de séquence en annexe 10.14 et 10.17 montrent respectivement l'interaction entre le client, FDJ-Envoy et le service, Envoy et l'external processor.

4.1 - Infrastructure logicielle de test d'Envoy

À la fin de l'évaluation pratique, un produit est sélectionné pour être industrialisé et mis à niveau. De nouveaux dépôts de sources sont créés à ce moment-là afin de repartir d'une base propre. Le premier dépôt créé est celui d'Envoy, il contient le matériel nécessaire pour construire et tester l'image OCI/Docker d'Envoy respectant les bonnes pratiques et les conventions de l'entreprise. Le second dépôt, similaire au premier, contient le code source de l'external processor ainsi que le matériel associé. Enfin, le dépôt FDJ-Envoy contient l'assemblage des deux dépôts précédents.

Comme expliqué dans le chapitre 3.7.1, les dépôts sont des livrables. Ils doivent respecter un certain niveau de qualité, et donc, contenir une suite de tests fonctionnels.

4.1.1 - Le besoin

La solution finale étant soumise à des contraintes de qualités, le Product Owner a exprimé le besoin suivant : il faut tester **tous les cas possibles**, passant et cassant. À la manière d'un éditeur logiciel, on veut qualifier **toutes** les fonctionnalités dans la limite du possible.

Il est important de souligner que l'on ne veut pas seulement tester les fonctionnalités individuellement, mais aussi leur combinaison. Prenons l'exemple d'une API Gateway avec une fonctionnalité d'authentification et de routage des requêtes. Les cas de test par fonctionnalité sont les suivants :

- Routage :
    - route invalide;
    - route valide;
- Authentification :
    - pas d'authentification requise;
    - authentification invalide;
    - authentification valide.

Exécuter ces cas de test individuellement n'est pas suffisant. Si une requête contient un jeton d'authentification invalide et référence un service inexistant, c'est bien un status code "404 Not Found" qui doit être retourné et non un "401 Unauthorized". La validation du jeton doit être faite après avoir déterminé le service de destination.

Pour résumer, si X est le nombre de fonctionnalités et Y_n le nombre de cas de test de la fonctionnalité numéro *n, on* a $Y_0 \times Y_1 \times \ldots \times Y_x$ combinaison de cas de tests et non $Y_0 + Y_1 + \ldots + Y_x$. On se retrouve donc avec une explosion combinatoire exponentielle :

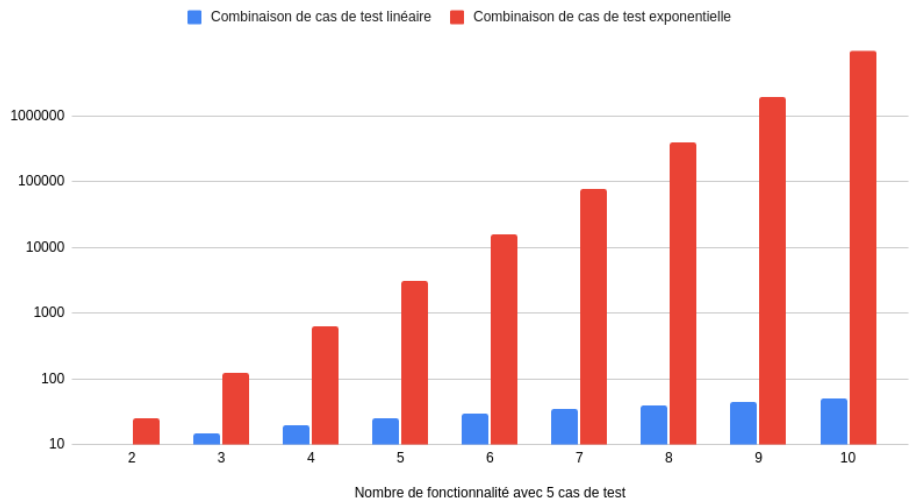


Figure 11 : Graphique à échelle logarithmique du nombre de combinaisons de cas de tests linéaires et exponentiels en fonction du nombre de fonctionnalités.

Le graphique ci-dessus nous alerte sur la faisabilité de la tâche. L'explosion combinatoire n'est pas un problème en soi, ces combinaisons existent dans la réalité et on veut les tester. Il n'y a pas de moyen de réduire le nombre de combinaisons, néanmoins, cette complexité ne doit pas se retrouver dans la suite de tests. Il n'est ni rationnel ni possible d'implémenter plus d'un million de combinaisons de cas de tests manuellement pour tester 10 fonctionnalités avec 5 cas de test chacune.

La question suivante se pose :

Comment couvrir la combinaison exponentielle des cas de tests tout en maintenant un développement linéaire ?

En général, il est recommandé que les testeurs ne possèdent pas une connaissance approfondie des détails d'implémentation d'un composant qu'ils évaluent. Cette approche garantit l'objectivité des tests, valide les spécifications, détecte des scénarios inattendus et prépare les tests à des modifications futures. La section qui suit détaille le fonctionnement interne d'Envoy dans le but de créer un modèle mental favorisant la compréhension de la solution.

4.1.2 - Fonctionnement interne d'Envoy

Envoy est un produit énormément utilisé qui a fait ses preuves dans l'industrie. Il est un composant clé utilisé dans la quasi-totalité des clusters Kubernetes, un outil permettant d'automatiser le déploiement, la mise à l'échelle et la gestion d'applications conteneurisées. Envoy a de nombreuses fonctionnalités et démontre une ingéniosité remarquable. Néanmoins, étant un produit très complet, il est plus complexe à approcher comme le montre l'architecture ci-dessous :

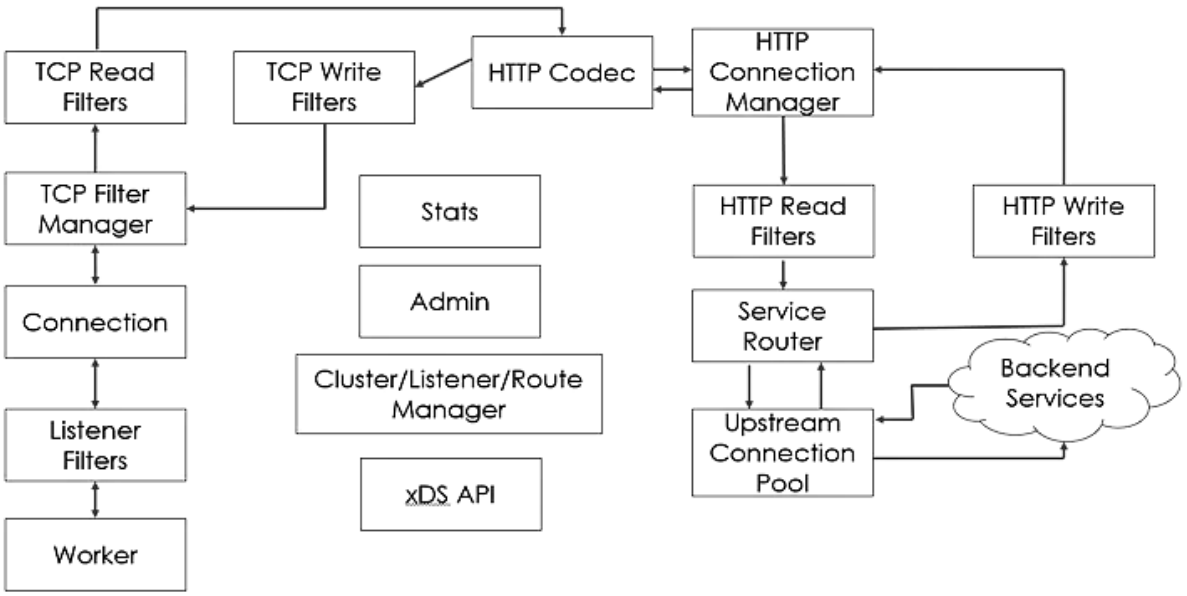


Figure 12 : Vue d'ensemble de l'architecture d'Envoy.

Fort heureusement, seul le sous-système HTTP composé des blocs "HTTP Connection Manager", "HTTP Read Filters", "HTTP Write Filters" et "Service Router" nous intéresse.

Lorsque Envoy reçoit une requête HTTP, le composant "HTTP Connection Manager" la traite. Ce dernier est responsable de la gestion du cycle de vie des flux HTTP. Il est responsable du cycle de vie des requêtes et expose une chaîne de filtres HTTP appelée "HTTP filters".

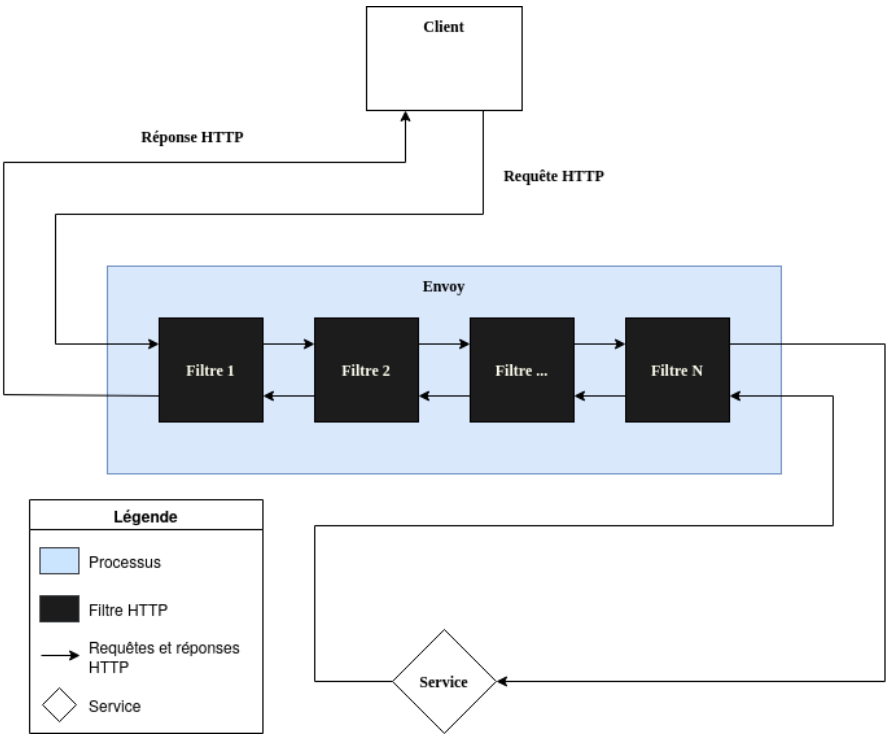


Figure 13 : Chaîne de filtre HTTP du gestionnaire de connexion HTTP d'Envoy.

Les filtres HTTP correspondent aux blocs "HTTP Read Filters" et "HTTP Write Filters" de la figure 12. Ces derniers travaillent à la couche application (couche 7) du modèle OSI, ils ne manipulent pas

des flux d'octet, mais de la donnée structurée. En plus de pouvoir exécuter du code arbitraire, générer une réponse, ils sont capables d'altérer les requêtes et réponses HTTP transitant. Lorsqu'un filtre *N* génère une réponse, le filtre *N+1* n'est pas utilisé et la réponse générée par le filtre *N* est envoyée au client. Lorsqu'un filtre altère une requête, mais ne génère pas de réponse, le filtre suivant est appliqué sur la requête altérée et ainsi de suite.

Enfin, le bloc "Service Router" est toujours le **dernier filtre HTTP** de la chaîne. Il est responsable du transfert de la requête, altérée ou non, au service demandé.

Le pseudo code suivant est un bon modèle mental de l'algorithme de traitement de requête du "HTTP Connection Manager" :

```
function handleHttpRequest(request, filterList)
    for filter in filterList do
        // Altère la requête ou produit une réponse.
        processingResult ← filterProcessRequest(filter, request)
        if isHttpResponse(processingResult) then
            response ← extractHttpResponse(processingResult)
            respondTo(request, response)
            // Requête traitée, les filtres suivants ne sont pas appelés
            return
        else if isAlterRequest(processingResult) then
            request ← alterRequest(request, processingResult)
        end if
    end for
     // Aucun filtre n'a produit de réponse, "Service Router" est absent.
    panic("configuration error, service router HTTP filter is missing")
end function
```

Dans le cadre de notre API Gateway, il y a une correspondance de un pour un entre les fonctionnalités et les filtres HTTP. Une suite de test permettant de tester des filtres HTTP est suffisante pour tester notre API Gateway.

4.1.3 - Environnement de test

L'environnement dans lequel sera exécutée la suite de test est inspiré de ce qui se fait en production. Comme toujours, lorsque l'on effectue un test, il faut se rapprocher de la réalité, simuler des conditions réelles. Autrement, les résultats ne seront pas représentatifs. Cela est d'autant plus vrai lorsque l'on évalue les performances et la résilience d'un produit.

Dans le contexte de la tâche, nous voulons seulement qualifier fonctionnellement le produit. Déployer une cinquantaine de services serait inutile, néanmoins, il faut a minima en déployer cinq.

Quatre services sont nécessaires afin de tester le bon routage et la fonctionnalité proxy vers des services HTTP/1.1, HTTP/2, avec et sans le protocole TLS (Transport Layer Security). Chaque service supporte un ensemble d'opérations avec des réponses prédéfinies et une opération écho. Cette dernière renvoie, dans le corps de la réponse, la requête reçue.

Un dernier service est indispensable pour produire des jetons JWT (Json Web Token), spécifiquement des “ID token” et des “access token”, conformément au protocole OpenID Connect (OIDC). La prochaine mission du Tech-Lab étant une solution d’Identity Provider (IdP) utilisant ce protocole, il fait sens que l’API Gateway soit capable de valider ces jetons.

Pour avoir un maximum de contrôle sur l’environnement, les services utilisés sont des bouchons (mock en anglais) développés, testés et maintenus par l’équipe.

L’unique composant non maintenu par l’équipe est l’injecteur, le client HTTP: K6. Ce dernier est un outil de test de performance qui peut aussi être utilisé pour écrire des tests fonctionnels. Le contexte industriel de FDJ incite à contrôler chaque élément et ne pas laisser de place au hasard, c’est pourquoi nous qualifions aussi l'outil de test lui-même. Les différentes missions ont montré les points forts, les limitations et même quelques bugs de l’outil.

Voici une vue d’ensemble de l’environnement de test :

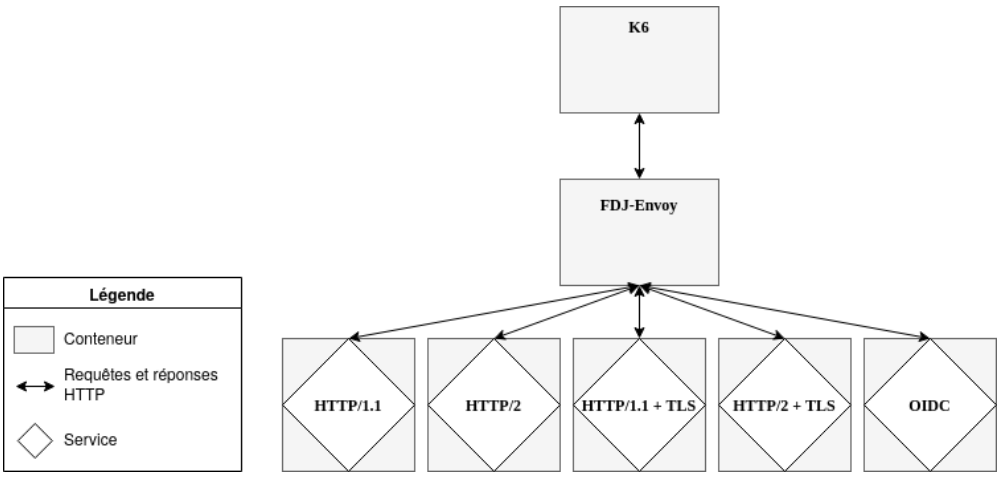


Figure 14 : Vue d’ensemble de l’environnement de test.

La ressemblance avec la figure 10 est notable, l’environnement de test est au plus proche de la réalité.

Afin d’améliorer l'expérience développeur et avoir un unique environnement, chaque composant est déployé dans un conteneur Docker. Peu importe le lieu d’exécution de la suite de tests, que ce soit en chaîne d’intégration continue ou en local sur le poste de travail, les tests donneront les mêmes résultats.

Le fonctionnement interne d’Envoy et l’environnement de test étant explicités, il est désormais possible de concevoir et d'implémenter une solution.

4.1.4 - Conception et implémentation

Pour répondre au besoin, il fait sens de commencer par approcher une version simplifiée du problème.

Prenons une instance d’Envoy configurée pour agir comme un proxy vers des services. On souhaite écrire des tests unitaires qui vérifient que la requête est envoyée au bon service. Trois cas de test doivent être rédigés : le premier vise à confirmer l'acheminement correct de la requête vers le service

1, le second vise à confirmer l'acheminement correct de la requête vers le service 2, et le troisième vise à vérifier que lorsqu'un service est absent, une erreur “404 Not Found” est renvoyée de manière appropriée.

Si l’on suit la méthodologie de test unitaire AAA. Chaque cas de test se découpe alors en trois phases :

- **A**rrange : création de la requête HTTP (en-têtes, corps, trailers);
- **A**ct : envoie de la requête et réception de la réponse;
- **A**ssert : validation de la réponse.

Le cas de test échoue si, lors du “Assert”, le résultat obtenu ne correspond pas à l’attendu. Voici le pseudo-code du premier cas :

```
function TestProxyService1
    // Arrange
    arrange ← newHttpClient
    method ← "GET"
    endpoint ← "http://service1.test-env.local/name"
    request ← newHttpRequest(method, endpoint)


    // Act
    response ← doRequest(client, request)


    // Assert
    expectedStatusCode ← 200
    statusCode ← extractStatusCode(response)
    assertEqual(expectedStatusCode, statusCode)


    expectedServiceName ← "service1"
    serviceName ← extractBody(response)
    assertEqual(expectedServiceName, serviceName)
end function
```

Les autres cas de tests sont similaires, seules quelques variables des phases “Arrange” et “Assert” changent.

Prenons maintenant un cas plus complexe et plus abstrait, l’API Gateway est configurée avec deux filtres. Chaque filtre effectue deux vérifications distinctes et émet une réponse différente selon l’erreur. Comme expliqué dans la section 4.1.2, si un filtre génère une réponse, les filtres suivants ne sont pas appelés et la requête n’atteint pas le service.

Dès lors que plusieurs filtres sont configurés, la situation se complique. Il devient nécessaire non seulement de créer des requêtes pour satisfaire ou ne pas satisfaire chaque filtre individuellement, mais de composer avec une multitude de possibilités.

Au-delà d’un certain nombre, il est préférable de décrire les différents cas sous formes de tableaux. Cela permet d’avoir une meilleure vue d’ensemble.

	Arrange		Assert	
	Filtre 1	Filtre 2	Source de la réponse	Cause de la réponse
1	OK	OK	Service X	*N/A*
2	OK	**KO*A***	Filtre 2	Erreur A
3	OK	**KO*B***	Filtre 2	Erreur B
4	**KO*Y***	OK	Filtre 1	Erreur Y
5	**KO*Z***	OK	Filtre 1	Erreur Z
6	**KO*Y***	KO*A*	Filtre 1	Erreur Y
7	**KO*Y***	KO*B*	Filtre 1	Erreur Y
8	**KO*Z***	KO*A*	Filtre 1	Erreur Z
9	**KO*Z***	KO*B*	Filtre 1	Erreur Z

OK: la requête est valide pour ce filtre.
KO*X*: la requête est invalide pour la raison *X*, le filtre va émettre une réponse.

Tableau 2 : Matrice d'exemple des combinaisons de cas de test d'une chaîne de filtres HTTP de taille trois.

Le tableau ci-dessus comporte deux colonnes, "Arrange" et "Assert", qui contiennent elles-mêmes des sous colonnes. Les deux premières sous-colonnes décrivent la combinaison de cas de test tandis que les deux dernières décrivent l'acteur ainsi que la raison à l'origine de la réponse.

Lorsque l'on combine plus d'un filtre, il faut, comme expliqué dans la section 4.1.1, évaluer la source de la réponse reçue. À la ligne numéro six du tableau, c'est bien le filtre 1 qui est à l'origine de la réponse, et ce, même si le filtre suivant est supposé générer une réponse. La source d'une réponse est le premier filtre de la chaîne dont on attend un rejet.

La cause de l'erreur est aussi importante. Par exemple, un filtre en charge de la validation des jetons d'authentification pourrait générer des réponses distinctes en cas d'expiration du jeton ou d'invalidité de sa signature. C'est la raison pour laquelle il est impératif de valider la cause à l'origine de la génération d'une réponse par un filtre.

Abordons maintenant les diverses possibilités de mutualisation de nos cas de tests. Pour rappel, l'objectif n'est pas simplement de créer neuf tests unitaires, mais bien de trouver une approche permettant de développer seulement six tests.

Pour mutualiser les combinaisons de cas de tests, il faut déterminer les parties ou sous parties qui sont mutualisables. Si l'on observe chaque ligne du tableau, on ne remarque aucun duplicata. Chaque combinaison de cas de tests est unique. Néanmoins, si l'on se focalise sur les sous-colonnes de "Assert", on observe des doublons.

Ces doublons signifient que ces tests unitaires partagent la même phase “Assert”, le résultat attendu est le même. Par exemple, les combinaisons de cas de test de la ligne 4 et 6 attendent la même réponse.

En programmation informatique, la duplication d'une action ou d'une procédure est souvent un signal indiquant qu'il est judicieux de créer une fonction.

```
function TestCase4
    // Arrange
    client ← newHttpClient
    request ← newHttpRequestForTestCase4

    // Act
    response ← doRequest(client, request)

    // Assert
    assertResponseIsFromFilter1ErrorY(response)
end function
```

```
function TestCase6
    // Arrange
    client ← newHttpClient
    request ← newHttpRequestForTestCase6

    // Act
    response ← doRequest(client, request)

    // Assert
    assertResponseIsFromFilter1ErrorY(response)
end function
```

Même si l'on semble se rapprocher de la solution, la mutualisation des vérifications de la réponse ne résout pas le problème. La section "Arrange" de chaque test diffère, ce qui signifie que la complexité exponentielle persiste dans le code. Le découpage par phases semble avoir ses limites. Il faut trouver une alternative pour regrouper le code de manière plus efficace.

Si l'on se réfère au tableau, les lignes sous la colonne "Arrange" sont uniques, cependant, certaines cellules apparaissent à plusieurs reprises. Une fois de plus, c’est un signe qu’il est possible de mutualiser.

La phase "Arrange" garantit qu'à son terme, une requête HTTP est prête à être envoyée. Cette dernière contient les données ou métadonnées nécessaires pour être acceptées ou rejetées par les filtres. Deux options sont possibles : composer la requête à l’aide de “morceaux” de requête ou bien à l’aide de fonctions.

La première option présente moins de flexibilité, explorons donc la seconde.

```
function TestCase4WithComposition
    // Arrange
    client ← newHttpClient
    request ← newHttpRequest
    request ← arrangeFilter1ErrorY(request)
    request ← arrangeFilter2Valid(request)

    // Act
    response ← doRequest(client, request)

    // Assert
    assertResponseIsFromFilter1ErrorY(response)
end function
```

```
function TestCase6WithComposition
    // Arrange
    client ← newHttpClient
    request ← newHttpRequest
    request ← arrangeFilter1ErrorY(request)
    request ← arrangeFilter2ErrorA(request)

    // Act
    response ← doRequest(client, request)

    // Assert
    assertResponseIsFromFilter1ErrorY(response)
end function
```

Les différents filtres ayant des champs d'actions bien définis, il ne devrait pas y avoir de collision entre les fonctions modifiant la requête. Cela ne doit pas arriver :

Requête en sortie de ArrangeFilter1ErrorY		Requête en sortie de ArrangeFilter2ErrorA

`GET /a/path HTTP/1.1`
`Host: service1.test-env.local`
`X-Header: foo`
	⟶	`GET /a/path HTTP/1.1`
		`Host: service1.test-env.local`
		`X-Header: bar`

Il demeure donc judicieux que les fonctions vérifient à ne pas écraser de valeurs et qu'elles fassent échouer le test autrement.

Malgré l'élimination de toute duplication de code, la complexité exponentielle demeure. En raison de la composition manuelle des tests, l'explosion combinatoire persiste dans le code.

L'unique moyen de résoudre ce problème est de ne pas avoir une composition statique des cas de test, mais de les générer dynamiquement, pendant l'exécution de la suite de test. De cette manière, il est possible de supprimer toutes les fonctions non mutualisées.

Pour ce faire, il faut introduire le concept d'infrastructure logicielle.

Une infrastructure logicielle ou un framework en anglais est une base structurée et cohérente fournissant des outils, des conventions et des modèles de conception pour faciliter le développement, la personnalisation et la maintenance d'applications logicielles. Elle offre un cadre préétabli qui guide le processus de développement en fournissant des fonctionnalités de base, des abstractions, des patrons de conception et des composants réutilisables. Elle se distingue des bibliothèques logicielles qui sont moins généralistes et n'imposent pas de structure particulière. Une infrastructure logicielle peut s'imposer comme le point d'entrée du programme.

L'infrastructure logicielle aura pour rôle de composer et d'exécuter les différentes combinaisons de cas de test. Afin de faciliter la composition, les tests seront écrits dans un format particulier et devront exposer un objet implémentant l'interface **TestSuite**.

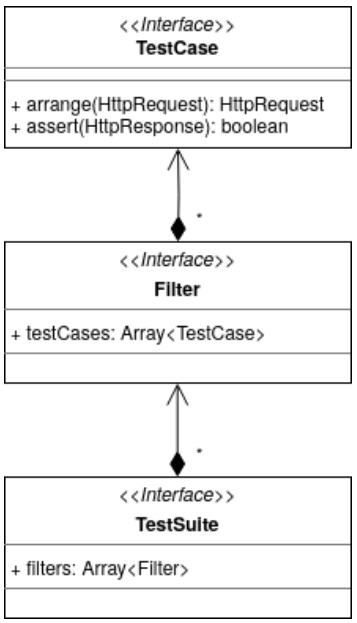


Figure 15 : Diagramme de classe d'une suite de tests respectant le format de l'infrastructure logicielle

Il convient de noter que l'interface **TestCase** ne comporte pas de méthode "act", car celle-ci est commune à l'ensemble de la suite.

La **TestSuite** exposée est ensuite utilisée par l'infrastructure logicielle pour recréer les combinaisons une par une. Un algorithme de parcours d'arborescence est suffisant si l'on modélise les combinaisons sous formes d'arbres.

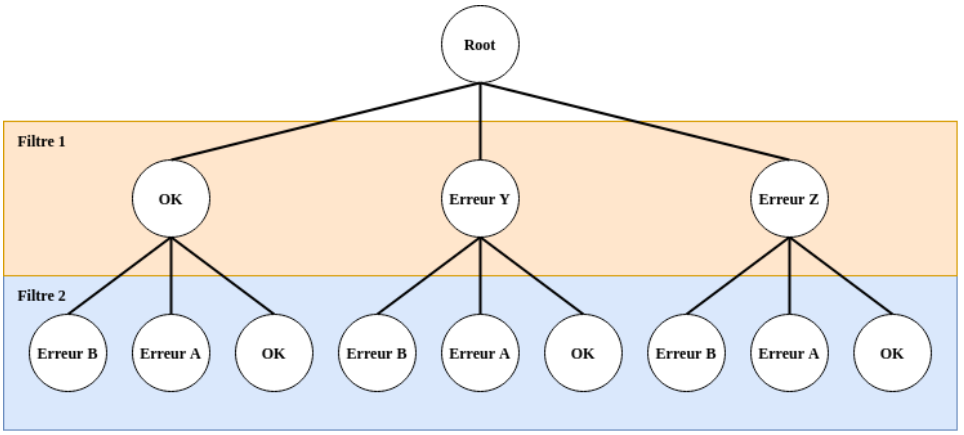


Figure 16 : Arborescence des combinaisons des cas de tests

Dans la figure ci-dessus, chaque chemin vers une feuille représente une combinaison unique.

Voici un tableau approximatif des complexités temporelles et spatiales moyennes pour différents algorithmes de parcours d'arbres, en fonction du nombre de nœuds dans l'arbre.

Algorithme	Complexité temporelle	Complexité spatiale
Parcours en profondeur	O(n)	O(h)
Parcours en largeur	O(n)	O(w)
Zigzag Traversal	O(n)	O(n)
Vertical Order Traversal	O(n log n)	O(n)
Topological Sort	O(V + E)	O(V)

- n : nombre total de nœuds dans l'arbre
- h : hauteur de l'arbre
- w : largeur du niveau le plus large de l'arbre (diamètre de l'arbre)
- V : nombre de nœuds (vertices) dans le graphe (utilisé pour le tri topologique)
- E : nombre d'arêtes dans le graphe (utilisé pour le tri topologique)

Tableau 3 : Complexité en temps et en espace des algorithmes de profondeur dans le pire cas.

Le parcours en profondeur est le choix idéal, car performant et simple d'implémentation. L'algorithme écrit en JavaScript est disponible en annexe 10.18.

Dès lors que l'algorithme atteint un nœud feuille, un nœud sans enfant, la combinaison est complète et exécutée dans l'instant. L'exécution suit la méthodologie AAA et se fait en trois temps.

Premièrement, une requête vierge est créée. Celle-ci est agrémentée par les méthodes *arrange* des différents **TestCase**. Une fois toutes les méthodes exécutées, la requête est prête à être envoyée.

```
function ExecuteSingleTest(combination)
    // Arrange
    client ← newHttpClient
    request ← newHttpRequest
    for index, testCase in combination do
        request ← testCase.arrange(request)
    end for
    ...
```

L'étape "Act" reste inchangée, la requête est envoyée et une réponse est reçue. Maintenant, il faut déterminer si elle est conforme à l'attendu.

```
    ...
    // Act
    response ← doRequest(client, request)
    ...
```

Si un filtre est à l'origine de la réponse, il faut déterminer lequel. Autrement, il n'est pas possible d'évaluer la réponse, car l'attendu est inconnu. Dans le cas où la réponse provient du service, il faut appeler la méthode *assert* de chacun des **TestCase**. Dans l'état actuel des choses, l'infrastructure

logicielle ne dispose pas des moyens nécessaires pour déterminer si un filtre est à l'origine de la réponse.

Il faut ajouter un marqueur, permettant de savoir si la méthode "arrange" d'un **TestCase** va retourner une requête qui sera rejetée par le filtre. Dans le pseudo-code ci-dessous, la propriété "isNotValid", positionnée par les développeurs des tests, joue le rôle du marqueur.

```
function ExecuteSingleTest(combination)
    // Arrange
    client ← newHttpClient
    // La valeur -1 signifie que la réponse provient du service.
    responseSource ← -1
    request ← newHttpRequest
    for index, testCase in combination do
        request ← testCase.arrange(request)
        if responseSource == -1 and testCase.isNotValid then
            responseSource ← index
        end if
    end for
    ...
```

Il devient alors possible d'évaluer la réponse.

```
    // Arrange
    // La réponse provient du service.
    if responseSource == -1 then
        for testCase in combination do
            if testCase.assert(response) then
                panic("actual response doesn't match expected")
            end if
        end for
    else
        // La réponse provient d'un filtre.
        assert ← combination[responseSource].assert
        if assert(response) then
            panic("actual response doesn't match expected")
        end if
    end if
end function
```

L'architecture logicielle est désormais en mesure d'exécuter dynamiquement l'ensemble exponentiel des combinaisons. Cependant, elle reste peu utilisable, car un point était omis jusqu'à présent.

Les filtres ne sont pas activés pour toutes les routes. Par exemple, certains services ne requièrent pas d'authentification. Jusqu'ici, ce n'était pas un problème, parce que peu importe le chemin emprunté, on supposait que tous les filtres étaient activés.

Il faut que l'opération, l'URL et la méthode de la requête, soient définies par l'infrastructure logicielle, autrement, les cas de tests seront couplés et l'ajout d'une nouvelle fonctionnalité risque de casser la suite de test.

4.1.5 - Le routeur

Afin d'attribuer une route, ou opération, à une combinaison, il est nécessaire de développer une fonctionnalité de routage dans l'infrastructure logicielle. Pour les mêmes motifs qui limitent la rédaction exhaustive des combinaisons de cas de tests, il n'est pas possible d'assigner de manière statique une route à une combinaison spécifique.

Le routeur doit être capable, à partir d'une liste de routes et d'une combinaison, de trouver une route permettant d'exécuter la combinaison. Si aucune route ne le permet, un avertissement est levé.

Deux possibilités sont à considérer : partir d'une route et construire une combinaison ou partir d'une combinaison et trouver une route.

La deuxième option semble être celle à privilégier, car l'algorithme de recherche des combinaisons est déjà en place. De plus, détecter une combinaison sans route ne nécessitera pas d'effort supplémentaire.

Chaque cas de test doit pouvoir spécifier un ensemble de contraintes. La route trouvée par le routeur doit être conforme à la somme des contraintes spécifiées par les différents cas de test de la combinaison.

Après examen de la suite de tests à mettre en place, les contraintes doivent être conçues pour indiquer la présence ou l'absence de filtre sur la route.

Si l'on prend un filtre d'authentification quelconque par exemple, on veut pouvoir vérifier qu'une requête avec un jeton invalide n'est pas rejetée lorsque le filtre est désactivé.

Il est aussi primordial de pouvoir ajouter des contraintes arbitraires telles que la version du protocole HTTP supportée ou bien la possibilité d'ajouter un corps au sein de la requête.

Le nombre de possibilités étant sans limites, la solution doit être assez générale pour couvrir tous les cas d'usage.

Dans ce cas, le plus simple et le plus flexible est d'implémenter un système de contraintes nominatives. Chaque route contient une liste d'attributs tandis que chaque cas de tests spécifie un ensemble d'attributs qui doivent être présents ou absents.

Voici un exemple d'une liste de routes et de leurs attributs :

	Routes	Attributs
1	GET https://service-1.test-env.local/echo	`HTTP2, TLS`
2	POST http://service-1.test-env.local/echo	`HTTP2, with_body`
3	GET https://service-2.test-env.local/users	`TLS`
4	POST https://service-1.test-env.local/users	`HTTP2, TLS, authentication, with_body`
5	DELETE https://service-1.test-env.local/users	`HTTP2, TLS, authentication, with_body`

Tableau 4 : Exemple d'une liste de routes et de ses attributs.

Si une combinaison de **TestCase** nécessite l'envoi d'une requête avec un corps (`with_body`) mais sans authentification (`authentication`), le routeur doit choisir la route numéro 2. Si une combinaison nécessite l'envoi d'une requête avec un corps vers un service HTTP/1.1, donc sans l'attribut `HTTP2`, le routeur doit générer un avertissement car aucune route ne le permet.

```
function FindRoute(routes, requirementSet)
    for requirement in requirementSet do
        if requirement.value == REQUIRED then
            // On ne garde que les routes qui ont l'attribut.
            routes ← SELECT(routes, requirement.name)
        else if requirement.value == EXCLUDED then
            // On supprime les routes qui ont l'attribut.
            routes ← FILTER(routes, requirement.name)
        end if
    end for
    if routes.length == 0 then
        WARN("no route found for this combination")
    end if
     return routes[0]
end function
```

Dans le pseudo-code ci-dessus, le choix de retourner la première route a été fait, cependant il est également envisageable de sélectionner une route aléatoirement ou même de retourner toutes les routes et d'exécuter le cas de tests sur chacune d'elles. L'algorithme d'exécution d'un seul test est disponible en annexe 10.20 tandis que le diagramme de classe finale est disponible en annexe 10.19.

Un aspect qui n'a pas encore été discuté concerne la création de la liste de routes. L'infrastructure logicielle n'impose pas de restriction, les opérations et attributs peuvent être générés dynamiquement ou rédigés statiquement selon l'envie du développeur.

4.1.6 - Rétrospective

Pour conclure cette étape, une rétrospective sur les aspects positifs et négatifs de l'infrastructure logicielle s'impose. Cette dernière est utilisée dans nos dépôts d'Envoy et FDJ-Envoy.

L'infrastructure logicielle a permis d'accroître la vélocité de développement des tests en les découplant. L'ajout de cas de tests pour une nouvelle fonctionnalité s'effectue sans modifications des autres.

Le format exigé des cas de tests facilite la lecture et la maintenance de la suite. Tous les tests suivent la méthodologie AAA.

Enfin, la segmentation des tests par filtres ou fonctionnalités a permis la mutualisation des tests fonctionnels et de performance. Une propriété "weight" a été ajoutée à l'interface **TestCase** afin de pouvoir pondérer les différents cas. De cette manière, configurer et exécuter un tir de performance avec ⅓ des requêtes non authentifiées, ⅓ des requêtes authentifiées et valides et ⅓ des requêtes authentifiées, mais invalide se fait à moindre coût.

Le seul aspect négatif qui a entravé l'adoption de l'infrastructure logicielle est sa documentation. En raison de sa qualité insuffisante, elle fut incomprise par une partie de l'équipe. Toutefois, de la même manière que le développement, la documentation évolue itérativement et est maintenant comprise par l'ensemble de l'équipe.

Pour conclure, le choix de développer l'infrastructure logicielle s'est avéré extrêmement bénéfique. Non seulement elle a considérablement réduit le temps nécessaire, mais elle a également simplifié la mise en place des tests et a permis une mutualisation efficace des tests fonctionnels et de performance. D'un point de vue personnel, les diverses étapes d'évaluation du projet m'ont permis de découvrir une facette essentielle du développement logiciel qui m'était jusqu'alors inconnue. Les objectifs, l'approche et les compétences requises diffèrent grandement du développement pur, nécessitant d’avoir une approche méthodique et cartésienne.

4.2 - Rotation à chaud des fichiers de journalisations

Ce chapitre présente le travail que j'ai réalisé pour implémenter la rotation à chaud des fichiers de logs de l'external processor. L'intérêt de ces fichiers et la nécessité d'en faire la rotation sera explicité en premier lieu. Une fois le besoin présenté et justifié, les différents choix qui ont amené à l'algorithme final ainsi que son implémentation seront présentés. Enfin, il conviendra de conclure avec une rétrospective sur les problèmes rencontrés et la méthodologie de travail appliquée.

4.2.1 - Contexte

Après avoir sélectionné Envoy comme produit d'API Gateway à la sortie de l'évaluation pratique, la phase d'industrialisation et de mise à niveau commence. Il a été choisi d'étendre les fonctionnalités du produit via l'utilisation d'un filtre HTTP nommé "External Processing".

Pour rappel, le fonctionnement d'Envoy et de ses filtres sont présentés dans la section 4.1.2.

Le filtre "External Processing" permet, comme son nom l'indique, de déléguer le traitement d'une requête ou d'une réponse dans un composant externe à Envoy. Le filtre agit comme un adaptateur et la logique est implémentée dans un programme à part, un serveur gRPC. Un diagramme de séquence du traitement d'une requête avec l'external processor est disponible en annexe 10.14.

Étant un composant et un serveur à part entière, l'external processor est, comme toute entité déployée en production, de qualité industrielle. De nombreux facteurs contribuent à l'industrialisation d'un composant et notamment son niveau d'**observabilité**. La définition suivante est issue du guide d'observabilité de Honeycomb :

Observability is about being able to ask arbitrary questions about your environment without — and this is the key part — having to know ahead of time what you wanted to ask.

Toutes applications et composants en production doivent avoir un niveau d'observabilité suffisant. Autrement, en cas d'incident, le responsable de l'entité, que ce soit une application ou notre API Gateway, est complètement aveugle et ne peut faire de diagnostic.

L'observabilité aspire à résoudre ce problème. Un composant avec un bon niveau d'observabilité permet de détecter lorsqu'une inconnue survient et, potentiellement, d'en comprendre la source. L'observabilité est essentielle car il n'est pas possible, même si le nombre diminue avec l'expérience, de supprimer toutes les inconnues.

Une multitude de moyens existe pour rendre une application observable. Le choix logique d'implémenter des métriques et des journaux d'événements a été fait pour l'external processor.

Les métriques sont, comme leurs noms l'indiquent, des mesures, des données. Par exemple, le temps moyen d'une réponse, le nombre de requêtes et d'erreurs sont des métriques de l'external processor.

Les fichiers de logs ou journaux d'événements sont des fichiers écrits continuellement par un programme informatique pendant son exécution. Ils contiennent généralement, comme leur nom l'indique, l'historique des événements du programme. Par exemple, un serveur web qui écrit dans un fichier la date, l'heure, le statut de la réponse ainsi que l'adresse IP de chaque requête qu'il traite

constitue un fichier de logs. A contrario, lorsqu'un logiciel de traitement de texte sauvegarde un document, ce n'est pas un fichier de logs.

Dans le cadre de notre external processor, nous avons deux journaux d'événements : un premier pour les requêtes traitées (les accès) et un second pour les autres événements. Grâce à ces deux flux de logs, il est possible de savoir précisément ce qu'il se passe au sein de notre composant. Toutes actions fonctionnelles du serveur telles que la réception ou le rejet d'une requête entraîne l'écriture d'un ou plusieurs événements dans les journaux. Par souci de clarté, il a été choisi de se concentrer uniquement sur le journal des requêtes traitées, communément appelé journal d'accès ou "access log" en anglais.

La simple présence de journaux d'événements n'est pas suffisante pour considérer le serveur comme prêt à être déployé.

4.2.2 - Le besoin

Étant destiné à être déployé dans un environnement à haut taux transactionnel continu, notre serveur va écrire une ligne pour chaque requête traitée. Chaque ligne a une taille moyenne de 120 caractères. En simulant une moyenne basse de 300 requêtes par seconde pendant une heure, c'est **125Mo** de logs qui seront écrits dans le journal. Le premier gigaoctet est atteint en 8 heures tandis que le premier téraoctet est atteint en 333 jours. Ce n'est qu'une question de temps avant que le programme se heurte aux limites physiques de la machine et que des événements soient perdus.

Les journaux d'événements, tout comme les métriques, sont consommés par des applications d'administration afin d'être analysés, stockés, etc. Cela permet, par exemple, de détecter des schémas d'attaques, de tracer le chemin d'une transaction au sein du SI en cas d'incident ou bien fournir des informations aux autorités sur demande. Certains journaux, celui d'accès notamment, doivent être conservés une durée spécifique pour des raisons légales. Il n'est pas envisageable de perdre des logs par manque d'espace disque.

La rotation à chaud des journaux d'événements permet de résoudre ce problème. Une rotation de fichier consiste à changer le fichier journal utilisé. L'ancien fichier peut ensuite être déplacé sur une autre machine, compresser ou bien supprimer.

Les raisons explicitées précédemment ont mené les demandeurs à exprimer le besoin d'avoir un mécanisme de rotation des fichiers de logs. Les exigences non fonctionnelles suivantes découlent de leurs demandes et du contexte industriels de FDJ :

- le fichier actuel est conservé en cas d'erreur;
- le taux transactionnel ne doit pas chuté;
- aucun événement ne doit être perdu;
- les événements sont atomiques: le début d'une entrée jusqu'à sa fin est écrit dans le même fichier.

Les exigences ci-dessus se traduisent par des contraintes d'implémentation :

- gérer proprement les erreurs;
- ne pas ajouter de contention dans le système de logs;
- ne perdre aucun événements;
- garantir l'atomicité des événements;

- libérer les ressources associées au fichier après la rotation.

Comme il sera démontré par la suite, la dernière contrainte est implicite et relève des bonnes pratiques de programmation.

Avant d'entamer toute réflexion, la première question à se poser est : " Comment est-ce implémenté ailleurs ? "

4.2.3 - L'implémentation d'Apache2, NGINX et Envoy

La rotation de fichiers étant un problème commun, des personnes ont déjà abordé le problème. Si une solution générique existe, il n'est pas utile de réinventer la roue.

Prenons le cas d'Apache2 ou httpd, un des serveurs web les plus utilisés au monde. Leur documentation propose deux solutions :

- écrire dans l'entrée standard de "rotatelogs";
- redémarrer le serveur à chaud

Voyons le détail de chacune de ces solutions.

La première solution consiste à utiliser l'entrée standard de "rotatelogs" comme fichier de logs. "rotatelogs" est un utilitaire fourni avec Apache2 qui reçoit en entrée un flux de données et écrit dans un fichier ce même flux. L'utilitaire est capable de changer de fichier dès lors que le flux écrit atteint une certaine taille ou bien à intervalles réguliers. Dans la figure ci-dessous, "rotatelogs" est configuré pour changer de sortie chaque jour et utilise la date du jour comme suffixe.

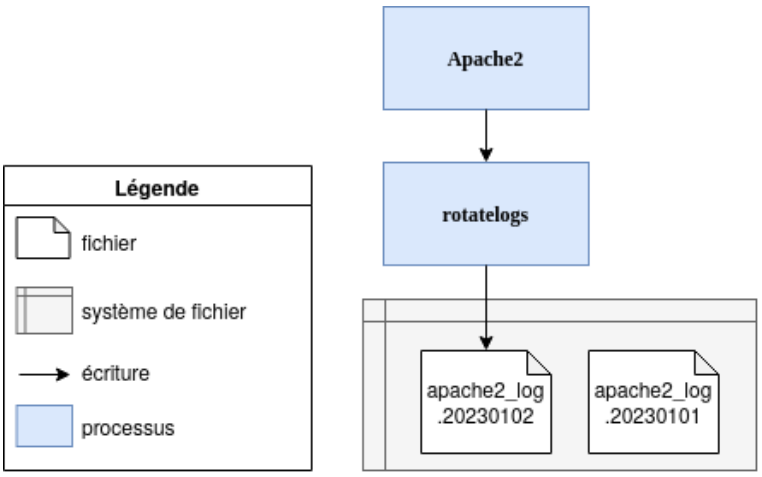


Figure 17 : Flux de log d'Apache2 au fichier de log en passant par rotatelogs

L'un des avantages majeurs de cette solution est que le programme, ici Apache2, bénéficie d'une rotation de logs à moindre coût. Il sera démontré par la suite que d'effectuer une rotation de logs n'est pas une simple tâche, surtout lorsque l'application génère des logs en concurrence. Néanmoins, cette option vient avec les limitations suivantes :

- flexibilité : nous sommes limités par les options de "rotatelogs";
- dépendance : ajout d'une dépendance externe, notre programme n'est plus autonome;
- pas d'atomicité des logs : le programme peut écrire deux moitiés de log dans deux fichiers différents.

Même si les deux premières limitations ne sont pas documentées comme rédhibitoires, elles le sont. Voyons maintenant la seconde solution, celle qui est recommandée dans leur documentation.

La rotation de log via redémarrage à chaud consiste à renommer (ou déplacer) le fichier de log puis redémarrer le serveur à chaud. Lors de cette manipulation, le serveur est arrêté et démarré en parallèle. Pendant la séquence d'arrêt, le serveur arrête d'accepter de nouvelle connexion, il finit de traiter les requêtes en cours puis ferme le fichier de logs. Lors du démarrage, le serveur crée et ouvre le fichier de logs et se met à l'écoute de requêtes. Chaque nouvelle requête traitée aura une entrée dans le nouveau journal d'événements.

En résumé, une fois que le fichier de logs est déplacé, le serveur **rouvre** le fichier de logs. Cette simple opération est le cœur de toute rotation de fichier.

Lorsqu'un processus ouvre un fichier, le noyau Linux lui retourne un "File Descriptor". Ce dernier est un identifiant unique au processus permettant d'identifier un fichier, un tuyau ("pipe" en anglais) ou une socket réseau. Lorsqu'un processus lit, écrit ou change les permissions d'un fichier, il ne le référence pas via son chemin. De cette manière, un fichier peut être déplacé (en restant sur le même système de fichiers) pendant que des processus lisent et écrivent dedans.

Une réouverture de logs se résume donc essentiellement à obtenir un nouveau "File Descriptor", l'échanger avec l'actuel puis fermer l'ancien.
L'étape d'échange des "Files Descriptors" est la source de la complexité d'une réouverture. Dans les applications hautement concurrentielles telles que les serveurs web, une mauvaise implémentation peut entraîner la perte de logs ou de la contention. Cette étape d'échange peut grandement influer sur le design du système de log d'une l'application, c'est pourquoi il doit être pensé en amont.

L'étape d'échange des "File Descriptor" diffère dans chaque implémentation. Chaque système de logs nécessite une implémentation différente. C'est la raison pour laquelle il n'y a pas de solution générique. Il faut concevoir une solution adaptée au système de logs de l'external processor.

Dans les applications hautement concurrentielles telles que les serveurs web, une mauvaise implémentation peut entraîner la perte de logs ou de la contention. L'échange des "File Descriptor" peut grandement influer sur le design du système de log de l'application, c'est pourquoi il doit être pensé en amont.

Apache2 n'a pas de mécanisme de rotation de logs interne et fournit donc un utilitaire dédié à cette tâche. Leur solution de redémarrage à chaud fonctionne grâce à un effet de bord : la réouverture du fichier de logs.

Si l'on étudie ce qui est fait ailleurs, notamment notre composant Envoy, il expose un mécanisme de réouverture de logs via son interface d'administration. C'est aussi l'approche suivie par NGINX, un autre serveur web populaire.

Si ces composants n'effectuent pas la rotation eux-mêmes, c'est parce qu'ils ne peuvent pas. Étant déployés dans divers environnements de production, ils ne font donc pas de supposition sur ce dernier: rien ne garantit que le serveur a les permissions de renommer le fichier, en créer un nouveau, etc.

C’est la raison pour laquelle la réouverture est la méthode privilégiée, elle est flexible et n’impose aucune contrainte sur la stratégie de rotation. De cette manière, n’importe quel utilitaire peut être utilisé pour faire la rotation : “logrotate”, “logadm” ou bien même une solution interne.

Enfin, notre external processor va être déployé avec Envoy, par conséquent il est pertinent de prendre la même approche et d’implémenter une réouverture de logs. Même si une réouverture du fichier de logs n’est pas une rotation, cela répond au besoin.

Avant d’aborder l’algorithme de réouverture, il est requis de connaître le fonctionnement du système de journalisation de l’external processor.

4.2.4 - Système de journalisation de l’external processor

Au démarrage, notre external processor ouvre un fichier pour écrire ses logs d’accès. Dès lors qu’il reçoit une requête, il crée une “goroutine”, un mécanisme de concurrence propre au langage Go similaire au green thread ou thread virtuel. Cette dernière traite la requête, génère un log d’accès puis se termine. Afin de minimiser le temps de traitement d’une requête et ne pas impacter le taux transactionnel, l’événement généré est transféré à une goroutine responsable des opérations d’entrées sorties. Le transfert se fait sans contention et sans allocation mémoire.

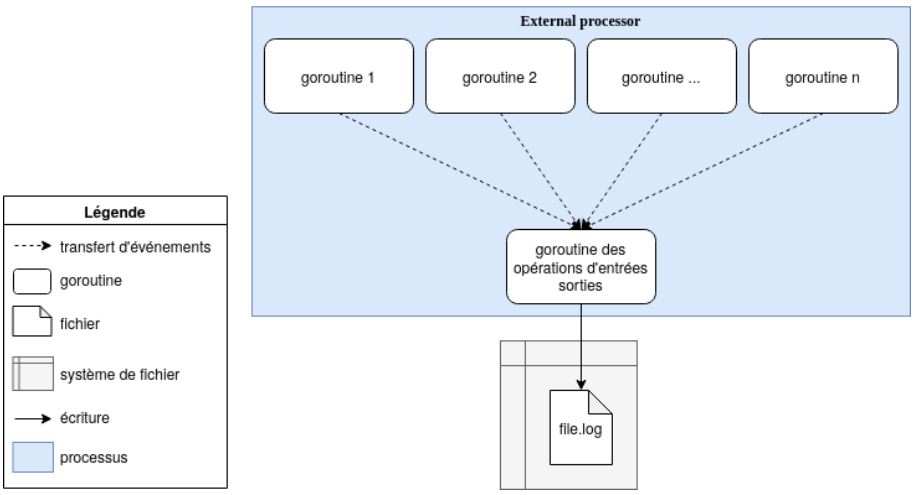


Figure 18 : Sous-système de journalisation de l’external processor

La goroutine responsable des opérations d’entrées sorties persiste sur disque chaque évènement qu’elle reçoit à l’aide de la fonction suivante :

```
function writeLog(logger, message)
    logFile ← loggerOutputFile(logger)
    writeFile(logFile, message)
end function
```

Le *logger* est une structure contenant des données et notamment le “File Descriptor” tandis que *message* est l’événement sous forme de tableau d’octet qui sera persisté sur disque.

Le système de journalisation actuellement en place permet d’implémenter la réouverture de fichier de deux manière différentes :

- changer de goroutine responsable des opérations d'entrées sorties pour les requêtes à venir;
- rouvrir le fichier et remplacer le "File Descriptor" au sein de la goroutine responsable des opérations d'entrées sorties.

La première option a l'avantage de l'atomicité par requêtes traitées, tous les logs générés par le traitement d'une même requête seront dans le même fichier. D'un autre côté, la seconde option est plus simple d'implémentation, car isolée au sein du logger. L'atomicité des logs ne justifiant pas le coût de la première option, il a été choisi avec le Product Owner, Christian, d'implémenter la seconde option.

4.2.5 - Algorithme

L'algorithme naïf de réouverture de fichier est simple dans les grandes lignes :

```
function naiveLogReopening(logger)
    logFilePath ← loggerOutputFilePath(logger)
    // openFile creates the file if it does't exist
    newLogFile ← openFile(logFilePath)
    swappedLogFile ← swapLoggerOutput(logger, newLogFile)
    closeFile(swappedLogFile)
end function
```

Néanmoins, dès lors que l'échange peut se produire en concurrence avec des écritures, l'ensemble se complexifie car il peut y avoir des situations de compétition.

Une situation de compétition (race condition en anglais) se produit lorsque plusieurs acteurs, ici des goroutine, tentent d'accéder au même moment à une ressource partagée. Dans notre cas, la ressource partagée est le "File Descriptor", le fichier de journalisation. Si une réouverture à lieu en concurrence d'une écriture d'un événement, les fonctions *SwapLoggerOutput* et *LoggerOutputFile* seront en situation de compétition. La première altère une zone mémoire tandis que la seconde la lit.

Supposons qu'un "File Descriptor" soit un octet quelconque, le logger risque de lire les 4 premiers bits de la nouvelle sortie et les 4 suivants de l'ancienne sortie. Le programme se retrouve à lire une adresse mémoire non allouée. Un signal de violation de segmentation est levé (*segmentation fault SIGSEGV*) et le programme plante. Il est donc important de s'assurer que l'échange ne perturbe pas une écriture en parallèle.

Une solution est d'utiliser un "mutex" ou n'importe quel système de synchronisation équivalent. Un "mutex", qui signifie "mutual exclusif", garantit qu'une seule goroutine à la fois puisse acquérir le verrou. Si deux goroutines essaient d'acquérir le "mutex" en parallèle, l'une d'elle sera mise en attente tant que l'autre ne le relâche pas. L'algorithme avec synchronisation via "mutex" est le suivant:

```
function logReopeningWithMutex(logger, mutex)
    logFilePath ← loggerOutputFilePath(logger)
    newLogFile ← openFile(logFilePath)
    acquireLock(mutex)
    swappedLogFile ← swapLoggerOutput(logger, newLogFile)
    releaseLock(mutex)
    closeFile(swappedLogFile)
end function
```

De même, lorsque la goroutine responsable des opérations d'entrées sorties veut persister un événement sur disque, elle doit acquérir le "mutex" **avant** de lire la sortie et le relâcher **après** l'écriture, autrement le problème persiste.

```
function writeLogWithMutex(logger, mutex, message)
    acquireLock(mutex)
    logFile ← loggerOutputFile(newLogFile)
    releaseLock(mutex)
    writeFile(logFile, message)
end function
```

Le problème de cette approche est qu'elle a un coût non négligeable. Acquérir un "mutex" est une opération potentiellement coûteuse qui doit être effectuée à chaque écriture.

Une approche alternative serait de lire et d'échanger la sortie à l'aide d'opérations atomiques. Ces dernières sont la base des sémaphores et autres mécanismes de synchronisation. Elles permettent de lire, écrire, comparer et échanger des valeurs sans contention et à moindre coût. L'algorithme simplifié devient :

```
function naiveAtomicLogReopening(logger)
    logFilePath ← loggerOutputFilePath(logger)
    newLogFile ← openFile(logFilePath)
    swappedLogFile ← atomicSwapLoggerOutput(newLogFile)
    closeFile(swappedLogFile)
end function
```

Une fois encore, pour ne pas avoir de problème de concurrence, la fonction d'écriture du logger doit effectuer une lecture atomique de la sortie avant d'écrire.

```
function atomicWriteLog(logger, mutex)
    logFile ← atomicLoggerOutputFile(logger)
    writeFile(logFile)
end function
```

Un nouveau problème survient avec cette implémentation : la fermeture prématurée du fichier. Après échange, on ferme le fichier même s' il y a une écriture en parallèle : on perd des logs.
On a deux possibilités :

- ignorer le problème et ne pas fermer le fichier de logs;
- fermer le fichier une fois qu'il n'est plus référencé.

Même si laisser le fichier ouvert est évidemment une mauvaise pratique, cela ne rajoute aucune charge de travail. Ce serait la solution idéale pour un programme destiné à ne pas évoluer et avoir un temps d'exécution court. Malheureusement, ce n'est pas notre cas et garder un fichier ouvert ne libère pas la ressource. Les systèmes basés sur UNIX maintiennent un compteur de référence par fichier. Lorsqu'un fichier est créé, son compteur vaut 1, il est incrémenté lorsqu'un processus l'ouvre et décrémenté lorsqu'il est fermé ou supprimé. Dès lors que son compteur atteindra zéro, le fichier sera réellement effacé. Si l'on ne ferme jamais le fichier, le compteur n'est pas décrémenté, il n'atteint jamais zéro et le fichier n'est pas effacé. C'est antinomique à la rotation des logs qui est là pour libérer la ressource.

Il est donc impératif d'implémenter la seconde option : s'assurer que le fichier n'est plus référencé avant de le fermer. Pour ce faire, on peut s'inspirer du mécanisme précédent et maintenir un compteur de référence de la sortie du logger. Ce dernier est incrémenté et décrémenté à l'aide d'opérations atomiques.

```
function atomicLogReopening(logger)
    logFilePath ← loggerOutputFilePath(logger)
    newLogFile ← openFile(logFilePath)
    newLogFileRC ← newFileReferenceCounter(newLogFile)
    swappedLogFileRC ← atomicSwapLoggerOutput(newLogFileRC)
    waitAndCloseFile(swappedLogFileRC)
end function
```

```
function waitAndCloseFile(logFileRC)
    while referenceCount(logFileRC) ≠ 0 do
        sleepOneSecond
    end while
    logFile ← referenceFile(logFilePath)
    closeFile(logFile)
end function
```

L'algorithme ci-dessus est complet à l'exception de la gestion des erreurs. Même s' il a été choisi, par souci de clarté, de ne pas l'ajouter jusqu'ici, cela reste une exigence non satisfaite.

Dans notre algorithme, des erreurs peuvent survenir à deux endroits:

- l'ouverture du fichier avec *OpenFile*;
- la fermeture du fichier avec *CloseFile*.

La première source d'erreur n'est pas problématique, aucun changement n'a été fait, il suffit de remonter l'erreur et le fichier de journalisation utilisé reste inchangé. À contrario, une erreur lors de la fermeture n'est pas anodine, des données peuvent être perdues.

D'après la documentation de l'appel système *close* qui est utilisé par la fonction *CloseFile*, avant la fermeture du fichier, les données en mémoire tampon sont écrites sur disques. C'est cette dernière écriture qui peut échouer et générer une erreur. Le fichier sera bien fermé et la ressource sera libérée, mais des données seront perdues. Si l'on souhaite traiter les erreurs liées à l'écriture, le manuel recommande de faire un appel système *fsync* en amont.

Après étude, dans le cadre de notre external processor, une erreur peut survenir seulement si l'espace disque venait à manquer. Les appels systèmes *fsync* n'aident pas dans cette situation, c'est pourquoi, il a été choisi de faire remonter l'erreur jusqu'à l'opérateur qui a enclenché la réouverture. Une fois prévenu, celui-ci pourra diagnostiquer le problème sous-jacent.

L'algorithme avec gestion d'erreur est le suivant :

```
function atomicLogReopening(logger)
    logFilePath ← loggerOutputFilePath(logger)
    openFileResult ← openFile(logFilePath)
    if isError(openFileResult) then
        openFileError ← extractError(openFileResult)
        return openFileError
    end if
    newLogFile ← extractFile(openFileResult)
    newLogFileRC ← newFileReferenceCounter(newLogFile)
    swappedLogFileRC ← atomicSwapLoggerOutput(newLogFileRC)
    return waitAndCloseFile(swappedLogFileRC)
end function
```

```
function waitAndCloseFile(logFileRC)
    while referenceCount(logFileRC) ≠ 0 do
        sleepOneSecond
    end while
    logFile ← referenceFile(logFilePath)
    // closeFile returns null if file is successfully closed
    closeFileError ← closeFile(logFile)
    return closeFileError
end function
```

Une fois l'algorithme implémenté en Go, il faut exposer la fonction *AtomicLogReopening* par un quelconque moyen pour qu'un opérateur puisse l'utiliser.

Traditionnellement, les réouvertures de logs sont enclenchées avec un signal POSIX, USR1 ou USR2. Un signal est une forme limitée de communication entre processus (Inter Process Communication ou IPC en anglais). Les signaux sont à sens unique, ne transmettent aucune information et n'ont pas d'acquittement, pour ces raisons, ils ne sont pas praticables dans notre cas d'usage.

Le contexte industriel de FDJ requiert qu'un opérateur utilisant la fonctionnalité doit pouvoir savoir si la réouverture est finie et si une erreur est survenue.

Une fois de plus, Envoy a été pris comme modèle, la réouverture est exposée sur l'interface d'administration HTTP de l'external processor. Dès lors qu'un opérateur souhaite effectuer une réouverture de fichier, il envoie une requête, l'external processor rouvre ses fichiers de journalisation et retourne une réponse. Si une ou plusieurs erreurs sont survenues, elles sont détaillées dans le corps de la réponse.

La fonctionnalité étant complètement implémentée, voyons le reste à faire pour compléter la tâche.

4.2.6 - Complétion de la tâche

Après avoir changé de méthodologie pour un modèle plus agile et après la phase de preuve de concept, le Product Owner a mis en place une “definition of done”. Le niveau de qualité et de travail requis pour compléter une tâche variant d’un membre de l’équipe à un autre, une définition a dû être mise en place.

La définition spécifie qu’une tâche est terminée si les fonctionnalités développées sont documentées, instrumentalisées et testées.

La documentation d’une fonctionnalité se fait à travers plusieurs prismes. Premièrement, il y a le guide utilisateur qui contient des configurations d’exemples, explique comment déployer le produit et utiliser les fonctionnalités. La documentation de design présente et justifie les différents choix architecturaux du projet. Enfin, le guide du mainteneur documente quelles sont les conventions utilisées, comment ajouter un test dans une suite ou mettre à jour les dépendances. Dans notre cas, le guide utilisateur et la documentation de design doivent être mis à jour. Le premier doit expliquer comment effectuer une rotation des fichiers de journalisation tandis que le second doit expliciter les choix algorithmiques présentés précédemment.

Une fonctionnalité est dite instrumentalisée lorsqu’elle implémente des mécanismes de télémétrie. Cela rejoint la notion d’observabilité présentée dans la section 4.2.1. Des métriques et des événements relatifs à la réouverture des logs doivent être ajoutés. Le nombre de réouvertures, d’erreurs survenues, le temps moyen par réouverture doivent être mesurés. Les métriques et autres dispositifs de télémétrie sont aussi documentés dans le guide utilisateur.

Enfin, il faut développer et intégrer des tests fonctionnels et d’exigences non fonctionnels dans la chaîne d’intégration continue. Les tests fonctionnels évaluent les cas passant et cassant, dans la limite du possible, de l’implémentation. Certains cas aux limites dont le coût d’automatisation est jugé trop élevé tels que le manque d’espace disque ne sont pas intégrés dans la chaîne d’intégration continue. Les tests d’exigences non fonctionnels évaluent les aspects transversaux de la solution telles que la sécurité, l’instrumentalisation et la performance.

Après avoir réalisé les éléments constitutifs ci-dessus, le code peut être revu par des membres de l’équipe. Une revue par les pairs peut mettre en évidence des problèmes et permettre d'apporter des suggestions pour améliorer la qualité du code. Cela peut inclure l'élimination des commentaires obsolètes, l'amélioration de la structure du code ou l'optimisation des performances si nécessaire.

Une fois le code validé par les pairs, les changements sont intégrés au dépôt et la tâche sera clôturée à la prochaine réunion quotidienne.

Pour conclure, implémenter correctement une rotation de fichier de logs dans un contexte industriel n’est pas une tâche triviale. Comprendre le fonctionnement des systèmes de fichiers, les signaux des processus et la programmation concurrentielle sont requis pour concevoir et implémenter avec succès cette fonctionnalité. Néanmoins, en suivant la bonne méthodologie, il est possible de réussir, et ce, même avec des lacunes. Les multiples failles dans mes implémentations ont pu être décelées grâce aux tests et aux revues de codes avec l’équipe. Enfin, étudier et faire de la rétroingénierie des implémentations d’autres produits m’a permis de découvrir de nouvelles approches avec différents points forts et faibles.

5 - Conclusion

Pour finir, une rétrospective sur la mission s'impose. La solution d'API Gateway Open Source livrée en 2022 est aujourd'hui déployée en production et répond aux besoins fonctionnels, de performance et de sécurité des parties prenantes. Le Tech-Lab a su, en un an et demi, satisfaire aux exigences, et ce, tout en garantissant l'objectivité, la traçabilité et la reproductibilité de nos choix.

En qualité de membre de l'équipe, les différents projets auxquels j'ai contribué, m'ont appris à travailler dans un contexte industriel. Nos livrables ne répondent pas seulement aux besoins fonctionnels, mais ont un certain niveau de qualité. Ainsi, j'ai acquis les compétences nécessaires pour élaborer une solution performante, facilement maintenable, sécurisée, bien documentée et soumise à des tests automatiques.

Je retire de ces trois années d'apprentissage, combinant expérience pratique à la FDJ et formation théorique à l'IMERIR, plusieurs enseignements clés. Parmi eux, se distinguent mon acculturation aux tests, en particulier aux tests de performance, la mise en place d'une méthodologie de projet, l'approche agile dans le travail, et ma capacité à présenter mes réalisations en fin d'itération.

Je considère également avoir fait des progrès sur le plan relationnel. Le travail en équipe, le partage des connaissances, les revues de code, la diversité et la confrontation des points de vue, la communication avec des équipes externes, et bien d'autres savoirs qui m'étaient jusqu'alors inconnus, m'ont fait évoluer favorablement.

J'ai eu l'opportunité de découvrir à la FDJ d'autres domaines informatiques telles que la performance, la data et la production. Mes interactions variées au sein de l'entreprise ont élargi ma perspective initialement restreinte de ce que représente ce vaste domaine qu'est l'informatique.

En somme, je considère avoir bénéficié d'une formation de qualité, durant laquelle on ne m'a pas seulement confié des tâches, mais enseigné les compétences et les méthodologies nécessaires pour travailler de manière autonome et efficace à long terme.

Aujourd'hui je me sens assez confiant et prêt à entreprendre. Je projette dans un avenir proche de travailler sous le statut d'auto-entrepreneur mais également sur des missions ponctuelles et ainsi continuer à élargir mes compétences techniques, gagner en autonomie et tester mon appétence pour le management.

6 - Glossaire

- ABU: Acceleration Business Unit
- ADC: Application Delivery Controller
- ANJ: Autorité Nationale des Jeux
- API: Application Programming Interface
- APM: Application and Performance Management
- BU: Business Unit
- CD: Continuous Deployment
- CDC: Cahier Des Charges
- CI: Continuous Integration
- CIP: Centre d'Ingénierie de Performance
- CNIL: Commission Nationale de l'Informatique et des Libertés
- CPU: Central Processing Unit
- DSI: Direction des Systèmes d'Informations
- DT: Direction Technique
- FDJ: Française Des Jeux
- HTTP: Hyper Text Transfer Protocol
- IPC: Inter Process Communication
- Idp: Identity Provider
- JSON: JavaScript Object Notation
- JWT: JSON Web Token
- Mock: bouchon est une entité (instance d'objet, service ou programme) qui n'effectue aucun traitement et retourne toujours le même résultat, utilisé pour remplacer une autre fonctionnalité.
- OCI: Open Container Initiative
- OIDC: OpenID Connect
- PMU: Pari Mutuel Urbain
- PO: Product Owner:
- POC: Proof Of Concept, preuve de concept
- POSIX: Portable Operating System Interface
- SLNLN: Société de la Loterie Nationale et du Loto National
- Sprint: itération de développement
- TLS: Transport Layer Security
- UBFT: Union des Blessés de la Face et de la Tête
- WLA: World Lottery Association

7 - Bibliographie

Groupe FDJ - site institutionnel de FDJ et de sa fondation d'entreprise. (s. d.). https://www.groupefdj.com/

Accueil. (s. d.). ANJ. https://anj.fr/

Contributeurs aux projets Wikimedia. (2023a). Loto (jeu de la Française des jeux). *fr.wikipedia.org*. https://fr.wikipedia.org/wiki/Loto_(jeu_de_la_Fran%C3%A7aise_des_jeux)

Contributeurs aux projets Wikimedia. (2023b). Couche réseau. *fr.wikipedia.org*. https://fr.wikipedia.org/wiki/Couche_r%C3%A9seau#

Contributeurs aux projets Wikimedia. (2023c). Cahier des charges. *fr.wikipedia.org*. https://fr.wikipedia.org/wiki/Cahier_des_charges

Contributeurs aux projets Wikimedia. (2023d). Preuve de concept. *fr.wikipedia.org*. https://fr.wikipedia.org/wiki/Preuve_de_concept

Contributeurs aux projets Wikimedia. (2023e). Française des Jeux. *fr.wikipedia.org*. https://fr.wikipedia.org/wiki/Fran%C3%A7aise_des_jeux

CrowdStrike. (2023, 2 mai). *Qu'est-ce que la rotation des logs ? | CrowdStrike*. crowdstrike.fr. https://www.crowdstrike.fr/cybersecurity-101/observability/log-rotation/

Groupe FDJ - site institutionnel de FDJ et de sa fondation d'entreprise. (s. d.). https://www.groupefdj.com/

Le Parisien. (2020, 5 février). *De la loterie des « gueules cassées » à la privatisation : l'histoire méconnue de la Française des . . .* [Vidéo]. YouTube. https://www.youtube.com/watch?v=hbfuJAOxPaw

Learn about observability | Honeycomb. (s. d.).
https://docs.honeycomb.io/concepts/learning-about-observability/

Les gueules cassées - UBFT. (s. d.). Les Gueules Cassées.
https://www.gueules-cassees.asso.fr/

Life of a request — envoy 1.28.0-DeV-F837d7 documentation. (s. d.).
https://www.envoyproxy.io/docs/envoy/latest/intro/life_of_a_request

Rehkopf, P. M. (s. d.). *User Stories | Exemples et modèle | Atlassian*. Atlassian.
https://www.atlassian.com/fr/agile/project-management/user-stories

The open source definition. (2023, 22 février). Open Source Initiative.
https://opensource.org/osd/

8 - Liste des illustrations

9 - Liste des tableaux

10 - Annexes

10.1 - Axes et critères de notation API Gateway

Axe	Critère	Pondération axe	Pondération critères
Gateway Provider		6	
	Dépendances		3
	Performance brute		1
	Performance à la cible		1
	Capacité d'évolution (« scalability »)		1
	Disponibilité		2
	Langages de développement		2
	Extensibilité		2
	« Rate Limiting »		1
	Support d'OpenAPI		1
	Supervision unifiée		1
	Métrologie unifiée		1
	Intégration OpenID Connect		1
	Autorisation d'accès		1
	Support des IT providers		1
	Prise en compte des modifications d'API		1
Operations ready by design		3	
	Logs d'accès/erreurs		1
	Distributed Tracing		0
	Supervision technique		
	Métrologie technique		1
	Déploiement à chaud		1

Security by design		4	
	Confidentialité des échanges de données sensibles		2
	Confidentialité de la persistance des données sensibles		2
	Protection des accès d'administration		1
	Surface d'exposition minimale		2
Resilient by design		2	
	Résistance sur sollicitation excessive		0
	Limitation de l'utilisation des ressources		1
	Lissage du trafic et circuit breaker		1
	Résistance en cas de défaillances		0
Roadmap, support et licenses		2	
	Licenses		3
	Support communautaire		2
	Support complémentaire		1
	Modèle de coûts du support		0
	Roadmap		2

10.2 - Exemples critères mission API Gateway

Capacité d'évolution (« scalability »)

Objectif

Ce critère sert à mesurer avec quelle facilité la solution sait gérer un trafic dans une fourchette de 100 à 3000 reqs/s. On s'assurera que cette capacité d'adaptation au trafic se fasse en utilisant de manière relativement linéaire des ressources physiques.

Méthode d'évaluation

De nouveau, on analysera des tests de performances existants et on notera séparément l'évolutivité horizontale et verticale, de l'évolutivité des dépendances : la note finale sera alors la moyenne de ces notes (arrondi bas).

Pour l'évolutivité verticale, on part d'une note de départ de 10 :

- On utilise le nombre de requêtes/s traités avec un nombre de cœurs le plus proche possible de 2 cœurs et 4 cœurs
- Si pour la seconde mesure, on ne dispose que d'une mesure pour plus de 8 cœurs, la note finale pour l'évolutivité verticale sera dégradée de 2 points
- Si on ne dispose d'aucune mesure ou d'une unique mesure, la note est de 0 (impossible de se prononcer)
- Sinon on calcule le nombre de requêtes/s traités par cœur dans les deux cas :
    - Si la dégradation des performances est > à 10%, note de -1
    - Sinon la note est dégradée de 1 par % de dégradation (0 si 10% de dégradation)

Pour l'évolutivité horizontale, on utilise le même ~~principe (~~note de départ à 10) appliqué à un nombre d'instances :

- On utilise le nombre de requêtes/s traitées par un nombre d'instances croissants avec au moins 1 mesure de >=2 instances et une seconde >= 4 instances.
- Dans la mesure du possible, on sélectionne pour la seconde mesure un test de performance utilisant un nombre d'instances le plus grand possible sans dépasser 5000 requêtes/s traitées
- Si besoin, pour la seconde mesure, on pourra utiliser une mesure >=4 instances mais la note finale pour l'évolutivité horizontale sera dégradée de 2 points
- Si on ne dispose d'aucune mesure ou d'une unique mesure, la note est de 0 (impossible de se prononcer)
- Sinon on calcule le nombre de requêtes/s traités par instance dans les deux cas :
    - Si la dégradation des performances est > à 10%, note de -1
    - Sinon la note est dégradée de 1 par % de dégradation (0 si 10% de dégradation)

Pour les dépendances :

- En l'absence de dépendances, la note sera de 10
- Pour chaque dépendance, on analysera le mécanisme d'évolutivité utilisé dans les tests utilisés ci-dessus :
    - Si l'évolutivité verticale a été utilisée, il faut vérifier le rapport par cœur pour en déduire la dégradation éventuelle.
    - Si l'évolutivité horizontale a été utilisée, il faut vérifier le rapport par instance pour obtenir la dégradation
    - En fonction de la dégradation observée, abaissement de la note :
        - 0 si dégradation < 5%
        - -1 si dégradation >= 5 et < 10%
        - -2 sinon

Pour la note finale, on retiendra alors la moyenne des 3 notes obtenues.

Disponibilité

Objectif

L'objectif à atteindre est que la solution n'expérimente pas plus 0.01% d'erreurs de traitement en cas de perte d'un composant de la solution.
A titre de référence, le tableau suivant donne une indication du temps d'indisponibilité maximale du service acceptable si une panne déclenche une interruption de service :

Période de référence	Temps d'interruption maximum
1 journée	9s
1 semaine	1 minute
1 mois	4min19s
6 mois	26min16s
1 an	52min33s

Méthode d'évaluation

Pour cette évaluation, il faudra utiliser la documentation des composants du produit. Pour chacun des composants du produit, il convient d'évaluer le dispositif à mettre en œuvre pour assurer la perte de ce composant :

- Si la disponibilité du composant peut être assurée par l'utilisation d'un simple LB frontal, la note retenue sera de 10
- Si le composant utilise un mécanisme de disponibilité intégrée, il conviendra d'évaluer le temps d'indisponibilité maximum que ce dispositif requiert pour effectuer une bascule complète et automatisée : la note de 10 sera dégradée de 1 points à chaque minute requise.
- Si le composant requiert l'utilisation d'un mécanisme de disponibilité externe au produit autre qu'un simple load-balancer (ex. cluster OS, disque partagé), on effectue le calcul du temps d'indisponibilité en dégradant de 2 points par minute requise (au lieu de 1 car deux fois plus complexes à réaliser).

Ensuite, il conviendra d'établir une moyenne pondérée des notes de chaque composant :

- Le facteur de pondération de chaque composant est établi sur la base de l'impact sur la disponibilité globale du système le temps que le dispositif de secours soit actif
- Il faut donc estimer le % de trafic perturbé le temps que le dispositif de secours soit actif : le facteur de pondération sera ce % divisé par 10
- Ainsi si 100% du trafic est perturbé, le poids de ce composant dans la note final sera de 10
- Si 0% trafic perturbé, alors le poids est de 1

10.3 - Grille de notation Krakend

Regroupement	Critère	Pondération Axe	Pondération Critères	Notation	Seuil éliminatoire	Résultat
Gateway Provider SPI		6		79		28
	Dépendances		3	10	0	30
	Performance brute		1	10	0	10
	Performance à la cible		1	0	0	0
	Capacité d'évolution (« scalability »)		1	10	0	10
	Disponibilité		2	10	0	20
	Langages de développement		2	10	0	20
	Extensibilité		2	10	0	20
	« Rate Limiting »		1	7	0	7
	Support d'OpenAPI		1	7	0	7
	Supervision unifiée		1	5	0	5
	Métrologie unifiée		1	8	0	8
	Intégration OpenID Connect		1	5	0	5
	Autorisations d'accès		1	6	0	6
	Support des IT Providers		1	6	0	6
Operations ready by design	Prise en compte des modifications d'API	3	1	4	0	4
	Logs d'accès/erreurs		1	80 8	0	14 8
	Distributed Tracing		0	0	0	0
	Supervision technique		1	10	0	10
	Métrologie technique		1	4	0	4
	Déploiement à chaud		1	10	0	10
Security by design		4		89		21
	Confidentialité des échanges de données sensibles		2	7	0	14
	Confidentialité de la persistance des données sensibles		2	10	0	20
	Protection des accès d'administration		1	8	0	8
	Surface d'exposition minimale		2	10	0	20
Resilient by design		2		95		11
	Resistance sur sollicitation excessive		0	0	0	0
	Limitation de l'utilisation des ressources		1	10	0	10
	Lissage du trafic et circuit breaker		1	9	0	9
	Résistance en cas de défaillances		0	10	0	0
Roadmap, support et licences		2		61		7
	Licences		3	6	0	18
	Support communautaire		2	10	0	20
	Support complémentaire		1	5	0	5
	Modèle de coût du support		0	0	0	0
	Roadmap produit		2	3	0	6
Total Produit			404			81

Porteurs C. Boitel
Evaluateurs F. Torres
JL Leboutet

10.4 - Grille de notation Kong

Regroupement	Critère	Pondération Axe	Pondération Critères	Notation	Seuil éliminatoire	Résultat
Gateway Provider SPI		6		71	0	25
	Dépendances		3	10	0	30
	Performance brute		1	10	0	10
	Performance à la cible		1	10	0	10
	Capacité d'évolution (« scalability »)		1	10	0	10
	Disponibilité		2	10	0	20
	Langages de développement		2	4	0	8
	Extensibilité		2	8	0	16
	« Rate Limiting »		1	7	0	7
	Support d'OpenAPI		1	4	0	4
	Supervision unifiée		1	5	0	5
	Métrologie unifiée		1	1	0	1
	Intégration OpenID Connect		1	4	0	4
	Autorisations d'accès		1	1	0	1
	Support des IT Providers		1	10	0	10
	Prise en compte des modifications d'API		1	6	0	6
Operations ready by design		3		88		15
	Logs d'accès/erreurs		1	8	0	8
	Distributed Tracing		0	10	0	0
	Supervision technique		1	10	0	10
	Métrologie technique		1	7	0	7
	Déploiement à chaud		1	10	0	10
Security by design		4		80		19
	Confidentialité des échanges de données sensibles		2	10	0	20
	Confidentialité de la persistance des données sensibles		2	10	0	20
	Protection des accès d'administration		1	10	0	10
	Surface d'exposition minimale		2	3	0	6
Resilient by design		2		60		7
	Resistance sur sollicitation excessive		0	10	0	0
	Limitation de l'utilisation des ressources		1	6	0	6
	Lissage du trafic et circuit breaker		1	6	0	6
	Résistance en cas de défaillances		0	10	0	0
Roadmap, support et licences		2		79		9
	Licences		3	7	0	21
	Support communautaire		2	10	0	20
	Support complémentaire		1	10	0	10
	Modèle de coût du support		0	10	0	0
	Roadmap produit		2	6	0	12
Total Produit				377		76

10.5 - Grille de notation Gloo

Regroupement	Critère	Pondération Axe	Pondération Critères	Notation	éliminatoire	Résultat	commentaires
Gateway Provider SPI		6		79		28	
	Dépendances		3	10	0	30	- note de 10 si déploiement en container -4 (-6) si utilisation hors container
	Performance brute		1	10	0	10	
	Performance à la cible		1	10	0	10	
	Capacité d'évolution (« scalability »)		1	5	0	5	
Operations ready by design		3		90		16	
	Disponibilité		2	8	0	16	La disponibilité est assurée par le déploiement en container, en dehors de ce mode de déploiement, il faut 1B pour Gloo, et déployer Vault et Consul en "cluster applicatif"
	Langages de développement		2	7	0	14	
	Extensibilité		2	10	0	20	
	« Rate Limiting »		1	8	0	8	
	Support d'OpenAPI		1	7	0	7	
	Supervision unifiée		1	5	0	5	
	Métrologie unifiée		1	10	0	10	
	Intégration OpenID Connect		1	5	0	5	
	Autorisations d'accès		1	6	0	6	
	Support des IT Providers		1	7	0	7	
	Prise en compte des modifications d'API		1	4	0	4	
Security by design		4		86		20	
	Logs d'accès/erreurs		1	9	0	9	
	Distributed Tracing		0	10	0	0	
	Supervision technique		1	10	0	10	
	Métrologie technique		1	10	0	10	
	Déploiement à chaud		1	7	0	7	
	Confidentialité des échanges de données sensibles		2	10	0	20	
	Confidentialité de la persistance des données sensibles		2	10	0	20	
	Protection des accès d'administration		1	0	0	0	
	Surface d'exposition minimale		2	10	0	20	
Resilient by design		2		70		8	
	Résistance sur sollicitation excessive		0	10	0	0	
	Limitation de l'utilisation des ressources		1	8	0	8	
	Lissage du trafic et circuit breaker		1	6	0	6	
	Résistance en cas de défaillances		0	10	0	0	
Roadmap, support et licences		2		64		8	
	Licences		3	6	0	18	
	Support communautaire		2	10	0	20	
	Support complémentaire		1	1	0	1	
	Modèle de coût du support		0	10	0	0	

Gloo vs

Porteur JL Lebouteux
Evaluateurs L. Scalisi

10.6 - Grille de notation Envoy

Regroupement	Critère	Pondération Axe	Pondération Critères	Notation	Seuil éliminatoire	Résultat	commentaires
Gateway Provider SPI		6		78		27	
	Dépendances		3	10	0	30	
	Performance brute		1	10	0	10	
	Performance à la cible		1	10	0	10	
	Capacité d'évolution (« scalability »)		1	5	0	5	
	Disponibilité		2	10	0	20	
	Langages de développement		2	6	0	12	
	Extensibilité		2	10	0	20	
	« Rate Limiting »		1	8	0	8	
	Support d'OpenAPI		1	0	0	0	
	Supervision unifiée		1	5	0	5	
	Métrologie unifiée		1	10	0	10	
	Intégration OpenID Connect		1	5	0	5	
	Autorisations d'accès		1	6	0	6	
	Support des IT Providers		1	10	0	10	
	Prise en compte des modifications d'A		1	4	0	4	
Operations ready by design		3		98		17	
	Logs d'accès/erreurs		1	9	0	9	
	Distributed Tracing		0	10	0	0	
	Supervision technique		1	10	0	10	
	Métrologie technique		1	10	0	10	
	Déploiement à chaud		1	10	0	10	
Security by design		4		96		23	
	Confidentialité des échanges de donn		2	10	0	20	
	Confidentialité de la persistance des d		2	10	0	20	
	Protection des accès d'administration		1	7	0	7	
	Surface d'exposition minimale		2	10	0	20	
Resilient by design		2		70		8	
	Résistance sur sollicitation excessive		0	10	0	0	
	Limitation de l'utilisation des ressourc		1	8	0	8	
	Lissage du trafic et circuit breaker		1	6	0	6	
	Résistance en cas de défaillances		0	10	0	0	
Roadmap, support et licences		2		66		8	
	Licences		3	6	0	18	
	Support communautaire		2	10	0	20	
	Support complémentaire		1	1	0	1	
	Modèle de coût du support		0	10	0	0	
	Roadmap produit		2	7	0	14	
Total Produit				407		83	

Gloo vs

Porteur JL Leboutet
Evaluateurs L Scalisi

10.7 - Grille de notation Express Gateway

Regroupement	Critère	Pondération Axe	Pondération Critères	Notation	éliminatoire	Résultat
Gateway Provider SPI		6		72		25
	Dépendances		3	10	0	30
	Performance brute		1	2	0	2
	Performance à la cible		1	10	0	10
	Capacité d'évolution (« scalability »)		1	5	0	5
	Disponibilité		2	10	0	20
	Langages de développement		2	10	0	20
	Extensibilité		2	9	0	18
	« Rate Limiting »		1	10	0	10
	Support d'OpenAPI		1	7	0	7
	Supervision unifiée		1	7	0	7
	Métrologie unifiée		1	5	0	5
	Intégration OpenID Connect		1	2	0	2
	Autorisations d'accès		1	0	0	0
	Support des IT Providers		1	4	0	4
	Prise en compte des modifications d'API		1	4	0	4
Operations ready by design		3		53		9
	Logs d'accès/erreurs		1	7	0	7
	Distributed Tracing		0	10	0	0
	Supervision technique		1	7	0	7
	Métrologie technique		1	0	0	0
	Déploiement à chaud		1	7	0	7
Security by design		4		67		16
	Confidentialité des échanges de données sensibles		2	5	0	10
	Confidentialité de la persistance des données sensibles		2	10	0	20
	Protection des accès d'administration		1	9	0	9
	Surface d'exposition minimale		2	4	0	8
Resilient by design		2		20		2
	Resistance sur sollicitation excessive		0	10	0	0
	Limitation de l'utilisation des ressources		1	0	0	0
	Lissage du trafic et circuit breaker		1	4	0	4
	Résistance en cas de défaillances		0	10	0	0
Roadmap, support et licences		2		-1		-1
	Licences		3	6	0	18
	Support communautaire		2	-1	0	-1
	Support complémentaire		1	0	0	0
	Modèle de coût du support		0	10	0	0
	Roadmap produit		2	5	0	10
Total Produit				-1		-1

Porteur JL Leboutet
Evaluateurs F. Torres

10.8 - Grille de notation Spring Cloud Gateway

Regroupement	Critère	Pondération Axe	Pondération Critères	Notation	éliminatoire	Résultat
Gateway Provider SPI		6		72		25
	Dépendances		3	10	0	30
	Performance brute		1	10	0	10
	Performance à la cible		1	0	0	0
	Capacité d'évolution (« scalability »)		1	5	0	5
	Disponibilité		2	10	0	20
	Langages de développement		2	7	0	14
	Extensibilité		2	9	0	18
	« Rate Limiting »		1	7	0	7
	Support d'OpenAPI		1	7	0	7
	Supervision unifiée		1	5	0	5
	Métrologie unifiée		1	10	0	10
	Intégration OpenID Connect		1	6	0	6
	Autorisations d'accès		1	0	0	0
	Support des IT Providers		1	2	0	2
	Prise en compte des modifications d'API		1	10	0	10
Operations ready by design		3		88		15
	Logs d'accès/erreurs		1	9	0	9
	Distributed Tracing		0	10	0	0
	Supervision technique		1	10	0	10
	Métrologie technique		1	6	0	6
	Déploiement à chaud		1	10	0	10
Security by design		4		71		17
	Confidentialité des échanges de données sensibles		2	8	0	16
	Confidentialité de la persistance des données sensibles		2	10	0	20
	Protection des accès d'administration		1	8	0	8
	Surface d'exposition minimale		2	3	0	6
Resilient by design		2		75		9
	Résistance sur sollicitation excessive		0	10	0	0
	Limitation de l'utilisation des ressources		1	10	0	10
	Lissage du trafic et circuit breaker		1	5	0	5
	Résistance en cas de défaillances		0	10	0	0
Roadmap, support et licences		2		71		8
	Licences		3	6	0	18
	Support communautaire		2	10	0	20
	Support complémentaire		1	5	0	5
	Modèle de coût du support		0	10	0	0
	Roadmap produit		2	7	0	14
Total Produit			377			75

Porteur C. Boitel
Evaluateurs F. Torres
Revue 31-Jul

10.9 - Grille de notation Gravitee

Regroupement	Critère	Pondération Axe	Pondération Critères	Notation	éliminatoire	Résultat
Gateway Provider SPI		6		73		26
	Dépendances		3	6	0	18
	Performance brute		1	0	0	0
	Performance à la cible		1	8	0	8
	Capacité d'évolution (« scalability »)		1	0	0	0
	Disponibilité		2	9	0	18
	Langages de développement		2	10	0	20
	Extensibilité		2	10	0	20
	« Rate Limiting »		1	5	0	5
	Support d'OpenAPI		1	7	0	7
	Supervision unifiée		1	10	0	10
	Métrologie unifiée		1	10	0	10
	Intégration OpenID Connect		1	6	0	6
	Autorisations d'accès		1	10	0	10
	Support des IT Providers		1	6	0	6
	Prise en compte des modifications d'API		1	7	0	7
Operations ready by design		3		68	0	12
	Logs d'accès/erreurs		1	8	0	8
	Distributed Tracing		0	10	0	0
	Supervision technique		1	10	0	10
	Métrologie technique		1	3	0	3
	Déploiement à chaud		1	6	0	6
Security by design		4		93		22
	Confidentialité des échanges de données sensibles		2	10	0	20
	Confidentialité de la persistance des données sensibles		2	10	0	20
	Protection des accès d'administration		1	9	0	9
	Surface d'exposition minimale		2	8	0	16
Resilient by design		2		75	0	9
	Resistance sur sollicitation excessive		0	10	0	0
	Limitation de l'utilisation des ressources		1	8	0	8
	Lissage du trafic et circuit breaker		1	7	0	7
	Résistance en cas de défaillances		0	10	0	0
Roadmap, support et licences		2		66	0	8
	Licences		3	6	0	18
	Support communautaire		2	10	0	20
	Support complémentaire		1	1	0	1
	Modèle de coût du support		0	10	0	0
	Roadmap produit		2	7	0	14
Total Produit				374		76

Porteur F. Mihalic
Evaluateurs

10.10 - Grille de notation WSO2

Regroupement	Critère	Pondération Axe	Pondération Critères	Notation	éliminatoire	Résultat
Gateway Provider SPI		6		46		16
	Dépendances		3	0	0	0
	Performance brute		1	10	0	10
	Performance à la cible		1	0	0	0
	Capacité d'évolution (« scalability »)		1	0	0	0
	Disponibilité		2	7	0	14
	Langages de développement		2	8	0	16
	Extensibilité		2	7	0	14
	« Rate Limiting »		1	8	0	8
	Support d'OpenAPI		1	5	0	5
	Supervision unifiée		1	4	0	4
	Métrologie unifiée		1	4	0	4
	Intégration OpenID Connect		1	0	0	0
	Autorisations d'accès		1	0	0	0
	Support des IT Providers		1	6	0	6
	Prise en compte des modifications d'API		1	10	0	10
Operations ready by design		3		53		9
	Logs d'accès/erreurs		1	8	0	8
	Distributed Tracing		0	10	0	0
	Supervision technique		1	4	0	4
	Métrologie technique		1	5	0	5
	Déploiement à chaud		1	4	0	4
Security by design		4		80		19
	Confidentialité des échanges de données sensibles		2	10	0	20
	Confidentialité de la persistance des données sensibles		2	10	0	20
	Protection des accès d'administration		1	10	0	10
	Surface d'exposition minimale		2	3	0	6
Resilient by design		2		50		6
	Résistance sur sollicitation excessive		0	10	0	0
	Limitation de l'utilisation des ressources		1	5	0	5
	Lissage du trafic et circuit breaker		1	5	0	5
	Résistance en cas de défaillances		0	10	0	0
Roadmap, support et licences		2		63		7
	Licences		3	5	0	15
	Support communautaire		2	10	0	20
	Support complémentaire		1	5	0	5
	Modèle de coût du support		0	10	0	0
	Roadmap produit		2	5	0	10
Total Produit			291			57

Porteur F. Mihalic
Evaluateurs

10.11 - Grille de notation AWS API Gateway

Regroupement	Critère	Pondération Axe	Pondération Critères	Notation	éliminatoire	Résultat
Gateway Provider SPI		6		100		35
	Dépendances		3	10	0	30
	Performance brute		1	10	0	10
	Performance à la cible		1	10	0	10
	Capacité d'évolution (« scalability »)		1	10	0	10
	Disponibilité		2	10	0	20
	Langages de développement		2	10	0	20
	Extensibilité		2	10	0	20
	« Rate Limiting »		1	10	0	10
	Support d'OpenAPI		1	10	0	10
	Supervision unifiée		1	10	0	10
	Métrologie unifiée		1	10	0	10
	Intégration OpenID Connect		1	10	0	10
	Autorisations d'accès		1	10	0	10
	Support des IT Providers		1	10	0	10
	Prise en compte des modifications d'API		1	10	0	10
Operations ready by design		3		100		18
	Logs d'accès/erreurs		1	10	0	10
	Distributed Tracing		0	10	0	0
	Supervision technique		1	10	0	10
	Métrologie technique		1	10	0	10
	Déploiement à chaud		1	10	0	10
Security by design		4		100		24
	Confidentialité des échanges de données sensibles		2	10	0	20
	Confidentialité de la persistance des données sensibles		2	10	0	20
	Protection des accès d'administration		1	10	0	10
	Surface d'exposition minimale		2	10	0	20
Resilient by design		2		100		12
	Résistance sur sollicitation excessive		0	10	0	0
	Limitation de l'utilisation des ressources		1	10	0	10
	Lissage du trafic et circuit breaker		1	10	0	10
	Résistance en cas de défaillances		0	10	0	0
Roadmap, support et licences		2		-1		-1
	Licences		3	-1	0	-1
	Support communautaire		2	10	0	20
	Support complémentaire		1	10	0	10
	Modèle de coût du support		0	10	0	0
	Roadmap produit		2	10	0	20
Total Produit				-1		-1

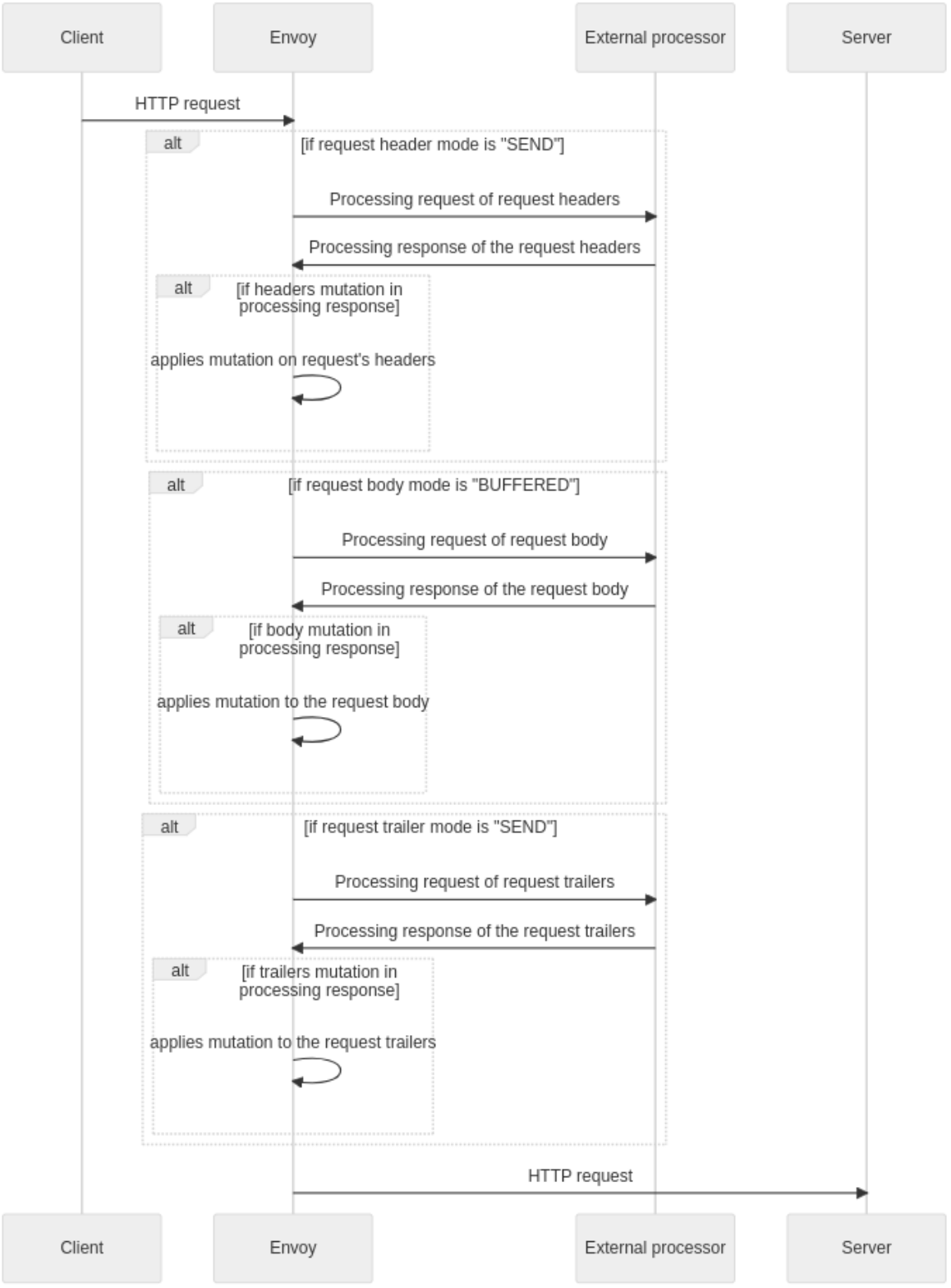
10.12 - Grille de notation Apigee Cloud

Regroupement	Critère	Pondération Axe	Pondération Critères	Notation	éliminatoire	Résultat
Gateway Provider SPI		6		100		35
	Dépendances		3	10	0	30
	Performance brute		1	10	0	10
	Performance à la cible		1	10	0	10
	Capacité d'évolution (« scalability »)		1	10	0	10
	Disponibilité		2	10	0	20
	Langages de développement		2	10	0	20
	Extensibilité		2	10	0	20
	« Rate Limiting »		1	10	0	10
	Support d'OpenAPI		1	10	0	10
	Supervision unifiée		1	10	0	10
	Métrologie unifiée		1	10	0	10
	Intégration OpenID Connect		1	10	0	10
	Autorisations d'accès		1	10	0	10
	Support des IT Providers		1	10	0	10
	Prise en compte des modifications d'API		1	10	0	10
Operations ready by design		3		100		18
	Logs d'accès/erreurs		1	10	0	10
	Distributed Tracing		0	10	0	0
	Supervision technique		1	10	0	10
	Métrologie technique		1	10	0	10
	Déploiement à chaud		1	10	0	10
Security by design		4		100		24
	Confidentialité des échanges de données sensibles		2	10	0	20
	Confidentialité de la persistance des données sensibles		2	10	0	20
	Protection des accès d'administration		1	10	0	10
	Surface d'exposition minimale		2	10	0	20
Resilient by design		2		100		12
	Resistance sur sollicitation excessive		0	10	0	0
	Limitation de l'utilisation des ressources		1	10	0	10
	Lissage du trafic et circuit breaker		1	10	0	10
	Résistance en cas de défaillances		0	10	0	0
Roadmap, support et licences		2		-1		-1
	Licences		3	-1	0	-1
	Support communautaire		2	10	0	20
	Support complémentaire		1	10	0	10
	Modèle de coût du support		0	10	0	0
	Roadmap produit		2	10	0	20
Total Produit				-1		-1

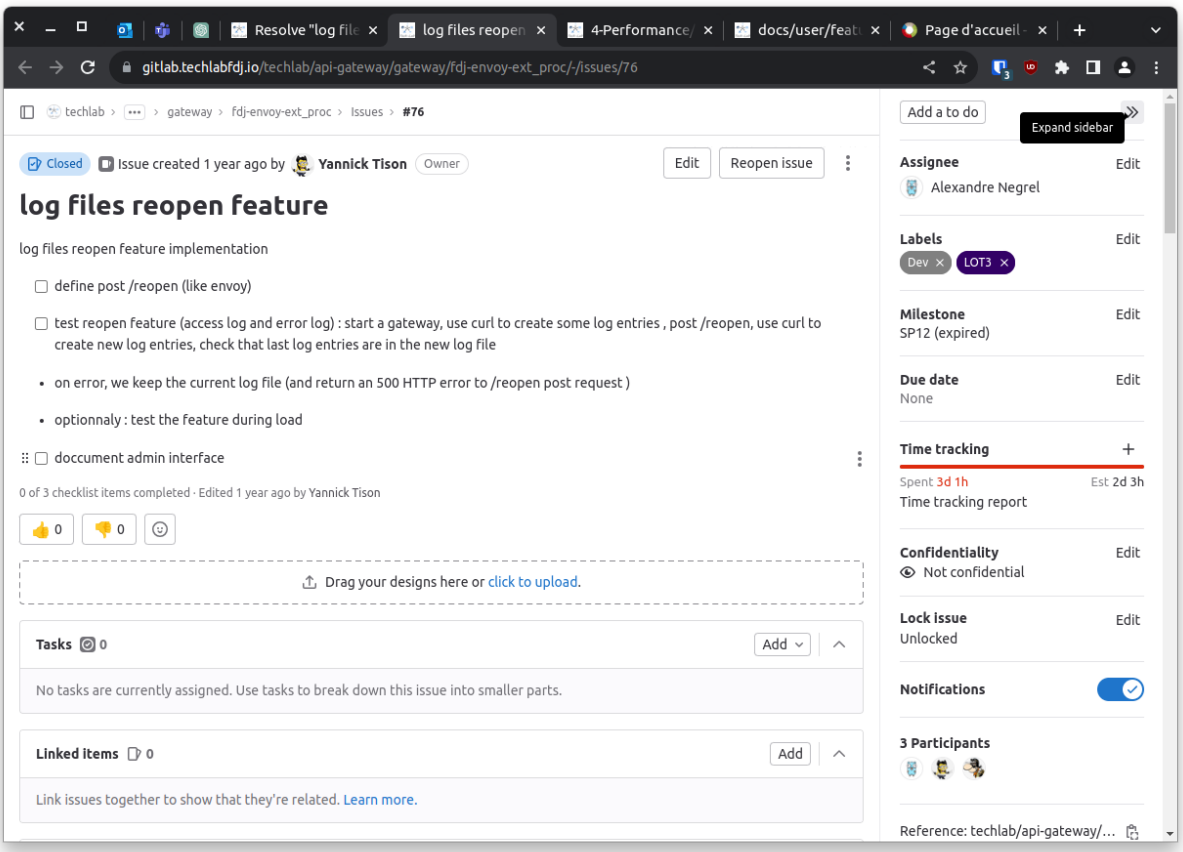
10.13 - Grille de notation Traefik

Regroupement	Critère	Pondération Axe	Pondération Critères	Notation	éliminatoire	Résultat
Gateway Provider SPI		6		-1		-1
	Dépendances		3	10	0	30
	Performance brute		1	10	0	10
	Performance à la cible		1	10	0	10
	Capacité d'évolution (« scalability »)		1	5	0	5
	Disponibilité		2	10	0	20
	Langages de développement		2	10	0	20
	Extensibilité		2	4	0	8
	« Rate Limiting »		1	10	0	10
	Support d'OpenAPI		1	4	0	4
	Supervision unifiée		1	5	0	5
	Métrologie unifiée		1	3	0	3
	Intégration OpenID Connect		1	-1	0	-1
	Autorisations d'accès		1	-1	0	-1
	Support des IT Providers		1	10	0	10
	Prise en compte des modifications d'API		1	10	0	10
Operations ready by design		3		93		16
	Logs d'accès/erreurs		1	8	0	8
	Distributed Tracing		0	10	0	0
	Supervision technique		1	9	0	9
	Métrologie technique		1	10	0	10
	Déploiement à chaud		1	10	0	10
Security by design		4		86		20
	Confidentialité des échanges de données sensibles		2	10	0	20
	Confidentialité de la persistance des données sensibles		2	10	0	20
	Protection des accès d'administration		1	4	0	4
	Surface d'exposition minimale		2	8	0	16
Resilient by design		2		65		8
	Resistance sur sollicitation excessive		0	10	0	0
	Limitation de l'utilisation des ressources		1	4	0	4
	Lissage du trafic et circuit breaker		1	9	0	9
	Résistance en cas de défaillances		0	10	0	0
Roadmap, support et licences		2		85		10
	Licences		3	10	0	30
	Support communautaire		2	10	0	20
	Support complémentaire		1	2	0	2
	Modèle de coût du support		0	10	0	0
	Roadmap produit		2	8	0	16
Total Produit				-1		-1

10.14 - Traitement d’une requête par l’external processor

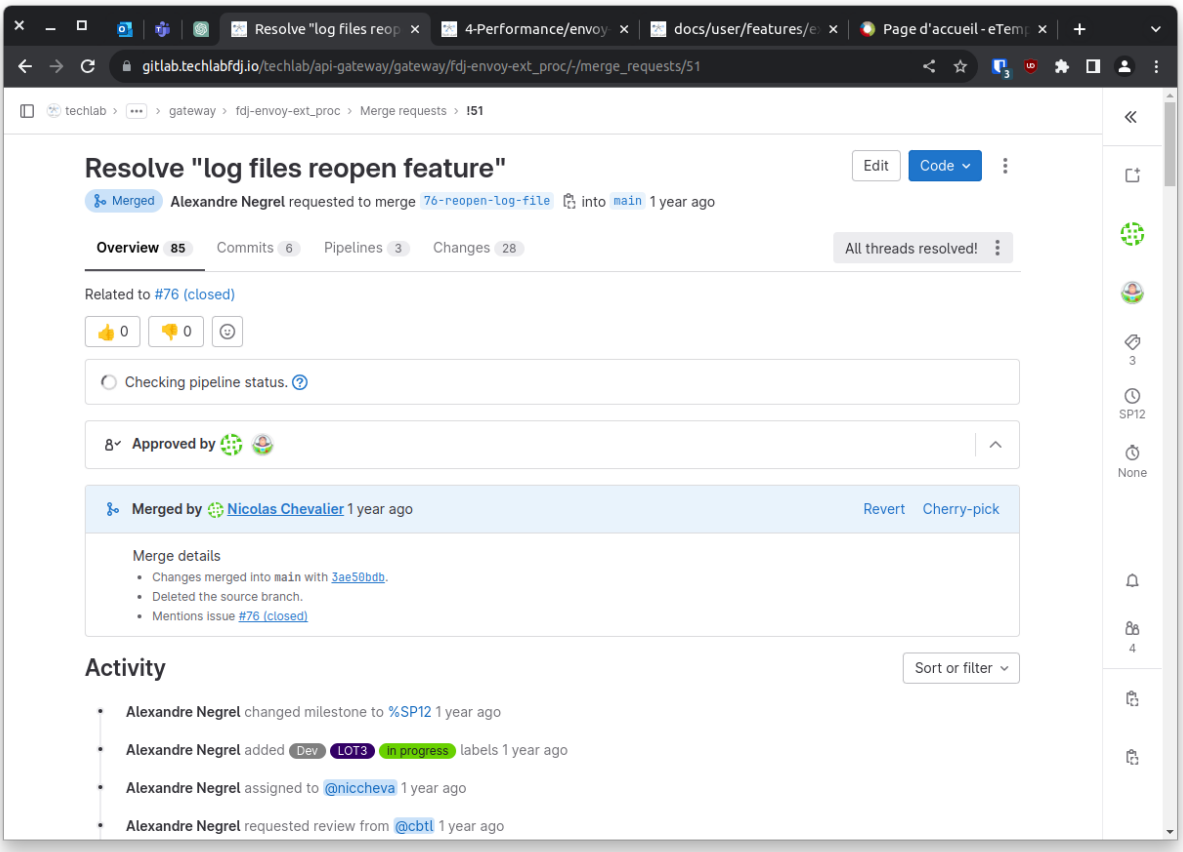


10.15 - Capture d’écran de la tâche réouverture à chaud des fichiers de journalisation dans GitLab

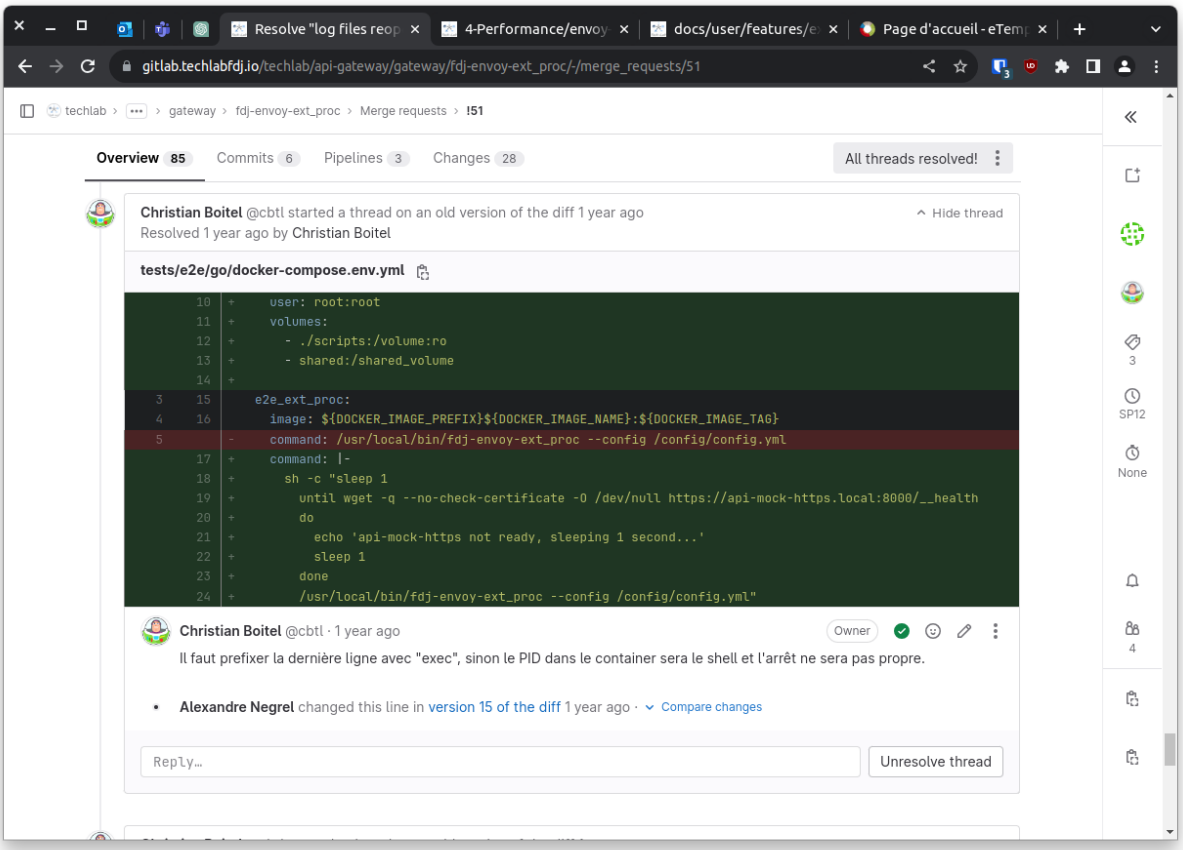


10.16 - Capture d’écrans de la Merge Request GitLab pour la réouverture à chaud des fichiers de journalisation

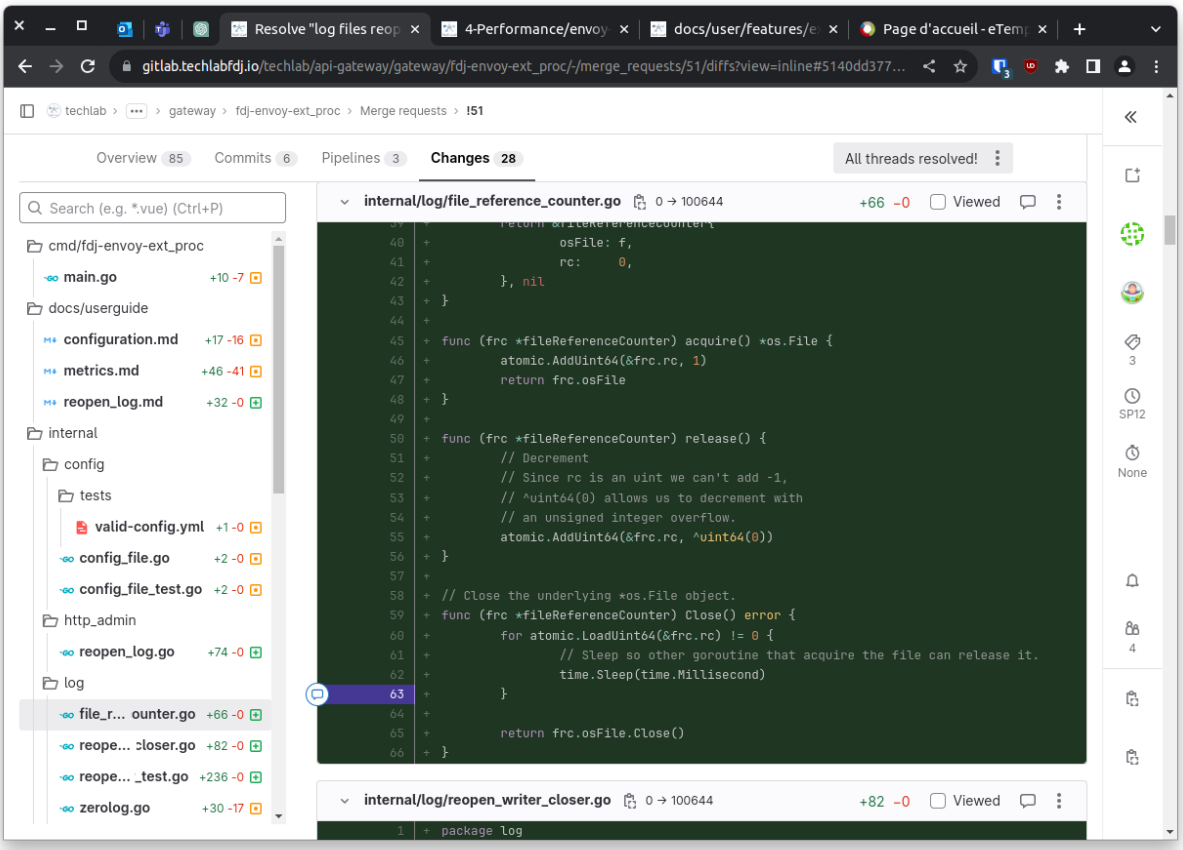
10.16.1 - Vue d’ensemble



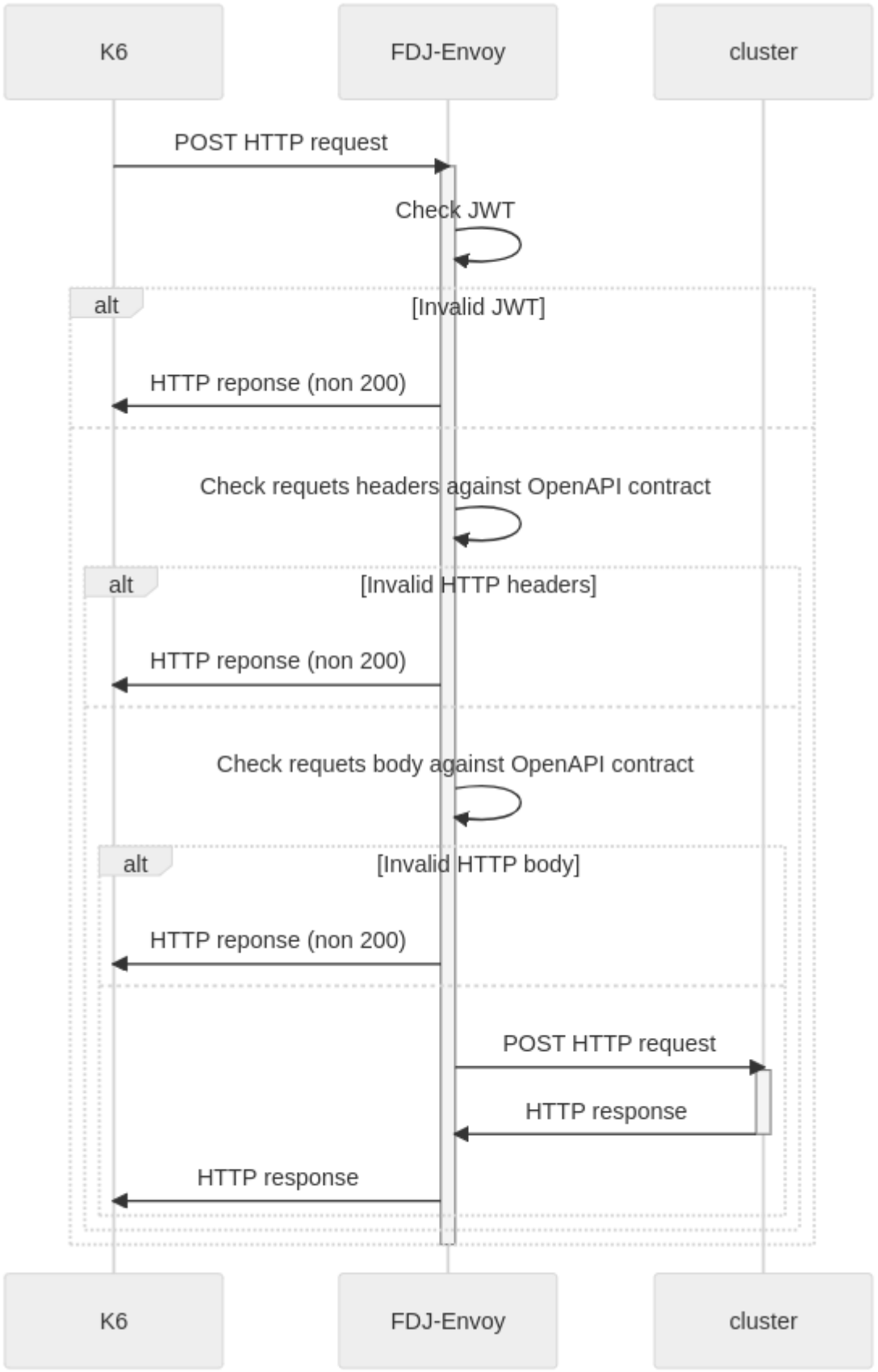
10.16.2 - Exemple d’un commentaire issu de la revue de code



10.16.3 - Onglet des changements



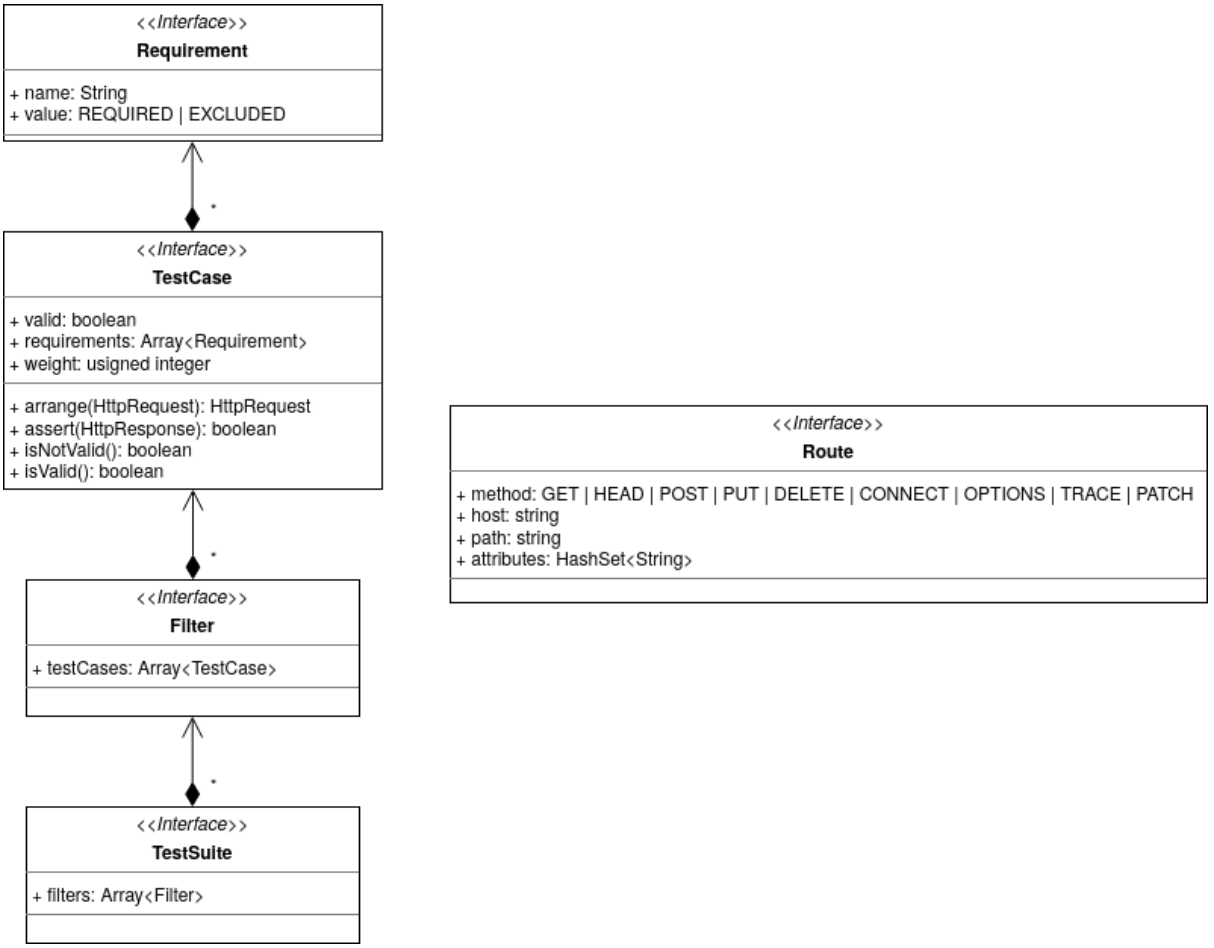
10.17 - Traitement d’une requête par FDJ-Envoy



10.18 - Recherche en profondeur de l’infrastructure logicielle

```
function executeTestSuite(testSuite) {
  forEachTestCaseCombination(testSuite.filters, executeSingleTest)
}

function executeSingleTest(combination) {
    // Execute a single test.
}

/** Finds all possible combination of test cases and call the given
 * callback with it.
 * @param filters Array<Filter> is the list of filters to combine.
 * @param callback function(Array<TestCase>) callback to call.
 * @param combination Array<TestCase> is a private parameter used for
 * recursive call.
 */
function forEachTestCaseCombination(fitlers, callback, combination = []) {
  // Combination is complete, we've reached a leaf node.
  if (combination.length === features.length) {
    callback(combination);
    return;
  }

  const filter = filters[combination.length];
  const filterTestCases = filter.testCases;

  for (testCase of filterTestCases) {
    // Add test case to unique combination.
    combination.push(testCase);

    forEachTestCaseCombination(filters, callback, combination)

    // Remove test case so next iteration can reuse this index.
    combination.pop()
  }
}
```

10.19 - Diagramme de classe de l'infrastructure logicielle de test d'Envoy



10.20 - Algorithme d'exécution d'un test par l'architecture logicielle.

```
function ExecuteSingleTest(combination, routes)
    // Arrange
    client ← newHttpClient
    // La valeur -1 signifie que la réponse provient du service.
    responseSource ← -1
    routeRequirements ← []
    request ← newHttpRequest
    for index, testCase in combination do
        request ← testCase.arrange(request)
        routeRequirements ← append(routeRequirements, testCase.requirements)
        if responseSource == -1 and testCase.isNotValid then
            responseSource ← index
        end if
    end for
    request.route ← findRoute(routes, routeRequirements)


    // Act
    response ← doRequest(client, request)


    // Arrange
    // La réponse provient du service.
    if responseSource == -1 then
        for testCase in combination do
            if testCase.assert(response) then
                panic("actual response doesn't match expected")
            end if
        end for
    else
        // La réponse provient d'un filtre.
        assert ← combination[responseSource].assert
        if assert(response) then
            panic("actual response doesn't match expected")
        end if
    end if
end function
```