

RAPPORT DE PROJET DE FIN D'ETUDES

Filière : Génie informatique

Framework de Génération et Automatisation des Tests d'API Rest

Effectué à :

Intelcia IT Solutions, Casablanca

Réalisé par :

Houssam Eddine
Abou el haoul

Encadré par :

Maitre de Stage :

Khadija Bazoun

Encadrant Pédagogique :

Said OUATIK EL ALAOUI

Soutenu le 27 Juin 2024

Devant le jury :

Said OUATIK EL ALAOUI	:	Professeur de l'enseignement supérieur, ENSA Kénitra
Ilham OUMAIRA	:	Professeur de l'enseignement supérieur, ENSA Kénitra
Youssef Gahi	:	Professeur de l'enseignement supérieur, ENSA Kénitra

Année Universitaire 2023-2024

“

Je dédie humblement ce travail avec un amour profond, en invoquant le nom de Dieu, le Tout
Miséricordieux, le Très Miséricordieux.

À ma très chère mère, qui est ma source d'amour et qui m'entoure de tendresse et d'espoir depuis
ma naissance, je ne saurais exprimer suffisamment ma gratitude pour ta présence à mes côtés.

À mon cher père, ma force et mon soutien, je te remercie pour tes sacrifices et tes efforts
inlassables pour nous offrir le meilleur.

À toute ma famille et mes chers ami(e)s, je leur suis reconnaissant(e) pour leur soutien
permanent, leurs encouragements et leur appui tout au long de mon parcours universitaire.

Je suis également reconnaissant envers mes chers professeurs pour leur serviabilité, leur soutien,
leur compréhension et leurs efforts qui méritent mon plus profond respect.

Que ce travail soit le reflet de ma gratitude et de mon affection envers tous ceux qui m'ont
entouré de leur amour et de leur soutien.

“

- Houssam Eddine

Remerciement

Au terme de ce travail, j'exprime mes vifs remerciements à toute personne ayant contribué, de près ou de loin, à la réalisation de ce travail.

Je tiens à remercier tout d'abord les professeurs de ENSA KENITRA, notamment mes professeurs qui n'ont ménagé aucun effort afin de nous assurer ce parcours et nous transmettre leurs connaissances, leur expérience et leurs valeurs.

Ma profonde gratitude s'adresse à l'entreprise « Intelcia It Solutions » au nom de mon manager KHADIJA BAZOUN qui m'a accueilli et qui a dirigé mon projet de fin d'études, pour son engagement permanent, son soutien constant et la confiance totale qu'elle m'a accordée. Je tiens à remercier également toute l'équipe de travail chez Intelcia It Solutions, Madame Khadija Bazoun, Monsieur YeYe Nailey, Franck BOSSARD, Monsieur Abdelmoughit RABIA, Yassine ZAKARI, Youssef MOUHIB, Hamza EL MOUTTAQI, Yassine BENJAA, Ahmed AIT BENOUAZAR pour le temps qu'ils m'ont consacré durant cette période et pour le partage de leur expertise et connaissance. C'est grâce à leur confiance et leur encouragement que j'ai été apte à accomplir mes missions et à surmonter toute difficulté rencontrée tout en gagnant de l'expérience.

J'adresse aussi mes profonds remerciements à Monsieur Said OUATIK EL ALAOUI, mon encadrant pédagogique à la ENSAM Casablanca, pour son encadrement, son assistance, pour tout le temps précieux qu'il m'a octroyé et pour ses conseils qui m'ont été bien utiles pour la rédaction de ce rapport.

Par ailleurs, je souhaite adresser mes remerciements aux membres du jury pour leur présence lors de l'évaluation de mon rapport de stage. Leur expertise et leurs commentaires constructifs m'ont permis de prendre du recul sur mon travail et de l'améliorer.

Liste des Abréviations

<u>Abréviation</u>	<u>Désignation</u>
<u>B2B</u>	<u>Business to Business</u>
<u>API</u>	<u>Application Programming Interface</u>
<u>CI/CD</u>	<u>Continuous Integration / Continuous Deployment</u>
<u>CLI</u>	<u>Command Line Interface</u>
<u>HTML</u>	<u>Hypertext Markup Language</u>
<u>HTTP</u>	<u>Hypertext Transfer Protocol</u>
<u>CSV</u>	<u>Comma-Separated Values</u>
<u>JDD</u>	<u>Jeux De Données</u>
<u>JSON</u>	<u>JavaScript Object Notation</u>
<u>REST</u>	<u>Representational State Transfer</u>
<u>BDD</u>	<u>Base de Données</u>
<u>UML</u>	<u>Unified Modeling Language</u>
<u>NPM</u>	<u>Node Package Manager</u>

Ce rapport offre une analyse détaillée du développement et de l'implémentation d'un Framework d'automatisation des tests d'API REST pour le projet Pléiade de Intelcia IT Solutions. Après avoir introduit l'entreprise et le contexte du projet, une analyse pointue des besoins spécifiques en matière de tests d'API REST pour Pléiade est présentée. Le rapport explore ensuite en détail la conception et la réalisation du Framework, en mettant en lumière les choix techniques stratégiques et les défis rencontrés tout au long du processus.

Le Framework développé a pour objectif principal d'améliorer la qualité, la fiabilité et l'efficacité des tests d'API au sein de Pléiade . En automatisant les tests, le projet vise à réduire de manière significative les délais de développement, à accroître la couverture des tests, à détecter rapidement les anomalies et à améliorer la réactivité de l'équipe face aux évolutions des services. Des technologies telles que Node.js, Postman, GitLab CI, Scrum et Swagger ont été utilisées pour développer et intégrer ce Framework.

Mots clés : API, REST, Framework, SCRUM, Node.js, Postman, Swagger, GitLab.

This report provides an in-depth analysis of the development and implementation of an API REST automation testing Framework for the Pléiade project by Intelcia IT Solutions. After introducing the company and project context, a sharp analysis of Pléiade 's specific needs regarding API REST testing is presented. The report then delves into the design and realization of the Framework, highlighting strategic technical choices and challenges encountered throughout the process.

The primary goal of the developed Framework is to improve the quality, reliability, and efficiency of API testing within Pléiade . By automating tests, the project aims to significantly reduce development timelines, increase test coverage, quickly detect anomalies, and enhance team responsiveness to service evolutions. Technologies such as Node.js, Postman, GitLab CI, Scrum, and Swagger were utilized to develop and integrate this Framework.

Keywords : API, REST, Framework, SCRUM, Node.js, Postman, Swagger, GitLab.

Table de Matières

Dédicace.....	2
Remerciement.....	3
Liste des Abréviations	4
Résumé	5
Abstract	6
Table de Matières	7
Liste des figures	10
Liste des tableaux.....	13
Introduction.....	14
Chapitre 1:Contexte du projet	17
INTRODUCTION	17
1.1 Présentation de l'organisme d'accueil	17
1.2 Services d'Intelcia.....	18
1.3 Le pôle IT Solutions.....	19
1.3.1 Services d'Intelcia IT Solutions	20
1.3.2 Clients Intelcia IT Solutions	23
1.3.3 Structure organisationnelle	25
1.4 Présentation Générale du Projet.....	26
1.4.1 La Présentation du Client SFR.....	26
1.4.2 Présentation du projet Pléiade Service	27
1.5 Problématique du PFE.....	28

CONCLUSION	29
Chapitre 2 : Etude Préalable.....	31
INTRODUCTION	31
2.1 Présentation des principes de Bases	31
2.1.1 Définition du principe http.....	31
2.1.2 Definition du principe API	32
2.2 Présentation Générale du principe des Tests.....	32
2.2.1 Définition du principe des Tests.....	32
2.2.2 Les Types de Tests	34
2.2.3 Comment tester les API Rest.....	35
2.2.4 Exemple d'un test API REST avec Postman	36
2.3 Critique de l'existant.....	37
2.4 Comparaison des outils d'Automatisation de Test	38
2.5 Objectifs du Framework	39
CONCLUSION	40
Chapitre 3:Conception et Réalisation	41
INTRODUCTION	41
3.1 Stack Technique.....	41
3.1.1 Node.js.....	42
3.1.2 Swagger	42
3.1.3 Postman / Newman.....	43
3.1.4 Gitlab CI	44
3.2 Conception du Framework	45
3.2.1 Planification et diagramme de GANTT	45

3.2.2	Architecture du Framework	48
3.2.3	Diagramme de Séquence	50
3.3	Approche de Développement	50
3.4	Fonctionnement du Framework.....	51
3.4.1	Workflow du Framework	51
3.4.2	Composants du Framework	54
3.4.3	Les Interactions entre les Composants	55
	CONCLUSION	57
	Chapitre 4:Présentation de la Solution	59
	INTRODUCTION	59
4.1	Présentation des Étapes de l'Automatisation	59
4.1.1	Téléchargement des Swagger	59
4.1.2	Conversion des Swagger en Collections.....	61
4.1.3	Configuration des Collections	63
4.1.4	Génération des fichiers CSV	64
4.1.5	Injection des Valeurs dans les Collections	65
4.1.6	Ajout des Assertions dans les Collections	67
4.1.7	Exécution des Collections.....	68
4.1.8	Génération des Rapports	70
4.2	Présentation de la Pipeline CI/CD	73
	CONCLUSION	76
	Conclusion et Perspectives.....	77
	Bibliographie	79

Liste des figures

Figure 1.1: Présence d’Intelcia à l'internationale	18
Figure 1.2: Solution offertes par Intelcia	19
Figure 1.3: Service d'infogérance	20
Figure 1.4: Service des solutions applicatives	21
Figure 1.5: Services de Business Intelligence	22
Figure 1.6: Service de Consulting	22
Figure 1.7: Clients d'Intelcia It Solutions	23
Figure 1.8: Organigramme de pôle IT Solution	25
Figure 1.9: Logo SFR.....	26
Figure 1.10: Pleiade Service.....	27
Figure 2.11: Principe du fonctionnement des API dans le contexte de Pléiade	32
Figure 2.12: Cout relatif de correction d'une anomalie	33
Figure 2.13: Exemple de Test API avec Postman.....	36
Figure 2.14: Exemple d'outils d'Automatisation de Test.....	38
Figure 3.15: Node.js.....	42
Figure 3.16:: Swagger	42
Figure 3.17: Postman	43
Figure 3.18: Newman	43
Figure 3.19: GitLab CI.....	44
Figure 3.20: Diagramme de GANTT	45
Figure 3.21: Test Automation Architecture.....	48

Figure 3.22: Diagramme de Séquence..... 50

Figure 3.23: Workflow du Framework..... 51

Figure 3.24: Processus Détaillé de comment la Collection est Traitée 53

Figure 3.25: Architecture du Framework 54

Figure 3.26: Le Module Data 55

Figure 3.27: Le Module SwaggerToPostman 56

Figure 3.28: Le Module Newman 57

Figure 4.29: Execution de npm run download 60

Figure 4.30: Les Swagger Téléchargés 60

Figure 4.31:Execution de npm run convert..... 61

Figure 4.32: Conversion des Swagger en Collection 62

Figure 4.33: Les Collections Convertis 63

Figure 34: Execution de npm run process 63

Figure 4.35:Execution de npm run process 64

Figure 4.36:Execution de npm run inject 65

Figure 4.37: Execution de npm run populate 66

Figure 4.38: Injection des données dans les Collections..... 66

Figure 4.39: Les Collections Configurées avec les Valeurs 67

Figure 4.40: Execution de npm run generate..... 68

Figure 4.41: Execution de npm run run 69

Figure 4.42: Le rapport local dans la CLI..... 69

Figure 4.43: Les rapports Générés dans le dossier Newman 70

Figure 4.44: L'interface Principale du Rapport..... 71

Figure 4.45: L'Onglet Total Requests..... 72

Figure 4.46: l'Onglet Failed Tests..... 72

Figure 4.47: Script de la Pipeline CI/CD 73

Figure 4.48: Onglets Jobs de Gitlab CI 74

Figure 49: Exemple d'écution d'un Job..... 75

Liste des tableaux

Tableau 2.1: Tableau de Benchmark des Outils d'Automatisation des Tests	39
Tableau 3.2: Tableau de GANTT	46

Introduction

De nos jours, chaque entreprise aspire à satisfaire ses clients et à améliorer ses applications afin d'atteindre une version finale de qualité. C'est dans cette optique qu'Intelcia IT Solutions s'est lancé dans le projet Pléiade, visant à optimiser et automatiser les tests d'API REST. L'objectif principal de ce projet est de créer un Framework d'automatisation des tests, permettant de renforcer la qualité et l'efficacité des services offerts.

Le projet Pléiade vise à centraliser et à simplifier les processus de test des API REST en introduisant un Framework automatisé. Pour ce faire, des technologies modernes et robustes telles que Node.js, Postman, GitLab CI, Scrum et Swagger ont été utilisées. Ces choix technologiques permettent d'assurer une meilleure gestion des tests, d'accroître la couverture des tests et de réduire les délais de développement.

Notre stage de fin d'études, réalisé au sein d'Intelcia IT Solutions sur une période de six mois, s'est déroulé dans ce contexte. Nous avons travaillé sur la conception et la réalisation d'un Framework d'automatisation des tests d'API REST pour le projet Pléiade, en résolvant les défis techniques et en répondant aux besoins spécifiques du projet. Ce rapport présente une synthèse du travail réalisé tout au long de ce projet.

Il est composé de quatre chapitres, qui sont les suivants :

- **Chapitre 1** : "Contexte général du projet", nous fournirons une présentation de l'entreprise d'accueil, du sujet du stage, de la problématique abordée et des objectifs visés. De plus, nous expliquerons en détail l'approche adoptée pour garantir le succès du projet et assurer son bon déroulement.

- **Chapitre 2 :** "Analyse et conception de Pléiade", qui décrit en détail le projet Pléiade, ses objectifs et les besoins fonctionnels que l'application doit satisfaire. Nous détaillerons également les méthodes utilisées pour définir les API et les requêtes HTTP, ainsi que la comparaison des outils d'automatisation des tests.
- **Chapitre 3 :** "Conception et réalisation du Framework", où nous analyserons en détail l'architecture technique du projet et en donnerons un aperçu détaillé. De plus, nous présenterons les outils, les technologies et les méthodologies de développement que nous avons sélectionnés et mis en œuvre.
- **Chapitre 4 :** "Présentation du Framework en fonctionnement", où nous présenterons la structure de notre projet et la mise en œuvre des différentes tâches dans le cadre du projet Pléiade, avec des exemples concrets de fonctionnement.

Chapitre 1: Contexte du projet

INTRODUCTION

Dans le cadre de mon Projet de Fin d'Études (PFE), j'ai eu l'opportunité d'intégrer Intelcia IT Solutions, une entreprise spécialisée dans les solutions technologiques innovantes. Intelcia IT Solutions se distingue par son engagement envers l'innovation et l'excellence technologique. Le projet spécifique auquel j'ai été assigné s'appelle Pléiade, et il vise à améliorer divers aspects des services de l'entreprise. Au sein de ce projet, ma mission principale était d'automatiser les tests d'API REST pour améliorer l'efficacité et la qualité des services fournis. Ce chapitre présente la société d'accueil et décrit le cadre du projet, en mettant en lumière les enjeux et les objectifs de cette initiative.

1.1 Présentation de l'organisme d'accueil

INTELCIA Group est le premier groupe marocain spécialisé dans les métiers de l'externalisation des services avec une démarche de partenaire à valeur ajoutée. Il intervient dans les secteurs télécoms, médias, assurances, e-commerce et d'autres secteurs. Créé il y a 20 ans au Maroc et porté par une ambition forte, le Groupe a rapidement étendu sa présence géographique et ses activités pour accompagner ses clients sur l'ensemble de leurs opérations d'externalisation et leur permettre de mieux se concentrer sur leurs enjeux et cœur de métier.

En 2020, INTELCIA poursuit son expansion internationale avec une implantation en Amérique dans les caraïbes et de nouvelles destinations en Europe. Avec plus de 26 580 collaborateurs, le groupe est présent aujourd'hui sur 56 sites à travers 17 pays dont 1 site client en France et 11 sites clients au Portugal et ambitionne d'atteindre 30 000 collaborateurs vers la fin de 2021 avec une présence plus importante en Europe et de nouvelles implantations aux Etats-Unis.



Figure 1.1: Présence d'Intelcia à l'internationale

1.2 Services d'Intelcia

Les services proposés par Intelcia sont vastes et variés, permettant à ses clients de se concentrer sur leur cœur de métier tout en bénéficiant d'une assistance efficace et personnalisée. Les pôles de solutions offrent une gamme complète de services qui couvrent différents domaines, allant de la gestion de la relation client à la gestion des processus métiers, en passant par la comptabilité, le recouvrement, les ressources humaines, la conformité et l'IT.

Grâce à une approche innovante et centrée sur le client, Intelcia offre des solutions sur mesure pour chaque entreprise, basées sur les dernières technologies et les meilleures pratiques de l'industrie. Les équipes d'experts d'Intelcia travaillent en étroite collaboration avec les clients pour comprendre leurs besoins spécifiques et leur offrir des solutions adaptées à leurs besoins, visant à améliorer l'efficacité, la qualité et l'expérience client. Les solutions d'Intelcia sont conçues pour aider les clients à atteindre leurs objectifs tout en optimisant leurs coûts et en améliorant leur compétitivité sur le marché.



Figure 1.2: Solution offertes par Intelcia

1.3 Le pôle IT Solutions

Intelcia It Solutions est une filiale d'Intelcia, spécialisée dans les services de technologies de l'information. L'entreprise offre une gamme complète de services IT pour aider ses clients à atteindre leurs objectifs commerciaux en utilisant les dernières technologies et les meilleures pratiques de l'industrie.

Les services proposés par Intelcia It Solutions comprennent la consultation, l'intégration, le développement d'applications, l'exploitation d'infrastructures, la régie et bien plus encore. Les experts d'Intelcia It Solutions travaillent en étroite collaboration avec les clients pour comprendre leurs besoins spécifiques et leur offrir des solutions personnalisées pour améliorer leur efficacité et leur productivité.

Grâce à son expertise en matière de technologie et à sa solide expérience dans l'externalisation de services, Intelcia It Solutions offre des solutions efficaces et rentables pour les entreprises de toutes tailles. Que ce soit pour la mise en œuvre de nouveaux systèmes, la gestion de projets informatiques complexes ou la maintenance de l'infrastructure existante, Intelcia It Solutions est un partenaire de confiance pour aider les clients à atteindre leurs objectifs commerciaux.

1.3.1 Services d'Intelcia IT Solutions

Intelcia IT Solutions est le pôle d'expertise IT du groupe Intelcia, qui offre une gamme complète de services pour aider les entreprises à améliorer leur efficacité et leur productivité grâce aux technologies de l'information. Les services offerts par Intelcia IT Solutions sont regroupés en 4 piliers principaux :

1. **Services managés** : Ce pilier de service fournit des solutions pour la gestion et la maintenance des infrastructures informatiques des entreprises. Cela inclut des services de surveillance, de maintenance préventive, de gestion des incidents, de sécurité et de sauvegarde pour assurer une disponibilité et une performance optimales des systèmes informatiques.

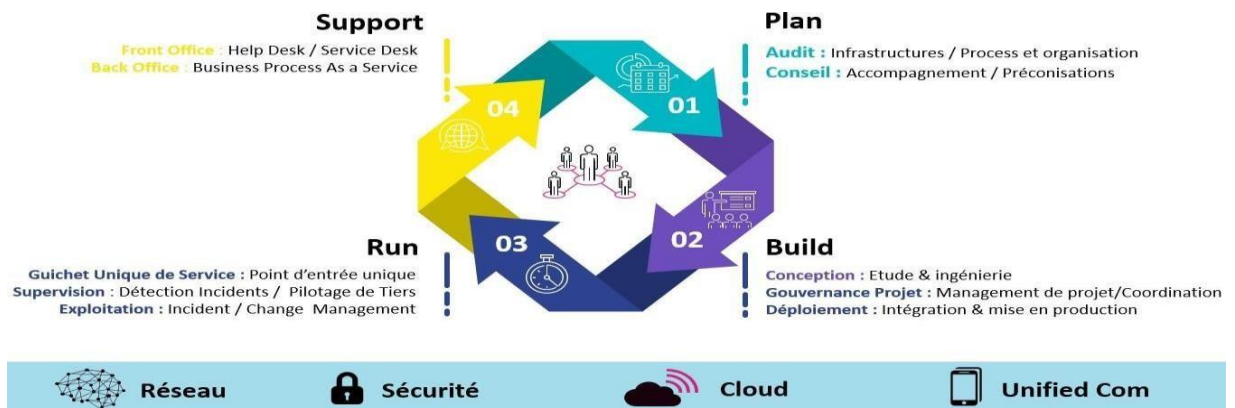


Figure 1.3: Service d'infogérance

2. **Solutions applicatives** : Ce pilier de service offre des solutions pour la conception, le développement et la mise en œuvre d'applications sur mesure pour les entreprises. Les experts d'Intelcia IT Solutions travaillent en étroite collaboration avec les clients pour comprendre leurs besoins spécifiques et concevoir des applications personnalisées pour répondre à ces besoins, tout en offrant un support continu.



Figure 1.4: Service des solutions applicatives

3. **Business Intelligence** : Ce pilier de service propose des solutions pour la collecte, l'analyse et la présentation de données commerciales clés, afin d'aider les entreprises à prendre des décisions éclairées et à améliorer leur efficacité. Les experts d'Intelcia IT Solutions utilisent des technologies avancées d'analyse de données pour identifier les tendances et les opportunités de croissance, et pour aider les entreprises à prendre des décisions basées sur des données.

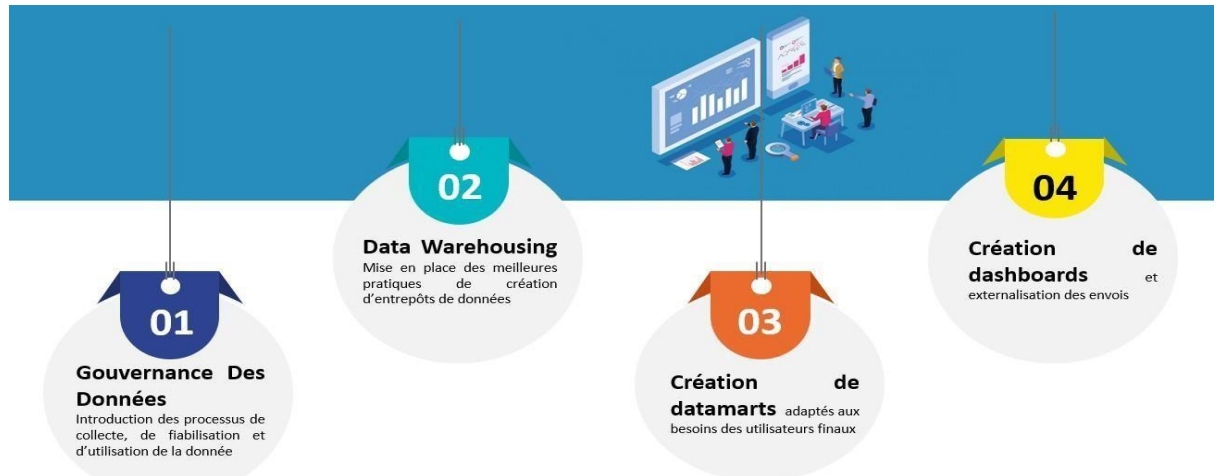


Figure 1.5: Services de Business Intelligence

- 4. Consulting :** Ce pilier de service offre des services de consultation pour aider les entreprises à élaborer une stratégie IT efficace pour leur entreprise, en évaluant leurs besoins et en proposant des solutions adaptées à leurs besoins spécifiques. Les experts d'Intelcia IT Solutions travaillent en étroite collaboration avec les clients pour comprendre leurs défis commerciaux et identifier les opportunités de croissance, tout en proposant des solutions rentables et efficaces pour y répondre.

Mise à disposition de ressources hébergées chez le client ou dans nos locaux à partir de nos sites de présence

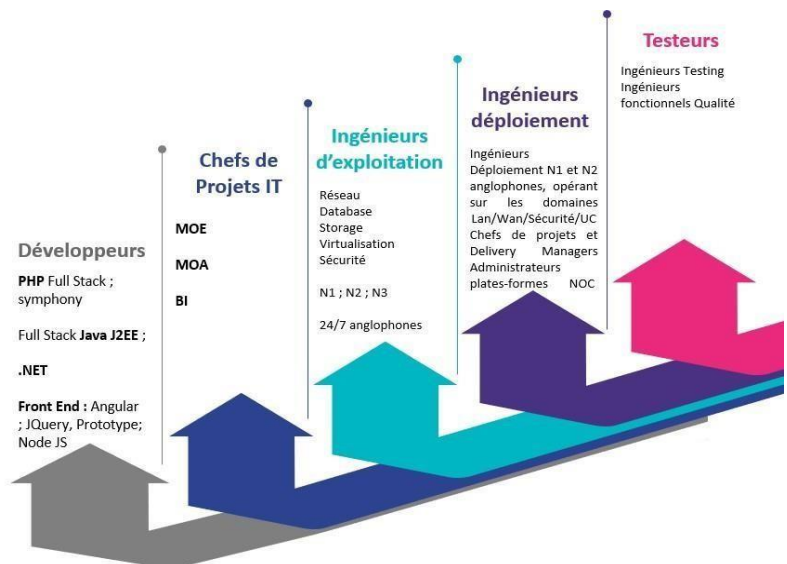
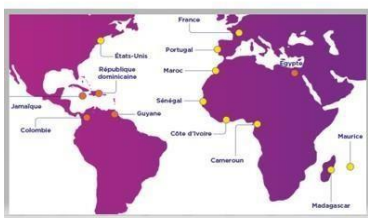


Figure 1.6: Service de Consulting

1.3.2 Clients Intelcia IT Solutions

Intelcia IT Solutions est un partenaire technologique de confiance pour une grande variété de clients, allant des petites et moyennes entreprises aux grandes entreprises. Les clients de l'entreprise proviennent de divers secteurs, tels que la finance, la santé, les télécommunications et la vente au détail. Intelcia IT Solutions comprend les besoins uniques de chaque client et fournit des solutions sur mesure pour répondre à leurs exigences spécifiques. Les clients de l'entreprise comptent sur son expertise dans des domaines tels que le cloud computing, l'analyse de données, la cybersécurité et le développement de logiciels pour améliorer leur efficacité opérationnelle, améliorer l'expérience client et stimuler la croissance. En mettant l'accent sur la construction de partenariats à long terme, Intelcia IT Solutions s'engage à offrir à ses clients un service exceptionnel et à fournir une valeur qui dépasse leurs attentes.



Figure 1.7: Clients d'Intelcia It Solutions

1.3.3 Structure organisationnelle

Le pôle IT Solutions est composé de plusieurs direction représentant chacune des Business Unit. La figure ci-dessous présente la structure organisationnelle du pôle :

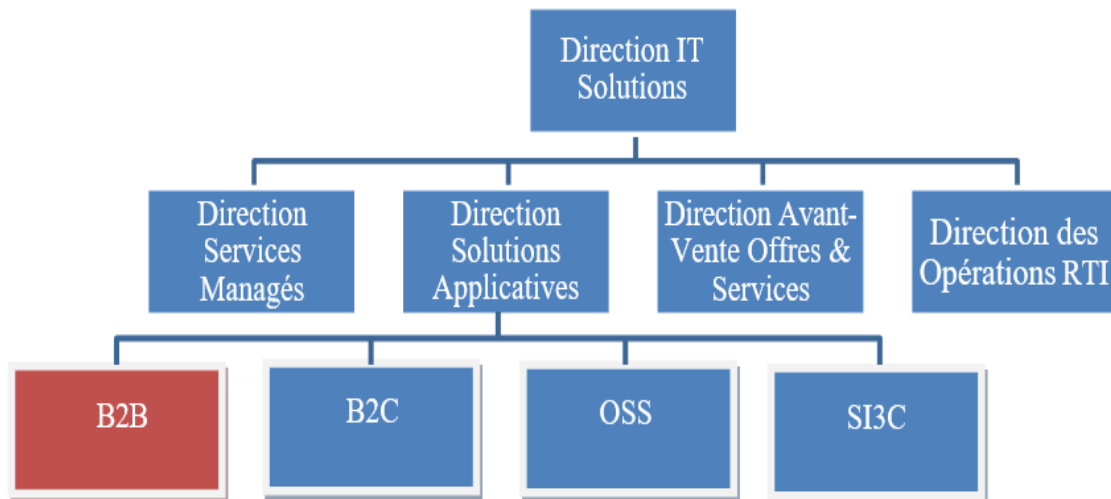


Figure 1.8: Organigramme de pôle IT Solution

Le stage PFE a été effectué au sein de la direction des solutions applicatives, particulièrement dans l'unité opérationnelle de B2B, sous l'encadrement du Manager B2B Mme Khadija Bazoun.

Le terme B2B (Business-to-Business) désigne les transactions commerciales entre entreprises, par opposition aux transactions entre une entreprise et des consommateurs individuels (B2C). Intelcia IT Solutions, spécialisée dans les solutions technologiques, compte parmi ses clients B2B de premier plan l'opérateur téléphonique SFR.

1.4 Présentation Générale du Projet

1.4.1 La Présentation du Client SFR



Figure 1.9: Logo SFR

La Société Française du Radiotéléphone, SFR, est un opérateur français de télécommunications créé en 1987 par la Compagnie générale des eaux. SFR propose un réseau très haut débit, fixe et mobile, sur le territoire français à destination des particuliers et des entreprises. L'entreprise réalise également des investissements importants pour le développement de la 5G.

Les services de SFR Business sont basés principalement sur des offres et solutions IT et infrastructures réseaux, sont présentées ci-contre :

- Les solutions de sécurité Internet
- Les solution d'Internet des Objets
- Les offres de Relation client
- Les solutions WAN et LAN
- Les offres et abonnements de téléphonie fixe et mobile (4G /5G).

1.4.2 Présentation du projet Pléiade Service

Pléiade service est un middle office qui a pour but de fournir des services génériques et simplifiés à l'ensemble des acteurs SI du B2B. En incluant un contrôle des habilitations en fonction des besoins de nos partenaires.

Objectif du Pléiade service :

- ❖ Fournir des services simples d'utilisations et génériques
- ❖ Garantir la confidentialité des données clients en fonctions des différents contextes
- ❖ Masquer la complexité du SI à l'ensemble de nos consommateurs
- ❖ Centraliser les API exposé aux clients externes de SFR

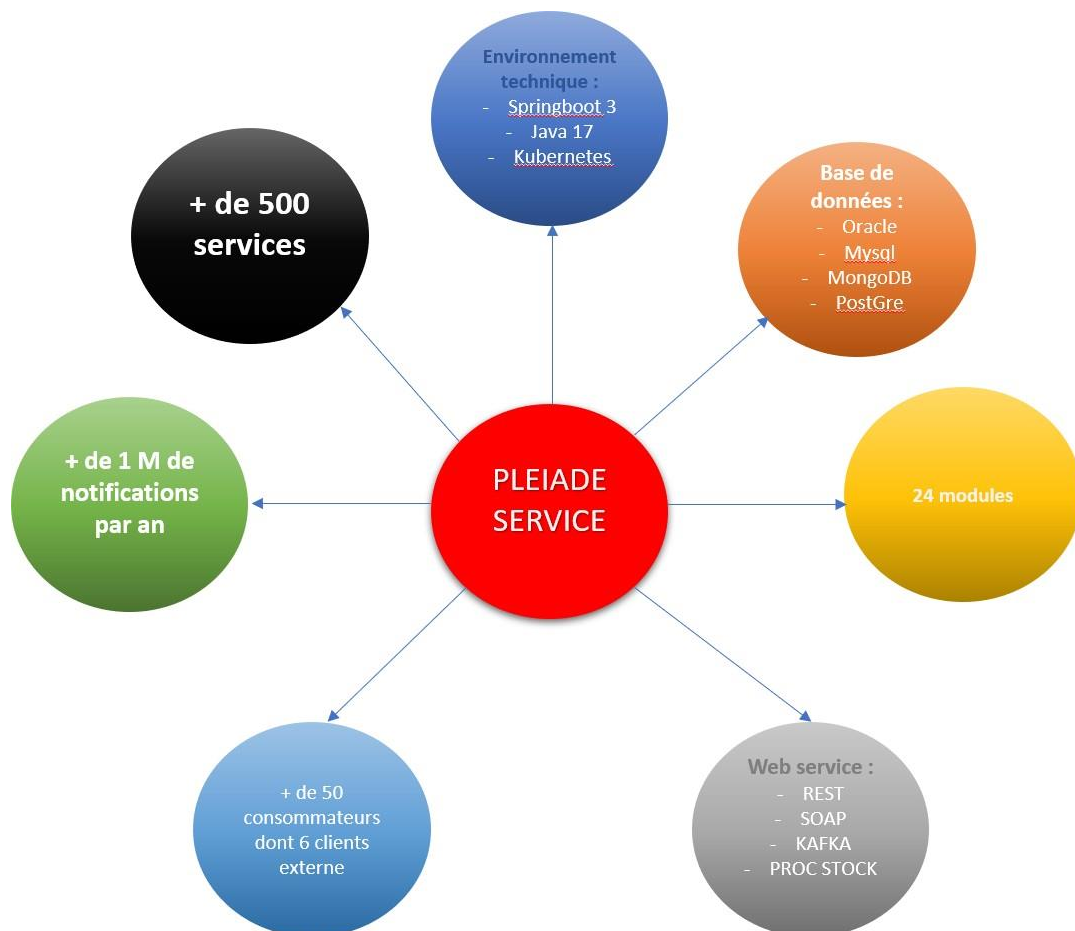


Figure 1.10: Pleiade Service

1.5 Problématique du PFE

Vu la taille du projet Pléiade et de son évolution constante, tester toutes les requêtes et API (plus de 600) se trouve être une tâche monumentale qui ne peut pas être réalisable manuellement dont voici les plus grands obstacles :

- **Changements Constants** : le projet Pléiade est en constante évolution pour toujours satisfaire les besoins des clients SFR, ainsi les API exposés changent constamment. Ce changement peut affecter soit les paramètres, soit les réponses des requêtes etc.
- **Evolution Continue** : le projet Pléiade ajoute toujours de nouveaux modules dans son catalogue qui doivent être testés à leur tour. En plus ces nouveaux modules peuvent engendrer des régressions dans des modules existants et qui fonctionnait bien avant.
- **Complexité des tests** : pour effectuer des tests sur les API Rest de Pléiade il faut une connaissance profonde du métier que fournit SFR et ses différents services que Pléiade consomme pour savoir si tout fonctionne comme prévu.

Durant le stage PFE, nous avons eu pour mission d'automatiser tout le processus de tests, c.-à-d. de s'occuper de la création des tests, de leur exécution et de la génération des rapports pour les chefs de projets.

Dans notre cas il n'y a pas d'outils ou Framework qui correspond à nos besoins spécifiques. Du coup ma mission consiste en l'élaboration d'une solution 'from scratch'.

CONCLUSION

En conclusion, Intelcia IT Solutions se positionne comme un acteur majeur dans le domaine des solutions technologiques, avec une forte orientation vers l'amélioration continue et l'optimisation de ses processus. Le projet Pléiade, auquel j'ai eu la chance de contribuer, m'a offert une perspective unique sur les défis et les opportunités liés aux technologies de l'information.

Le prochain chapitre sera consacré à l'étude préalable que j'ai réalisée sur l'automatisation des tests d'API REST, en abordant les méthodologies, les outils disponibles, et les bonnes pratiques identifiées pour la mise en œuvre de cette initiative.

Chapitre 2 : Etude Préalable

INTRODUCTION

Dans ce chapitre, nous aborderons la gestion du projet Pléiade avec la méthodologie Scrum, en décrivant les définitions des API et des requêtes HTTP, ainsi que les différents types de tests. Nous examinerons également une comparaison des outils d'automatisation des tests. Enfin, nous présenterons l'objectif principal de mon Projet de Fin d'Études (PFE), qui est de développer un Framework pour automatiser les tests d'API de Pléiade. Cette analyse permettra de comprendre comment Pléiade utilise des pratiques agiles pour optimiser le développement et la qualité des services tout en posant les bases de mon projet d'automatisation.

2.1 Présentation des principes de Bases

2.1.1 Définition du principe http

Le protocole HTTP (HyperText Transfer Protocol) est le fondement de la communication sur le World Wide Web. Il définit les règles pour le transfert de données entre un client (comme un navigateur web) et un serveur web. HTTP fonctionne selon un modèle de requête-réponse : le client envoie une requête HTTP au serveur, qui traite cette requête et renvoie une réponse contenant les données demandées (comme une page web) ou des informations sur l'état de la requête. Les requêtes et réponses HTTP sont stateless, ce qui signifie que chaque requête est indépendante et sans mémoire des interactions précédentes.

2.1.2 Definition du principe API

Une API (Interface de Programmation Applicative) dans le contexte de REST (Representational State Transfer) est une approche architecturale qui définit des règles pour la communication et la manipulation des ressources sur le web. Contrairement aux services web traditionnels, les API REST utilisent les méthodes HTTP standard (GET, POST, PUT, DELETE, etc.) pour effectuer des opérations sur des ressources, représentées par des URI (Uniform Resource Identifier) et des états. Cette approche permet une interaction uniforme et scalable entre les clients et les serveurs, en suivant les principes fondamentaux de REST tels que l'utilisation de ressources, l'adressage par URI, et la représentation des états des ressources.

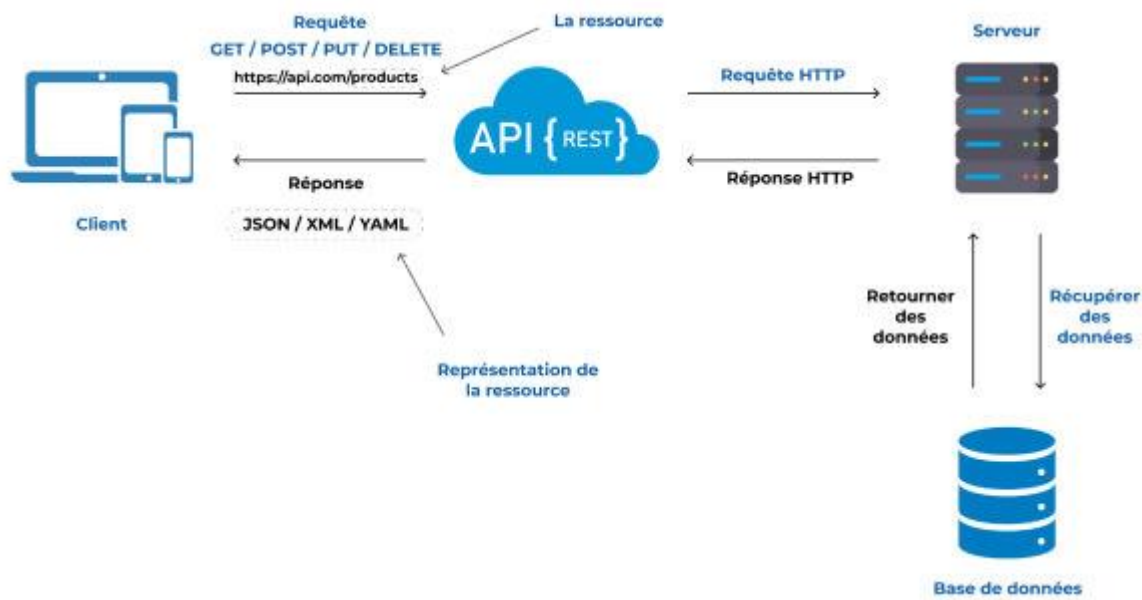


Figure 2.11: Principe du fonctionnement des API dans le contexte de Pléiade

2.2 Présentation Générale du principe des Tests

2.2.1 Définition du principe des Tests

Les tests en général consistent à évaluer et à vérifier que les différentes composantes d'un système ou d'un logiciel fonctionnent comme prévu. Cela inclut l'identification des défauts, des bugs ou des dysfonctionnements potentiels pour garantir que le produit final est fiable, performant et répond aux exigences spécifiées. Les tests peuvent être manuels ou automatisés et couvrent une large gamme

d'aspects, allant de la vérification des fonctionnalités individuelles à l'assurance de l'intégrité du système complet sous diverses conditions d'utilisation.

Les tests garantissent que ce que vous obtenez en fin de compte est ce que vous vouliez construire. Nous sommes des êtres humains et les êtres humains font des erreurs pendant le développement. Certaines erreurs n'ont pas beaucoup d'impact sur notre vie quotidienne et peuvent être ignorées, mais certaines erreurs sont si graves qu'elles peuvent briser tout le système ou le logiciel. Dans ce genre de situations, vous devez veiller à ce que ces erreurs soient détectées bien à l'avance et avant de déployer le système/logiciel dans l'environnement de production. Exemples : Les sites Web bancaires ou les distributeurs automatiques de billets devraient être soigneusement testés afin de maintenir la crédibilité de la banque. Le système de contrôle du trafic aérien est un système très critique qui doit être testé avant d'être déployé en direct, car les vies humaines en dépendent.

Comme vous pouvez le voir ci-dessous, la figure représente le coût relatif de correction de bugs. Elle offre une visualisation des coûts associés à la correction de bugs, en fonction de différents paramètres ou facteurs.

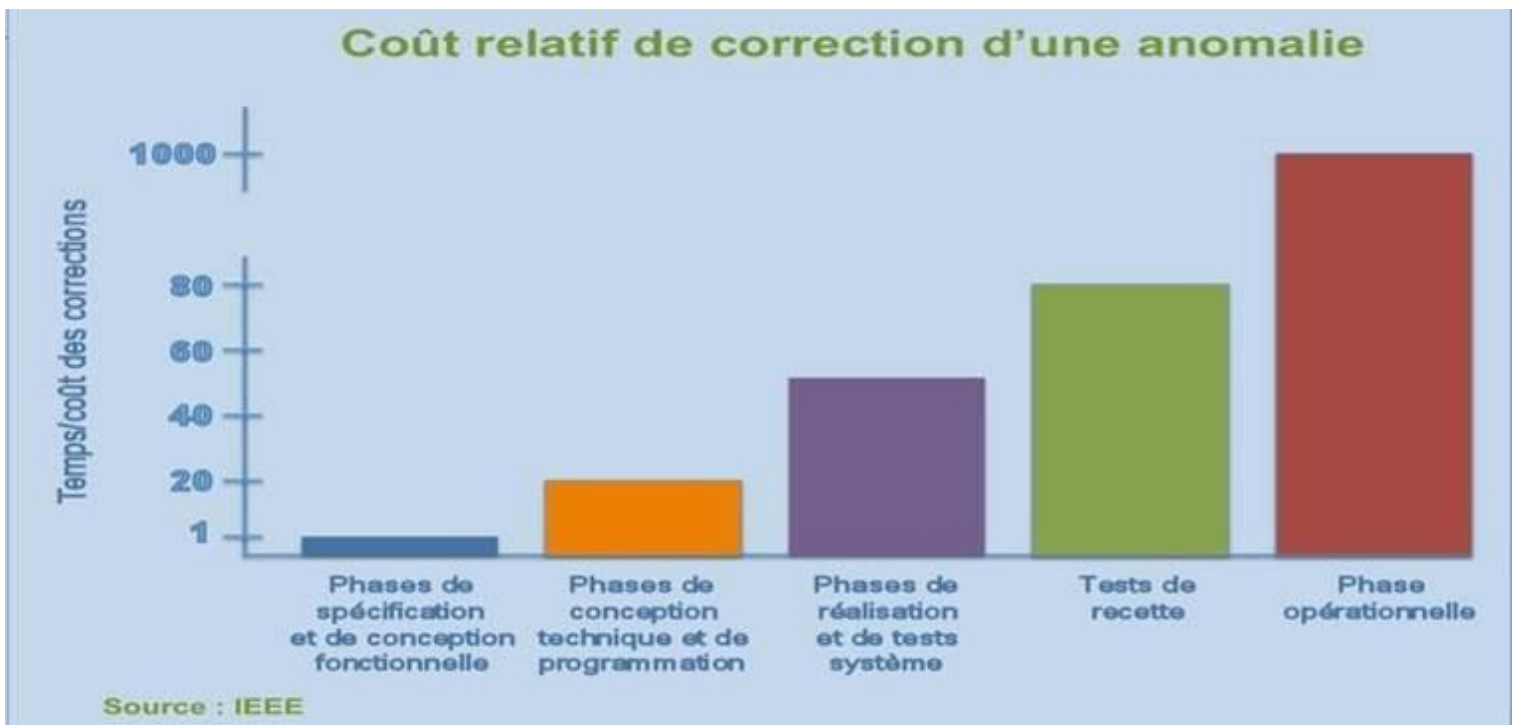


Figure 2.12: Coût relatif de correction d'une anomalie

2.2.2 Les Types de Tests

Plusieurs techniques de test sont utilisées pour garantir une couverture exhaustive des fonctionnalités du système. Parmi ces techniques, on retrouve :

- **Les tests unitaires** : Ces tests visent à vérifier chaque composant individuellement pour s'assurer de son bon fonctionnement.
- **Les tests d'intégration** : Ils consistent à vérifier la communication et l'interaction entre les différents composants du système.
- **Les tests de validation** : Ces tests sont effectués pour valider les fonctionnalités du système dans son ensemble, en s'assurant qu'il répond bien aux besoins et aux attentes des utilisateurs.
- **Les tests de performance** : Ces tests évaluent la rapidité, l'efficacité et la stabilité du système sous diverses charges de travail pour garantir qu'il fonctionne correctement même sous des conditions de stress.
- **Les tests de sécurité** : Ils visent à identifier les vulnérabilités potentielles du système et à s'assurer que les données sont protégées contre les accès non autorisés et les attaques malveillantes.
- **Les tests de compatibilité** : Ces tests vérifient que le système fonctionne correctement sur différentes configurations matérielles, systèmes d'exploitation et navigateurs, assurant ainsi une expérience utilisateur cohérente et fiable.
- **Les tests de régression** : Ils sont réalisés pour s'assurer que les modifications ou les ajouts de fonctionnalités n'ont pas introduit de régressions dans le système, c'est-à-dire des dysfonctionnements ou des effets indésirables sur les fonctionnalités existantes.

2.2.3 Comment tester les API Rest

Les API REST (Representational State Transfer) sont un type d'interface de programmation qui permet aux différentes applications de communiquer entre elles via le protocole HTTP. Elles suivent des principes de conception spécifiques, comme l'utilisation de méthodes HTTP standard (GET, POST, PUT, DELETE) et l'accès à des ressources via des URL. Les API REST sont largement utilisées en raison de leur simplicité, de leur scalabilité et de leur capacité à être facilement intégrées dans des systèmes distribués.

Tester les API REST consiste à vérifier que les différents endpoints de l'API fonctionnent comme prévu. Cela implique de s'assurer que les réponses des API sont correctes et conformes aux spécifications, en termes de format de données, de statut HTTP et de contenu. Les tests d'API REST peuvent inclure :

- **Tests de validation** : Vérifier que chaque endpoint répond correctement aux différentes requêtes (GET, POST, PUT, DELETE) avec les paramètres appropriés.
- **Tests de performance** : Mesurer le temps de réponse de l'API sous différentes charges pour s'assurer qu'elle peut gérer un grand nombre de requêtes simultanées.
- **Tests de sécurité** : S'assurer que l'API est protégée contre les attaques courantes, comme l'injection SQL, et que les autorisations sont correctement gérées.
- **Tests de compatibilité** : Vérifier que l'API fonctionne correctement sur différents environnements et avec diverses versions de clients.
- **Tests de régression** : Réaliser des tests automatisés pour vérifier que les nouvelles modifications n'ont pas introduit de bugs dans les fonctionnalités existantes.

Dans le contexte de notre projet, nous utilisons Postman pour effectuer ces tests manuellement.

2.2.4 Exemple d'un test API REST avec Postman

Voyons maintenant un exemple pratique de test d'API REST avec Postman avec cette figure ci-dessous :

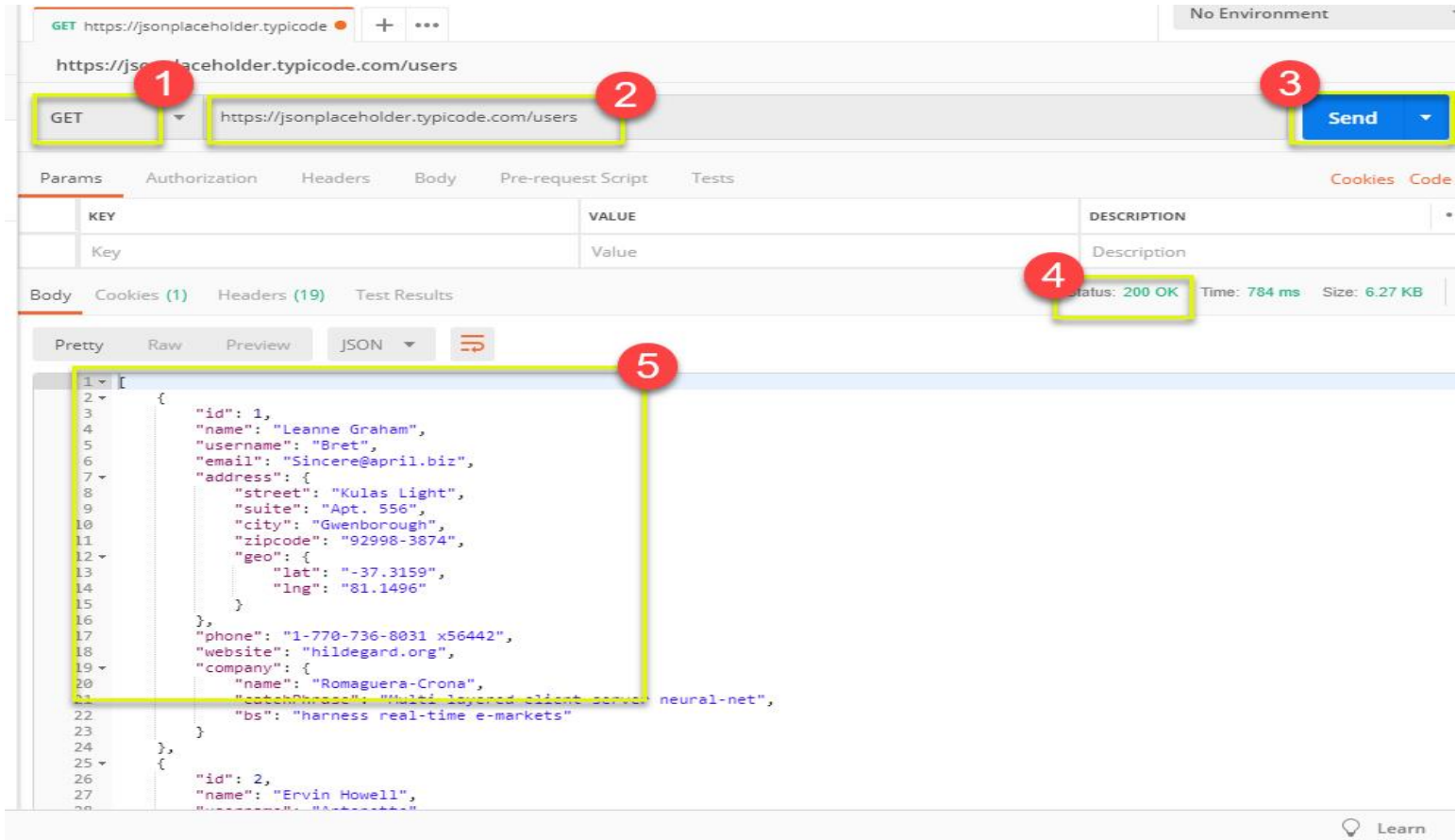


Figure 2.13: Exemple de Test API avec Postman

- 1** Sélectionnez la bonne méthode HTTP (GET, POST, PUT, DELETE, etc.), correspondant à l'action que vous souhaitez effectuer sur l'API.
- 2** Entrez l'URL de l'API dans le champ prévu à cet effet, spécifiant ainsi l'adresse de l'endpoint que vous souhaitez atteindre.
- 3** Cliquez sur le bouton "Send" pour envoyer la requête construite à l'API serveur pour traitement.
- 4** Vérifiez que le code de statut de la réponse indique un succès (par exemple, le code 200 OK), confirmant ainsi que la requête a été traitée avec succès par le serveur.

5 Validez le contenu de la réponse reçue pour vous assurer qu'elle contient les résultats attendus, comme par exemple, 10 enregistrements d'utilisateurs, conformément à vos attentes pour la réussite du test.

2.3 Critique de l'existant

Le processus actuel de test manuel des API dans le projet Pléiade présente plusieurs limitations majeures, dont voici les principales :

- **Complexité des tests manuels** : Dans le contexte du projet Pléiade, effectuer les tests manuels peut présenter un défi vu que on a besoin d'une très grande connaissance du métier et du fonctionnel pour savoir quoi exécuter, comment et à quoi s'attendre comme résultat.
- **Difficulté à maintenir les tests avec les évolutions continues** : Assurer la cohérence des tests à travers les nombreuses mises à jour et modifications.
- **Problèmes liés aux tests de régression** : Les tests manuels de régression deviennent presque impossibles à exécuter de manière complète et fiable dans ce contexte, vu la quantité énorme des tests à exécuter.

En vue de tous ces défis, il est primordial d'automatiser ces tests API Rest.

2.4 Comparaison des outils d'Automatisation de Test

Dans le contexte du stage PFE, nous avons eu la totale autonomie de choisir l'outil avec lequel nous allons accomplir notre tâche d'automatiser les tests des API. Donc, on a dû faire des recherches sur les différents outils d'automatisation pour définir lequel serait le plus adaptés à nos besoins. Voici quelques exemples d'outils :

Tool	Price	Programming Languages	API Support	Ease of Use	Learning Curve	Command Line Execution	CI/CD Integration	Reporting	Community Support
Postman	Free / Paid	JavaScript	REST, SOAP	Easy	Low	Yes	Yes	Good	Excellent
SoapUI	Free / Paid	Groovy, Java	REST, SOAP	Medium	Moderate	Yes	Yes	Good	Good
Rest Assured	Free	Java	REST	Medium	High	Yes	Yes	Excellent	Good
Katalon Studio	Free / Paid	Groovy, Java	REST, SOAP	Easy	Low	Yes	Yes	Good	Good
Karate	Free	Java	REST, SOAP	Medium	Moderate	Yes	Yes	Good	Good
JMeter	Free / Open-Source	Java	REST, SOAP	Medium	Moderate	Yes	Yes	Good	Excellent

Figure 2.14: Exemple d'outils d'Automatisation de Test

Et c'est en compilant toutes ces recherches sur ces différents outils, de leur documentation, de leur point forts et points faible qu'on a pu élaborer ce tableau récapitulatif :

	<u>Jmeter</u>	<u>Rest Assured</u>	<u>Katalon</u>	<u>Assertible</u>	<u>Postman</u>	<u>Soap UI</u>
Services	<u>Rest</u> + SOAP	<u>Rest</u>	Web + Mobile + <u>Rest</u> + SOAP	<u>Rest</u>	<u>Rest</u> + SOAP	<u>Rest</u> + SOAP
Spécialité	Test de Performance	<u>Rest</u> en JAVA	Polyvalent (mais plus interface graphique)	<u>Rest</u>	<u>Rest</u> + SOAP	<u>Rest</u> + SOAP
Facilité d'utilisation	Avancé	Avancé	Facile	Moyen	Facile	Moyen
Intégration CI/CD	CLI + <u>Maven</u> + plugins Jenkins	<u>Maven</u>	CLI	API d' <u>Integration</u>	CLI	CLI
Tarification	Open Source	Gratuit	Gratuit + Payant	Gratuit + Payant	<u>Gratuit</u> + <u>Payant</u>	Open Source + Payant

Tableau 2.1: Tableau de Benchmark des Outils d'Automatisation des Tests

Toutefois, aucun de ces outils ne satisfait pleinement nos besoins, en particulier en ce qui concerne l'automatisation et la maintenance des tests à long terme. C'est pourquoi on a dû élaborer notre propre Framework pour répondre spécifiquement aux exigences du projet Pléiade.

2.5 Objectifs du Framework

Le Framework que nous avons développé vise à accomplir les objectifs suivants à travers quatre axes principaux :

- **Automatisation de la maintenance des API** : Simplifier et automatiser les processus de maintenance des API tout au long de leur cycle de vie. Cela implique la mise à jour automatique des tests suivant les évolutions dans le code et les différentes versions de Pléiade.
- **Automatisation de la récupération des jeux de données de test** : Automatiser le processus de récupération et de gestion des jeux de données nécessaires aux tests des API, ainsi que leur maintenance et application dans les différents cas de tests.

- **Exécution automatique des tests** : Automatiser l'exécution régulière et systématique des tests pour assurer une validation continue des API.
- **Génération automatique des rapports** : Automatiser la génération de rapports détaillés sur les résultats des tests, facilitant ainsi l'analyse et la prise de décision par les différents acteurs et chefs de projet.

Ces axes visent à améliorer l'efficacité, la fiabilité et la cohérence des tests des API, tout en réduisant la charge de travail manuelle et en facilitant l'intégration continue dans le processus de développement du projet Pléiade.

CONCLUSION

En conclusion du Chapitre 2, la gestion du projet Pléiade avec la méthodologie Scrum a été examinée, mettant en évidence les définitions des API et des requêtes HTTP, ainsi que les différents types de tests. Une comparaison des outils d'automatisation des tests a également été effectuée. Cette analyse a permis de comprendre comment Pléiade intègre des pratiques agiles pour optimiser le développement et la qualité des services. En outre, l'objectif principal de mon Projet de Fin d'Études (PFE), qui est de développer un Framework pour automatiser les tests d'API de Pléiade, a été présenté.

Le prochain chapitre sera dédié à la conception et à la réalisation de ce Framework, détaillant les étapes clés et les choix techniques effectués pour atteindre les objectifs fixés.

Chapitre 3: Conception et Réalisation

INTRODUCTION

Dans ce Chapitre 3, nous aborderons en détail la conception et la réalisation du Framework d'automatisation des tests d'API pour le projet Pléiade. Cette étape cruciale de notre Projet de Fin d'Études (PFE) implique l'analyse approfondie des besoins, la planification des fonctionnalités, le choix des technologies et des outils, ainsi que la mise en œuvre concrète du Framework. Cette section vise à décrire de manière exhaustive le processus de conception et de développement, en mettant l'accent sur les méthodologies utilisées et les défis rencontrés dans la création de cet outil essentiel pour l'amélioration de la qualité des services.

3.1 Stack Technique

Notre Framework utilise **Node.js** pour son développement, offrant une plateforme flexible et performante pour l'automatisation des tests des API. En complément, il intègre **Postman** et **Newman** pour la création et l'exécution des collections de tests, assurant ainsi une couverture complète et une facilité d'utilisation. De plus, le Framework est déployé sur **GitLab CI**, permettant une intégration continue et une exécution automatique des tests à chaque déploiement, garantissant ainsi la fiabilité et la stabilité des API tout au long du cycle de développement.

3.1.1 Node.js



Figure 3.15: Node.js

Node.js est une plateforme JavaScript côté serveur qui facilite la gestion des bibliothèques et offre une interface de ligne de commande (CLI) puissante pour le développement et l'automatisation des tâches.

Le principal atout de Node.js est la possibilité de créer des commandes CLI, ce qui est très bénéfique dans le contexte de notre Framework. Cela permet de simplifier l'automatisation des tâches et la gestion des tests, rendant le processus plus efficace et flexible.

3.1.2 Swagger



Figure 3.16:: Swagger

Swagger est un outil de documentation et de conception d'API qui permet de définir les spécifications des API REST de manière standardisée. Il facilite la création d'une documentation interactive qui peut être utilisée pour comprendre et tester les endpoints de l'API.

Dans notre Framework, Swagger sera utilisé pour générer automatiquement les tests à partir des spécifications des API. En utilisant les fichiers de définition Swagger (généralement au format YAML ou JSON), nous pouvons extraire les endpoints, les paramètres et les réponses attendues, et créer des scripts de test correspondants. Cette approche permet d'assurer que les tests sont toujours à jour avec les spécifications de l'API et de simplifier la création et la maintenance des tests. En intégrant Swagger dans notre processus, nous pouvons générer des tests de manière efficace et garantir que notre Framework reste aligné avec les évolutions des API.

3.1.3 Postman / Newman



Figure 3.17: Postman

Postman est un outil de test d'API qui offre une interface intuitive pour créer, partager et tester des requêtes HTTP. Son principal atout est son intégration fluide avec les scripts JavaScript, permettant aux utilisateurs d'automatiser les tests et d'ajouter des assertions directement dans l'interface de Postman.

De plus, Postman permet d'exporter les collections en fichiers JSON, qui peuvent être facilement manipulés et intégrés dans notre Framework. Cela le rend particulièrement adapté pour notre Framework, en facilitant l'écriture, l'exécution de tests complexes et la gestion des collections de tests de manière efficace.



Figure 3.18: Newman

Newman est un outil en ligne de commande qui permet d'exécuter des collections de tests Postman en dehors de l'interface graphique de Postman. Son principal avantage réside dans sa capacité à intégrer ces tests dans des pipelines d'intégration continue et de déploiement continu (CI/CD).

En utilisant Newman, nous pouvons automatiser l'exécution des tests Postman directement depuis notre Framework, ce qui est particulièrement utile pour garantir que les tests sont exécutés de manière cohérente et fiable à chaque déploiement. Newman facilite également la génération de rapports détaillés sur les résultats des tests, ce qui permet une analyse approfondie et une prise de décision éclairée.

3.1.4 Gitlab CI



Figure 3.19: GitLab CI

GitLab CI est une plateforme d'intégration et de déploiement continu (CI/CD) qui permet d'automatiser l'ensemble du cycle de vie du développement logiciel. Dans le contexte de notre Framework, GitLab CI joue un rôle crucial en orchestrant l'exécution automatique des tests à chaque phase de déploiement.

En intégrant GitLab CI, nous pouvons configurer des pipelines qui automatisent le processus de test des API, garantissant ainsi une validation continue et une livraison rapide des fonctionnalités. Cela permet non seulement de maintenir la qualité et la stabilité des API, mais aussi de réduire les délais de mise en production en détectant et en corrigeant rapidement les éventuels problèmes.

3.2 Conception du Framework

3.2.1 Planification et diagramme de GANTT

La planification du projet de Framework pour l'automatisation des tests d'API au sein du projet Pléiade comprend trois étapes clés :

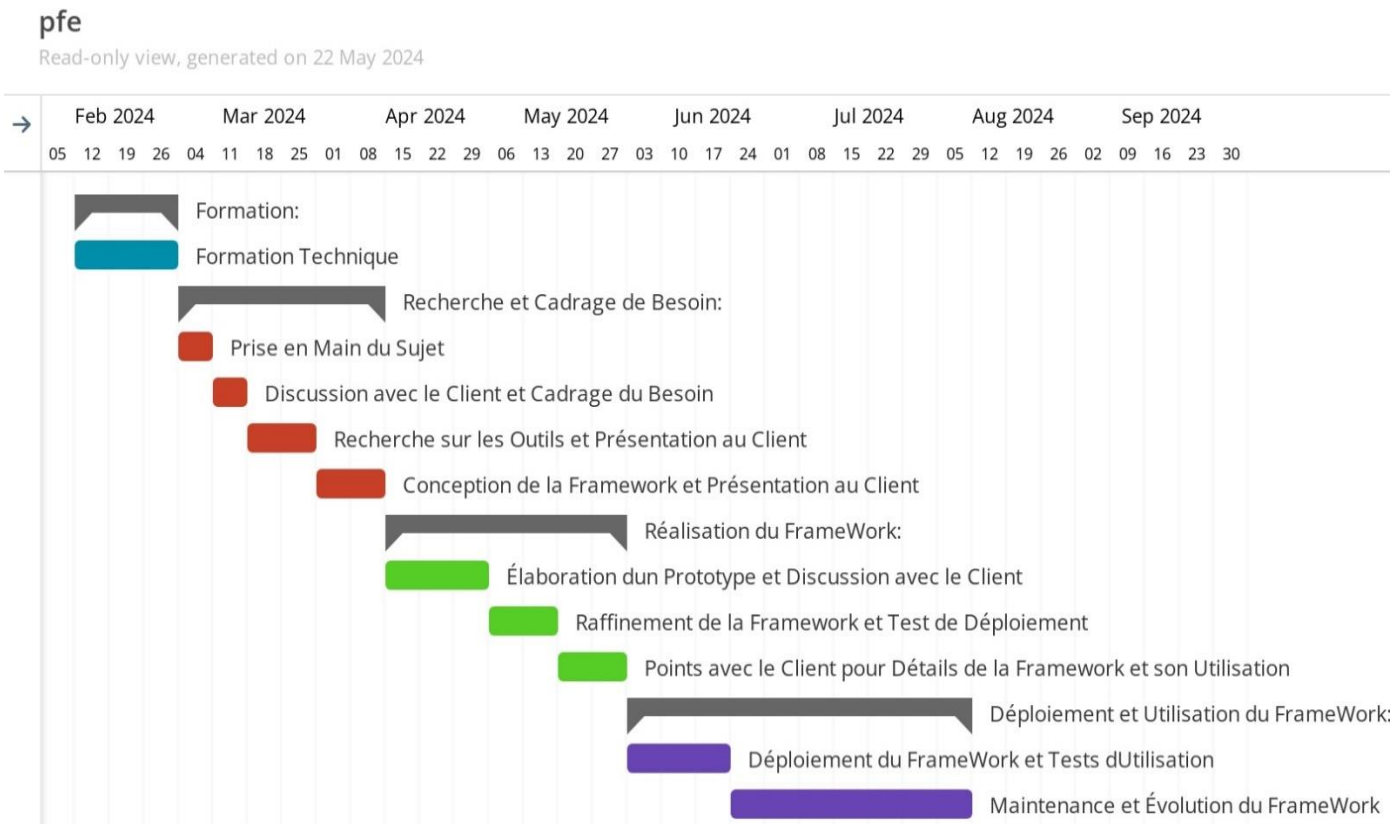


Figure 3.20: Diagramme de GANTT

Id	Name	Start	Due	Duration (Cal. Days)
1	Formation:	05/02/2024	01/03/2024	26
1.1	Formation Technique	05/02/2024	01/03/2024	26
2	Recherche et Cadrage de Besoin:	04/03/2024	12/04/2024	40
2.1	Prise en Main du Sujet	04/03/2024	08/03/2024	5
2.2	Discussion avec le Client et Cadrage du Besoin	11/03/2024	15/03/2024	5
2.3	Recherche sur les Outils et Présentation au Client	18/03/2024	29/03/2024	12
2.4	Conception de la Framework et Présentation au Client	01/04/2024	12/04/2024	12
3	Réalisation du Framework:	15/04/2024	31/05/2024	47
3.1	Élaboration d'un Prototype et Discussion avec le Client	15/04/2024	03/05/2024	19
3.2	Raffinement de la Framework et Test de Déploiement	06/05/2024	17/05/2024	12
3.3	Points avec le Client pour Détails de la Framework et son Utilisation	20/05/2024	31/05/2024	12
4	Déploiement et Utilisation du Framework:	03/06/2024	05/08/2024	64
4.1	Déploiement du Framework et Tests d'Utilisation	03/06/2024	21/06/2024	19
4.2	Maintenance et Évolution du Framework	24/06/2024	05/08/2024	43

Tableau 3.2: Tableau de GANTT

1. Recherche et Cadrage des Besoins :

- Identification des besoins spécifiques en matière d'automatisation des tests d'API, y compris l'évaluation des outils disponibles et la définition des fonctionnalités requises.
- Analyse approfondie des défis actuels liés aux tests d'API, notamment la gestion des nombreux endpoints et acquérir une compréhension du métier fonctionnel du projet Pléiade.
- Définition des objectifs du Framework, tels que l'automatisation de la maintenance des API, l'exécution automatique des tests, et la génération de rapports.

2. Réalisation du Framework :

- Développement du Framework en utilisant Node.js comme plateforme principale, intégrant Postman et Newman pour l'automatisation des tests d'API.
- Implémentation des fonctionnalités clés telles que la gestion des jeux de données de test, l'exécution régulière des tests, et la génération de rapports détaillés sur les résultats.
- Effectuer des points réguliers avec les chefs de projets et membres de l'équipe pour optimiser et consulter sur les différents aspects du Framework.

3. Déploiement et Utilisation du Framework :

- Intégration du Framework dans GitLab CI pour automatiser les tests à chaque déploiement, assurant une validation continue des API.
- Formation des équipes sur l'utilisation du Framework et sur les bonnes pratiques d'automatisation des tests d'API.
- Déploiement progressif du Framework dans l'environnement de développement et de production, avec une surveillance continue des performances et de l'efficacité des tests.

Cette approche structurée vise à assurer le succès du projet en optimisant la gestion des tests d'API, en réduisant les erreurs manuelles et en accélérant les cycles de développement et de déploiement des fonctionnalités dans le projet Pléiade.

3.2.2 Architecture du Framework

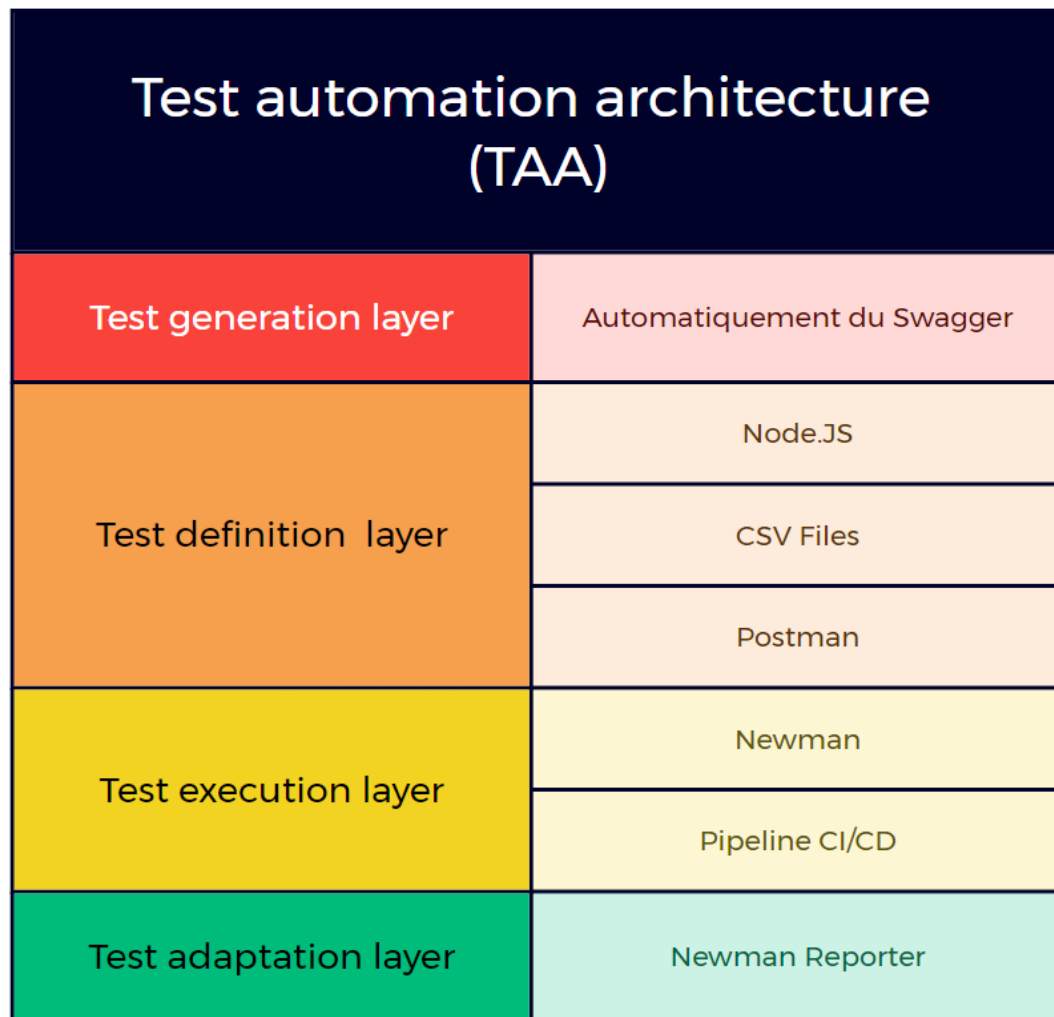


Figure 3.21: Test Automation Architecture

L'architecture du Framework que nous avons développé suit le standard TAA (Test Automation Architecture), organisée en quatre couches distinctes pour optimiser l'automatisation des tests d'API.

1. Test Generation (Génération des Tests) :

- Cette couche se concentre sur la génération automatique des scripts de test à partir des spécifications des API. Utilisant des outils comme Swagger, elle extrait les endpoints, les paramètres et les réponses attendues pour créer des tests cohérents et complets.

2. Test Definition (Définition des Tests) :

- Ici, les tests générés sont définis et structurés de manière à être facilement compréhensibles et maintenables. Utilisant Node.js et Postman, cette couche permet de manipuler les collections de tests, d'ajouter des assertions et des validations pour garantir la conformité des résultats des API avec les attentes spécifiées.

3. Test Execution (Exécution des Tests) :

- La couche d'exécution des tests utilise des outils comme Newman pour automatiser l'exécution des collections de tests Postman. Cette automatisation permet une validation continue des API à chaque étape du processus de développement, en utilisant l'environnement de GitLab CI pour assurer une intégration continue fluide et fiable.

4. Test Adaption (Adaptation des Tests) :

- Enfin, la couche d'adaptation des tests utilise Newman Reporter pour générer des rapports détaillés sur les résultats des tests. Cela permet d'analyser facilement les performances des API et de prendre des décisions informées pour ajuster les tests en fonction des évolutions des API ou des nouvelles fonctionnalités.

Cette architecture en couches facilite la gestion, l'automatisation et la maintenance des tests d'API dans notre projet, assurant ainsi une qualité et une fiabilité constantes des services exposés par les API du projet Pléiade.

3.2.3 Diagramme de Séquence

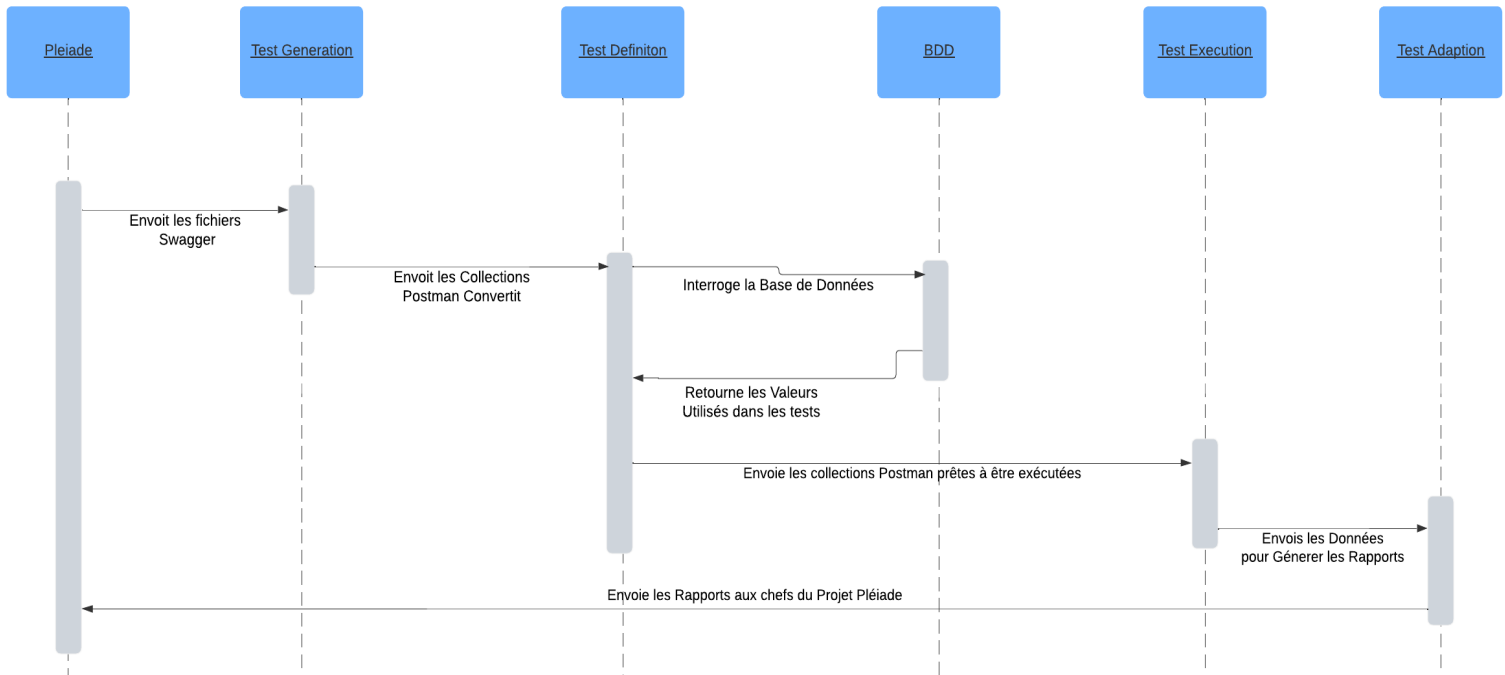


Figure 3.22: Diagramme de Séquence

3.3 Approche de Développement

L'approche de développement de notre Framework repose sur plusieurs principes clés pour assurer son optimisation continue :

- On réalise un test continu des différents workflows et méthodes pour optimiser le Framework. Cela inclut l'identification des goulots d'étranglement, l'amélioration des performances des tests, et l'exploration de nouvelles technologies ou pratiques pour renforcer l'efficacité globale du processus d'automatisation des tests d'API.
- On maintient une communication constante avec le client et les chefs de projet afin de comprendre leurs besoins évolutifs et de m'assurer que le Framework répond aux exigences spécifiques du projet Pléiade. Cela nous permet d'ajuster les fonctionnalités du Framework en fonction des retours et des nouvelles exigences du client.

- On reste ouvert aux conseils et aux contributions des internes d'Intelcia, en favorisant un environnement de collaboration et d'échange d'idées. Cette approche me permet de bénéficier de l'expertise interne et de développer des solutions adaptées et innovantes pour l'automatisation des tests d'API dans le contexte spécifique de mon projet.

Cette approche vise constamment à améliorer le Framework, assurant ainsi une optimisation continue des processus d'automatisation des tests d'API.

3.4 Fonctionnement du Framework

Passons maintenant au fonctionnement du Framework, ou on va voir comment le Framework gère les différents aspects de l'automatisation en plus des interactions des différents composants de mon Framework.

3.4.1 Workflow du Framework

WORKFLOW

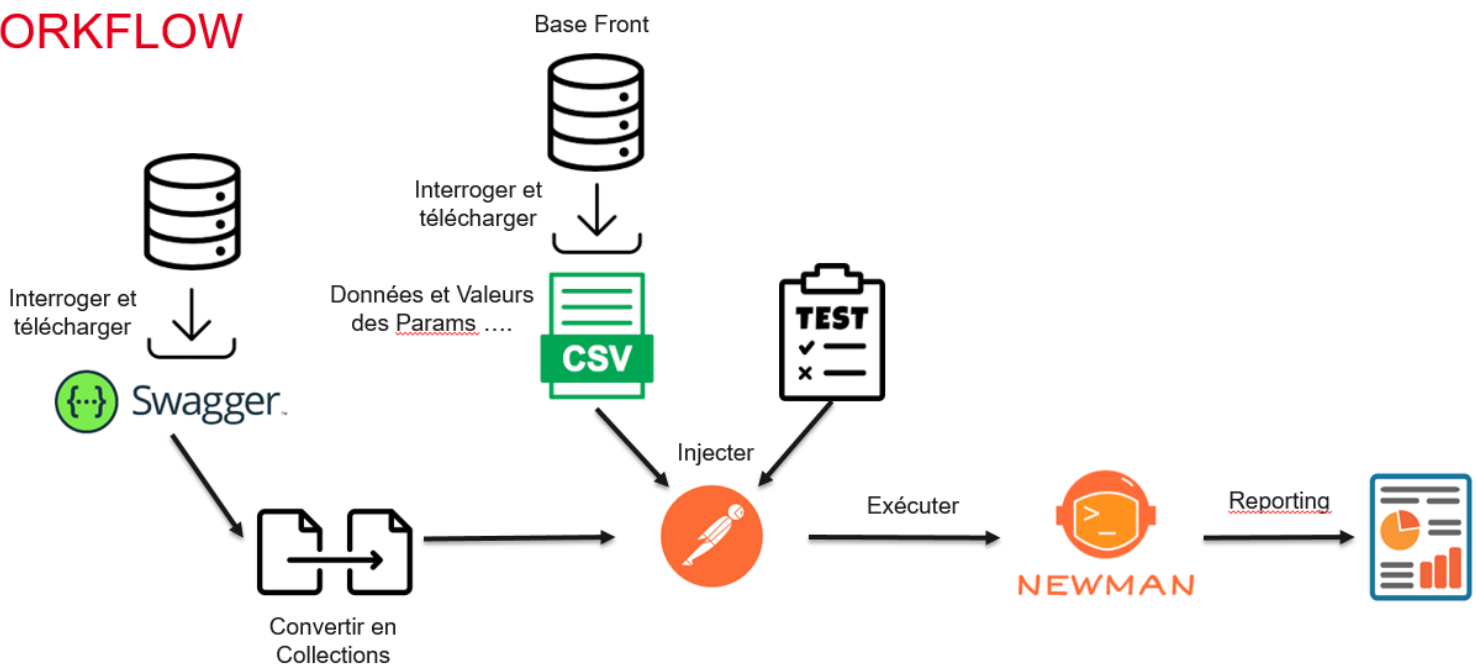


Figure 3.23: Workflow du Framework

Voici le workflow détaillé du Framework, ce processus s'applique à chaque collection de chaque module spécifique du projet Pléiade :

1. Téléchargement du Swagger depuis l'API Pléiade :

- Le fichier YAML contenant les spécifications Swagger est téléchargé directement depuis l'API Pléiade.

2. Conversion en Collection Postman avec la bibliothèque JavaScript :

- Utilisation d'une bibliothèque JavaScript compatible avec Postman pour convertir le Swagger téléchargé en une collection Postman prête à être utilisée.

3. Extraction des Valeurs de Test depuis la Base de Données :

- Node.js est utilisé pour extraire les données de test nécessaires à partir d'une base de données existante. Ces données sont ensuite formatées dans un fichier CSV.

4. Injection des Données dans la Collection Postman :

- Les données extraites du fichier CSV sont injectées dans la collection Postman afin de paramétrer dynamiquement les requêtes API avec des valeurs spécifiques.

5. Ajout d'Assertions aux Collections Postman :

- Des scripts Node.js sont intégrés pour ajouter des assertions aux requêtes de la collection Postman, garantissant ainsi que les réponses des API sont validées selon les critères définis.

6. Exécution des Tests et Collections avec Newman :

- Newman est utilisé pour automatiser l'exécution des collections Postman configurées. Cela assure une validation continue et systématique des API à chaque itération du test.

7. Génération de Rapports avec Newman :

- À la fin de l'exécution, Newman génère des rapports détaillés qui fournissent une analyse approfondie des résultats des tests, incluant les succès, les échecs et les performances des API testées.

PROCESS DE LA COLLECTION

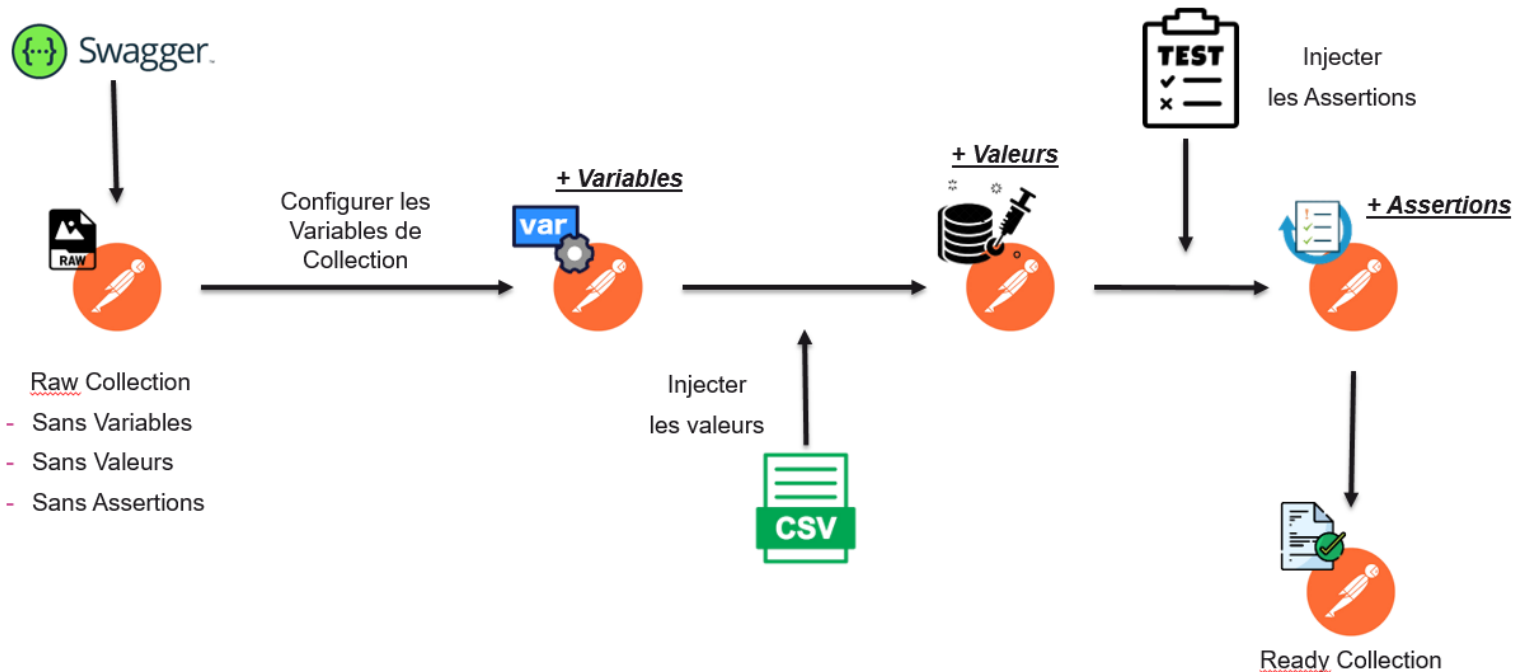


Figure 3.24: Processus Détaillé de comment la Collection est Traitée

Voici maintenant comment notre Framework traite les Collections Postman (qui sont en format JSON), essentiellement après être convertit depuis le Swagger chaque collection passe par 3 étapes distinctes :

1. **Étape 1 :** La collection est configurée avec des variables de collection pour que l'on puisse y injecter les valeurs qui seront utilisées pour les tests.
2. **Étape 2 :** On injecte les valeurs depuis le fichier csv qui contient les valeurs extraites depuis la base de données dans la collection Postman.
3. **Étape 3 :** On ajoute les assertions pour vérifier la validité des tests dans la collection.

Et ainsi on obtient une Collection Postman qui est éligible à être exécutée par Newman.

3.4.2 Composants du Framework

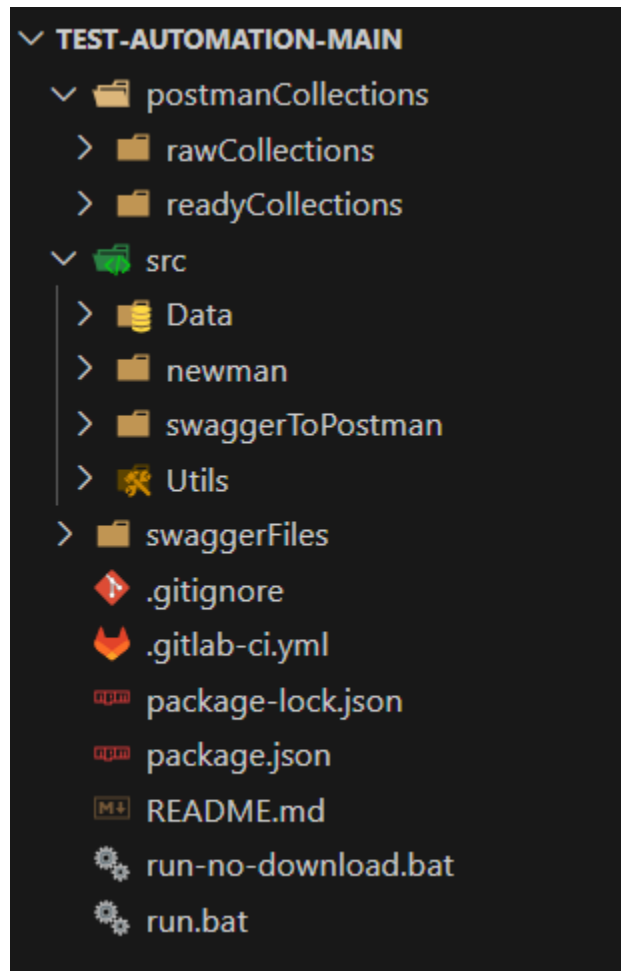


Figure 3.25: Architecture du Framework

Voici l'architecture de notre Framework, elle suit le principe de POO et elle contient 3 Package Principaux :

- SwaggerFiles : ou on va mettre les fichiers Swagger que l'on va télécharger via l'API de Pléiade.
- PostmanCollections : ou on va y mettre nos collections générées durant le cycle de vie de l'automatisation de test. rawCollections contient les collections qui sont fraîchement générées depuis le Swagger file , et readyCollections contient les collections traitées et prêtes à être exécutées.
- SRC : c'est le package qui contient notre code source.

Chaque Package dans le dossier src a des missions modulaires et distinctes comme suit :

- ❖ **Data** : qui s'occupe d'interroger la base de données pour y extraire les valeurs qui nous sont nécessaires et après va les stocker dans des fichiers csv dédiés.
- ❖ **Newman** : c'est le module qui s'occupe de l'exécution des tests et des collections Postman en plus de générer les rapports des tests.
- ❖ **Utils** : c'est le module qui nous aide dans la manipulation des fichiers et dossiers.
- ❖ **SwaggerToPostman** : celui-ci est le module qui s'occupe de tout ce qui touche aux fichiers Swagger et au traitement des collections Postman. Cela comprend :
 - Téléchargement des Swagger.
 - Conversion des Swagger en collections Postman.
 - Configuration des collections Postman.
 - Injections des valeurs depuis les fichiers CSV.
 - Traiter les collections pour les rendre exécutables.

3.4.3 Les Interactions entre les Composants

Nous allons maintenant entrer dans les details de chaque module :

- ❖ Data :

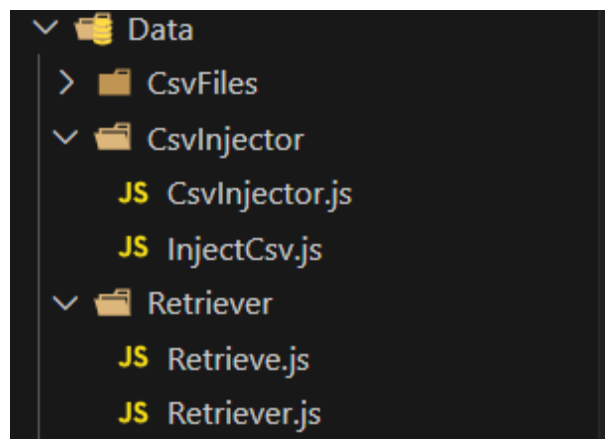


Figure 3.26: Le Module Data

Comme on peut voir, le module Data contient 2 packages :

- **Retrieveur** : c'est la classe qui s'occupe d'interroger la base de données et extraire les données.
- **CsvInjector** : c'est la classe qui s'occupe de générer les fichiers csv qui contiennent nos données extraites
- **CsvFiles** : c'est le dossier qui contient nos fichiers CSV générés.

❖ SwaggerToPostman :

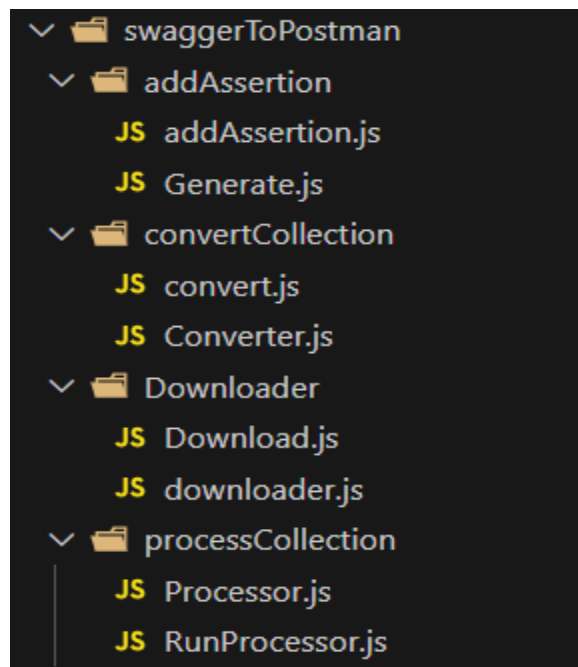


Figure 3.27: Le Module SwaggerToPostman

Le module SwaggerToPostman est celui qui passe tous les traitements nécessaires pour passer d'un fichier Swagger a une collection Postman exécutable, il contient 4 packages :

- **Downloader** : c'est la classe qui s'occupe de télécharger les fichiers Swagger et de les mettre à jour périodiquement.
- **ConvertCollection** : c'est la classe qui s'occupe de convertir les fichiers Swagger en collections Postman.
- **ProcessCollection** : c'est la classe qui s'occupe de configurer les variables de la collection en plus de l'injecter avec les données depuis les fichiers csv.

- **AddAssertion** : c'est la classe qui s'occupe d'ajouter les assertions et les tests à nos collections Postman.

❖ Newman :



Figure 3.28: Le Module Newman

Le module Newman s'occupe d'exécuter les collections Postman avec les tests, en plus de générer les rapports pour chaque module Pléiade, il contient 2 packages :

- **Runner** : c'est la classe qui s'occupe d'exécuter nos tests.
- **Reporter** : c'est la classe qui s'occupe de générer les rapports.

CONCLUSION

En conclusion du Chapitre 3, nous avons parcouru en détail la conception et la réalisation du Framework d'automatisation des tests d'API pour le projet Pléiade. Cette étape a nécessité une analyse minutieuse des besoins, une planification rigoureuse des fonctionnalités, ainsi que des choix judicieux en termes de technologies et d'outils. La mise en œuvre concrète du Framework représente une avancée significative dans l'amélioration de la qualité des services de Pléiade.

Le prochain chapitre se concentrera sur la présentation du Framework en action, mettant en lumière ses fonctionnalités clés, ses performances et son impact sur le processus de tests d'API au sein du projet.

Chapitre 4:Présentation de la Solution

INTRODUCTION

Dans ce Chapitre, nous mettons en lumière le fruit de notre travail : le Framework d'automatisation des tests d'API pour le projet Pléiade. À travers des captures d'écran fonctionnelles, nous illustrerons les différentes fonctionnalités et l'interface utilisateur du Framework en action. Cette présentation détaillée permettra de comprendre le fonctionnement pratique du Framework et son impact sur l'efficacité des tests d'API au sein de Pléiade.

4.1 Présentation des Étapes de l'Automatisation

Pour nos exemples et captures d'écrans dans cette partie, on a opté pour utiliser des données qui sont bidons pour ne pas enfreindre la confidentialité des données de nos clients, d'où les différents fails. Mais le workflow reste le même dans le cas nominal.

4.1.1 Téléchargement des Swagger

Pour télécharger les fichiers Swagger, on utilise la commande **npm run download**, les fichiers Swagger sont stockés dans le dossier SwaggerFiles. Ce téléchargement est essentiel car il permet à notre Framework de générer automatiquement des scripts de test à partir de ces spécifications, facilitant ainsi la mise en place des tests d'API. Grâce à cette fonctionnalité, les développeurs peuvent rapidement intégrer de nouveaux services et assurer la cohérence des tests tout au long du cycle de développement.

```
C:\Users\061654\Desktop\test-automation>npm run download

> test-automation@1.0.0 download
> node ./src/swaggerToPostman/Downloader/Download.js

File downloaded: admin
File downloaded: admins-requests
File downloaded: alerting
File downloaded: appointments
File downloaded: billboard
File downloaded: commands-tracking
File downloaded: console
File downloaded: customer-cases
File downloaded: customer-tickets
File downloaded: documents
File downloaded: eligibility
File downloaded: invoices
File downloaded: mobile-tracking
File downloaded: notifications
File downloaded: notifications-service-api
File downloaded: ott-services
File downloaded: parc-admin
File downloaded: parc-convergent
File downloaded: parc-fixe
File downloaded: parc-mobile
File downloaded: parc-service
File downloaded: portability
File downloaded: pre-sales
File downloaded: recife-api
File downloaded: shared-references
File downloaded: traces
File downloaded: umbrella
File downloaded: users
```

Figure 4.29: Execution de npm run download

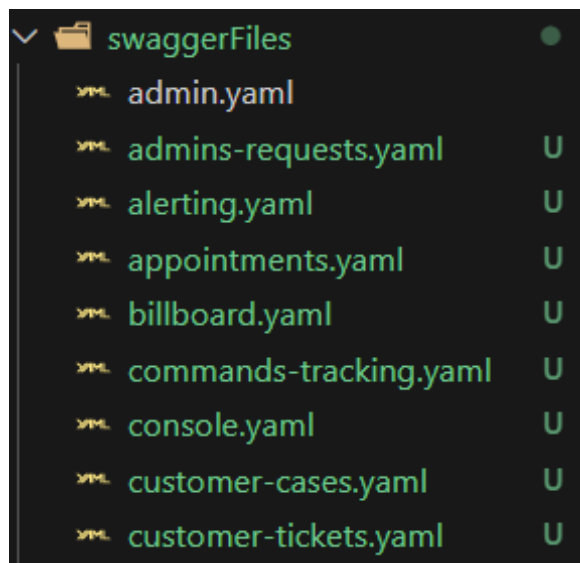


Figure 4.30: Les Swagger Téléchargés

4.1.2 Conversion des Swagger en Collections

Pour convertir les Swagger en Collections Postman on utilise la commande **`npm run convert`**, les collections sont stockées dans `PostmanCollection/rawCollection`. Grâce à cette conversion, notre Framework peut générer automatiquement des scénarios de test à partir des endpoints de l'API, facilitant ainsi la création et la gestion des tests d'API sans nécessiter de codage manuel.

```
C:\Users\061654\Desktop\test-automation>npm run convert

> test-automation@1.0.0 convert
> node ./src/swaggerToPostman/convertCollection/convert.js

Current directory filenames:
input paths:..\test-automation\swaggerFiles\admin.yaml
output paths:..\test-automation\postmanCollections\rawCollections\admin.json
-----
input paths:..\test-automation\swaggerFiles\admins-requests.yaml
output paths:..\test-automation\postmanCollections\rawCollections\admins-requests.json
-----
input paths:..\test-automation\swaggerFiles\alerting.yaml
output paths:..\test-automation\postmanCollections\rawCollections\alerting.json
-----
input paths:..\test-automation\swaggerFiles\appointments.yaml
output paths:..\test-automation\postmanCollections\rawCollections\appointments.json
-----
input paths:..\test-automation\swaggerFiles\billboard.yaml
output paths:..\test-automation\postmanCollections\rawCollections\billboard.json
-----
input paths:..\test-automation\swaggerFiles\commands-tracking.yaml
output paths:..\test-automation\postmanCollections\rawCollections\commands-tracking.json
-----
input paths:..\test-automation\swaggerFiles\console.yaml
output paths:..\test-automation\postmanCollections\rawCollections\console.json
-----
input paths:..\test-automation\swaggerFiles\customer-cases.yaml
output paths:..\test-automation\postmanCollections\rawCollections\customer-cases.json
-----
input paths:..\test-automation\swaggerFiles\customer-tickets.yaml
output paths:..\test-automation\postmanCollections\rawCollections\customer-tickets.json
-----
```

Figure 4.31:Execution de `npm run convert`

```
Command output: Input file: C:\Users\061654\Desktop\test-automation\swaggerFiles\portability.yaml
Writing to file: true C:\Users\061654\Desktop\test-automation\postmanCollections\rawCollections\portability.json {
  result: true,
  output: [ { type: 'collection', data: [Object] } ],
  analytics: {}
}
Conversion successful, collection written to file

Command output: Input file: C:\Users\061654\Desktop\test-automation\swaggerFiles\notifications-service-api.yaml

Command output: Input file: C:\Users\061654\Desktop\test-automation\swaggerFiles\console.yaml
Writing to file: true C:\Users\061654\Desktop\test-automation\postmanCollections\rawCollections\console.json {
  result: true,
  output: [ { type: 'collection', data: [Object] } ],
  analytics: {}
}
Conversion successful, collection written to file

Command output: Input file: C:\Users\061654\Desktop\test-automation\swaggerFiles\umbrella.yaml
Writing to file: true C:\Users\061654\Desktop\test-automation\postmanCollections\rawCollections\umbrella.json {
  result: true,
  output: [ { type: 'collection', data: [Object] } ],
  analytics: {}
}
Conversion successful, collection written to file

Command output: Input file: C:\Users\061654\Desktop\test-automation\swaggerFiles\eligibility.yaml
Writing to file: true C:\Users\061654\Desktop\test-automation\postmanCollections\rawCollections\eligibility.json {
  result: true,
  output: [ { type: 'collection', data: [Object] } ],
  analytics: {}
}
Conversion successful, collection written to file

Command output: Input file: C:\Users\061654\Desktop\test-automation\swaggerFiles\traces.yaml
Writing to file: true C:\Users\061654\Desktop\test-automation\postmanCollections\rawCollections\traces.json {
  result: true,
  output: [ { type: 'collection', data: [Object] } ],
  analytics: {}
}
Conversion successful, collection written to file
```

Figure 4.32: Conversion des Swagger en Collection

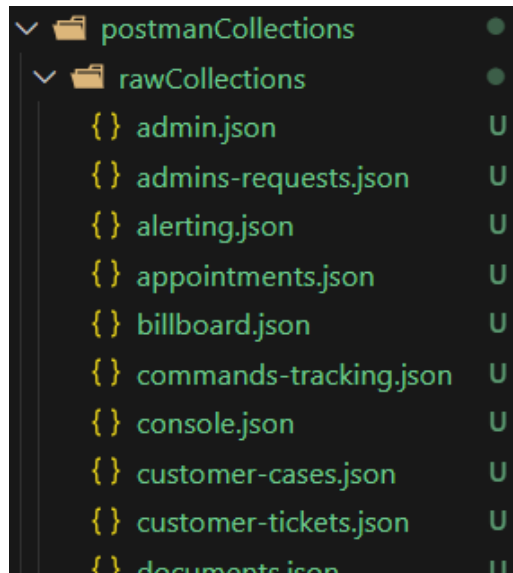


Figure 4.33: Les Collections Convertis

4.1.3 Configuration des Collections

Pour configurer les Collections et y ajouter les variables on utilise la commande **npm run process**. Cette étape est cruciale car elle permet à notre outil d'automatisation des tests de définir les paramètres nécessaires pour chaque collection, tels que les variables d'environnement, les headers, les authentications, etc. La configuration des collections garantit ainsi que les tests seront exécutés dans un environnement contrôlé et conforme aux spécifications de l'API.

```
C:\Users\061654\Desktop\test-automation>npm run process

> test-automation@1.0.0 process
> node ./src/swaggerToPostman/processCollection/Process.js

Current directory filenames:
Updated collection saved to ..\test-automation\postmanCollections\rawCollections\test_keep.json
File ..\test-automation\src\Data\CsvFiles\GeneratedCsv\test_keep_data.csv already exists. Skipping write operation.
input paths:..\test-automation\postmanCollections\rawCollections\test_keep.json
output paths:..\test-automation\src\Data\CsvFiles\GeneratedCsv\test_keep_data.csv
Map(10) {
  'X-User' => '',
  'X-User-Substitute' => '',
  'WS-ConsoAuthent' => '',
  'X-Point-Of-Sale' => '',
  'X-Company' => '',
  'Accept' => '',
  'username' => '',
  'password' => '',
  'offerTypes' => '',
  'configured' => '',
  'manufacturerIds' => '',
  'serviceAgreements' => '',
  'articleNumbers' => '',
  'status' => '',
  'idSupport' => '',
  'OmniSearch' => ''
}

-----
Updated collection saved to ..\test-automation\postmanCollections\rawCollections\portability.json
Map data has been written to ..\test-automation\src\Data\CsvFiles\GeneratedCsv\portability_data.csv
input paths:..\test-automation\postmanCollections\rawCollections\portability.json
output paths:..\test-automation\src\Data\CsvFiles\GeneratedCsv\portability_data.csv
Map(13) {
  'X-User' => '',
  'X-User-Substitute' => '',
  'WS-ConsoAuthent' => '',
  'X-Point-Of-Sale' => '',
  'X-Company' => '',
  'Content-Type' => '',
  'Accept' => '',
  'username' => '',
  'password' => '',
  'portaDate' => '',
  'slot' => '',
  'rio' => '',
  'msisdn' => ''
}

-----
Updated collection saved to ..\test-automation\postmanCollections\rawCollections\pre-sales.json
Map data has been written to ..\test-automation\src\Data\CsvFiles\GeneratedCsv\pre-sales_data.csv
input paths:..\test-automation\postmanCollections\rawCollections\pre-sales.json
output paths:..\test-automation\src\Data\CsvFiles\GeneratedCsv\pre-sales_data.csv
Map(18) {
  'X-User' => '',
  'X-User-Substitute' => '',
  'WS-ConsoAuthent' => '',
  'X-Point-Of-Sale' => '',
  'X-Company' => '',
  'Content-Type' => '',
  'Accept' => '',
  'username' => '',
  'password' => '',
  'portaDate' => '',
  'slot' => '',
  'rio' => '',
  'msisdn' => '',
  'offerTypes' => '',
  'configured' => '',
  'manufacturerIds' => '',
  'serviceAgreements' => ''
}
```

Figure 4.35:Execution de npm run process

4.1.4 Génération des fichiers CSV

Pour extraire les données depuis la Base de Données et les injecter dans des fichiers csv on utilise la commande **npm run inject**, les fichiers csv sont stockés dans Data/csvFiles. Cette fonctionnalité permet de créer des fichiers CSV contenant les valeurs des paramètres de test, tels que les identifiants, les tokens d'authentification, les données d'entrée, etc. Ces fichiers CSV servent ensuite de sources de données pour les scénarios de test, permettant ainsi une gestion efficace et modulaire des valeurs des paramètres.


```

C:\Users\061654\Desktop\test-automation>npm run inject

> test-automation@1.0.0 inject
> node ./src/Data/CsvInjector/InjectCsv.js

Current directory filenames:
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\admins-requests_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\admin_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\alerting_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\appointments_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\billboard_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\commands-tracking_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\console_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\customer-cases_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\customer-tickets_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\documents_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\eligibility_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\invoices_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\mobile-tracking_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\notifications_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\ott-services_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\parc-admin_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\parc-convergent_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\parc-fixe_data.csv
injected..\test-automation\src\Data\CsvFiles\GeneratedCsv\parc-mobile_data.csv

```

Figure 4.36: Execution de `npm run inject`

4.1.5 Injection des Valeurs dans les Collections

Pour injecter les valeurs depuis les csv vers leurs collections respectives on utilise la commande **npm run populate**, les collections ainsi étant totalement configurées sont stockées dans PostmanCollections/readyCollections. Cette fonctionnalité permet à notre outil d'automatisation des tests d'API d'utiliser dynamiquement les valeurs des paramètres lors de l'exécution des scénarios de test. L'injection des valeurs des paramètres garantit ainsi une exécution personnalisée et précise des tests, en prenant en compte les données spécifiques nécessaires à chaque requête API.

```
C:\Users\061654\Desktop\test-automation>npm run populate

> test-automation@1.0.0 populate
> node ./src/swaggerToPostman/processCollection/Populate.js

Current directory filenames:

Current directory filenames:
input paths:..\test-automation\postmanCollections\rawCollections\admin.json
output paths:..\test-automation\postmanCollections\readyCollections\admin.json
-----
Global

basic
```

Figure 4.37: Execution de npm run populate

```
'status' => '',
'ip' => '',
'omniSearch' => '',
'offset' => '',
'size' => '',
'sort' => '',
'Content-Type' => '',
'trace' => '',
'code' => '',
'excludeConnection' => '',
'' => ''
}
Updated collection saved to ..\test-automation\postmanCollections\readyCollections\umbrella.json
input paths:..\test-automation\postmanCollections\rawCollections\users.json
output paths:..\test-automation\postmanCollections\readyCollections\users.json
-----
Global

basic

undefined
Map(10) {
  'key' => '',
  'X-User' => '',
  'X-User-Substitute' => '',
  'WS-ConsoAuthent' => '',
  'X-Point-Of-Sale' => '',
  'X-Company' => '',
  'Accept' => '',
  'username' => '',
  'password' => '',
  '' => ''
}
Updated collection saved to ..\test-automation\postmanCollections\readyCollections\users.json
```

Figure 4.38: Injection des données dans les Collections

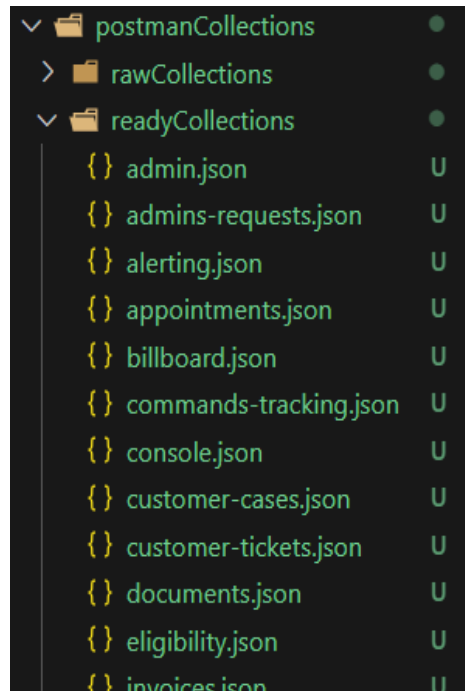


Figure 4.39: Les Collections Configurées avec les Valeurs

4.1.6 Ajout des Assertions dans les Collections

Pour ajouter les assertions et tests dans les collections on utilise la commande **npm run generate**, on fait tout simplement une mise à jour des collections de PostmanCollections/readyCollection. Les assertions sont des règles ou des conditions définies par les utilisateurs pour vérifier la validité des réponses reçues lors de l'exécution des tests d'API. Elles permettent de s'assurer que les réponses des requêtes correspondent aux attentes définies, en vérifiant des éléments tels que les codes de statut HTTP, les données renvoyées, les temps de réponse, etc.

```
C:\Users\061654\Desktop\test-automation>npm run generate

> test-automation@1.0.0 generate
> node ./src/swaggerToPostman/addAssertion/Generate.js

Current directory filenames:
Updated collection saved to ..\test-automation\postmanCollections\readyCollections\admin.json
-----
Updated collection saved to ..\test-automation\postmanCollections\readyCollections\admins-requests.json
-----
Updated collection saved to ..\test-automation\postmanCollections\readyCollections\alerting.json
-----
Updated collection saved to ..\test-automation\postmanCollections\readyCollections\appointments.json
-----
Updated collection saved to ..\test-automation\postmanCollections\readyCollections\billboard.json
-----
Updated collection saved to ..\test-automation\postmanCollections\readyCollections\commands-tracking.json
-----
Updated collection saved to ..\test-automation\postmanCollections\readyCollections\console.json
-----
Updated collection saved to ..\test-automation\postmanCollections\readyCollections\customer-cases.json
-----
Updated collection saved to ..\test-automation\postmanCollections\readyCollections\customer-tickets.json
-----
Updated collection saved to ..\test-automation\postmanCollections\readyCollections\documents.json
-----
Updated collection saved to ..\test-automation\postmanCollections\readyCollections\eligibility.json
-----
Updated collection saved to ..\test-automation\postmanCollections\readyCollections\invoices.json
-----
Updated collection saved to ..\test-automation\postmanCollections\readyCollections\mobile-tracking.json
-----
```

Figure 4.40: Execution de `npm run generate`

4.1.7 Exécution des Collections

Pour exécuter tous nos collections on utilise la commande **`npm run run`**. L'utilisation de Newman pour l'exécution des collections garantit une automatisation robuste, cohérente et fiable des tests d'API. Cette intégration avec Newman permet également une gestion plus efficace des tests en ligne de commande, facilitant ainsi l'intégration du processus de tests d'API dans les pipelines CI/CD.

```
C:\Users\061654\Desktop\test-automation>npm run run

> test-automation@1.0.0 run
> node ./src/newman/Runner/Run.js

Current directory filenames:
..\test-automation\src\Data\CsvFiles\Transposed\admin_data.csv
newman

admin

Iteration 1/2

□ admin / termination-reasons / {id}
└─ update Termination Reason
  PUT http://admin.pleiade-service.ctnl.pic.services.pack/admin/v1.0/admin/termination-reasons/ [errored]
  getaddrinfo ENOTFOUND admin.pleiade-service.ctnl.pic.services.pack

└─ delete Termination Reason
  DELETE http://admin.pleiade-service.ctnl.pic.services.pack/admin/v1.0/admin/termination-reasons/ [errored]
  getaddrinfo ENOTFOUND admin.pleiade-service.ctnl.pic.services.pack

□ admin / termination-reasons
└─ find Termination Reasons
  GET http://admin.pleiade-service.ctnl.pic.services.pack/admin/v1.0/admin/termination-reasons?offerTypes=&configured= [errored]
  getaddrinfo ENOTFOUND admin.pleiade-service.ctnl.pic.services.pack

└─ create Termination Reason
  POST http://admin.pleiade-service.ctnl.pic.services.pack/admin/v1.0/admin/termination-reasons [errored]
  getaddrinfo ENOTFOUND admin.pleiade-service.ctnl.pic.services.pack

□ admin / offers / {offerId}
└─ get Offer
  GET http://admin.pleiade-service.ctnl.pic.services.pack/admin/v1.0/admin/offers/ [errored]
  getaddrinfo ENOTFOUND admin.pleiade-service.ctnl.pic.services.pack
```

Figure 4.41: Execution de npm run run

	executed	failed
iterations	2	0
requests	36	36
test-scripts	0	0
prerequest-scripts	0	0
assertions	0	0
total run duration: 13.6s		
total data received: 0B (approx)		

#	failure	detail
01.	Error iteration: 1	getaddrinfo ENOTFOUND admin.pleiade-service.ctnl.pic.services.pack at request inside "admin / termination-reasons / {id} / update Termination Reason"
02.	Error iteration: 1	getaddrinfo ENOTFOUND admin.pleiade-service.ctnl.pic.services.pack at request inside "admin / termination-reasons / {id} / delete Termination Reason"
03.	Error iteration: 1	getaddrinfo ENOTFOUND admin.pleiade-service.ctnl.pic.services.pack at request inside "admin / termination-reasons / find Termination Reasons"
04.	Error iteration: 1	getaddrinfo ENOTFOUND admin.pleiade-service.ctnl.pic.services.pack at request inside "admin / termination-reasons / create Termination Reason"

Figure 4.42: Le rapport local dans la CLI

4.1.8 Génération des Rapports

Les rapports sont générés simultanément depuis **npm run run** et sont sous format html, Les rapports générés par newman contiennent plusieurs informations utiles et complètes tels :

- Le nombre total des requêtes http, des tests et des itérations de tests.
- Le nombre des assertions passed et failed.
- Les détails de chaque requête et son exécution.
- Pourquoi un test ne passe pas pour chaque assertion.

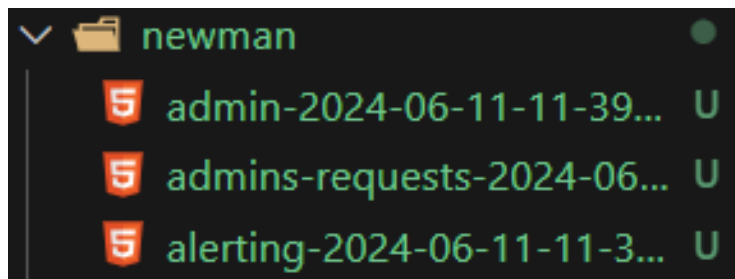


Figure 4.43: Les rapports Générés dans le dossier Newman

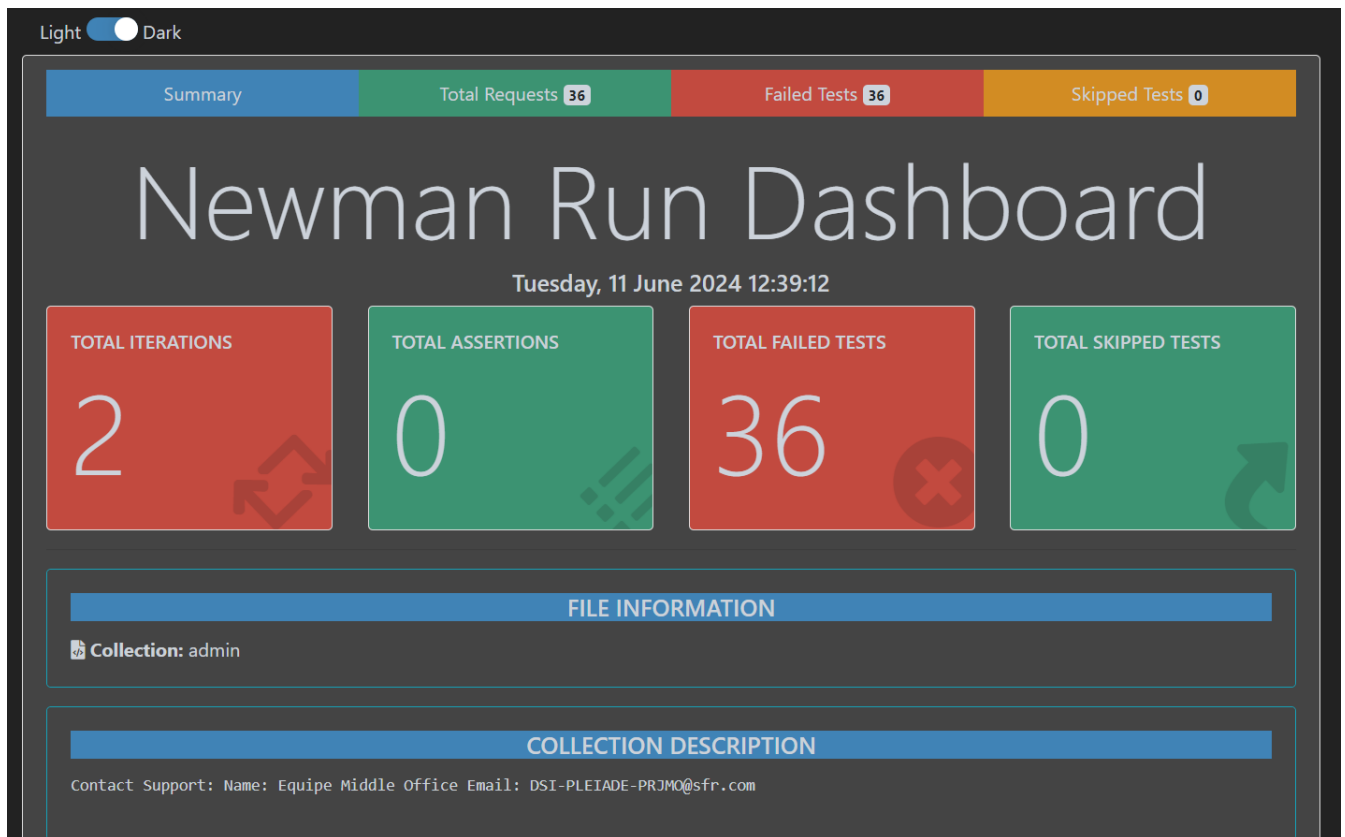


Figure 4.44: L'interface Principale du Rapport

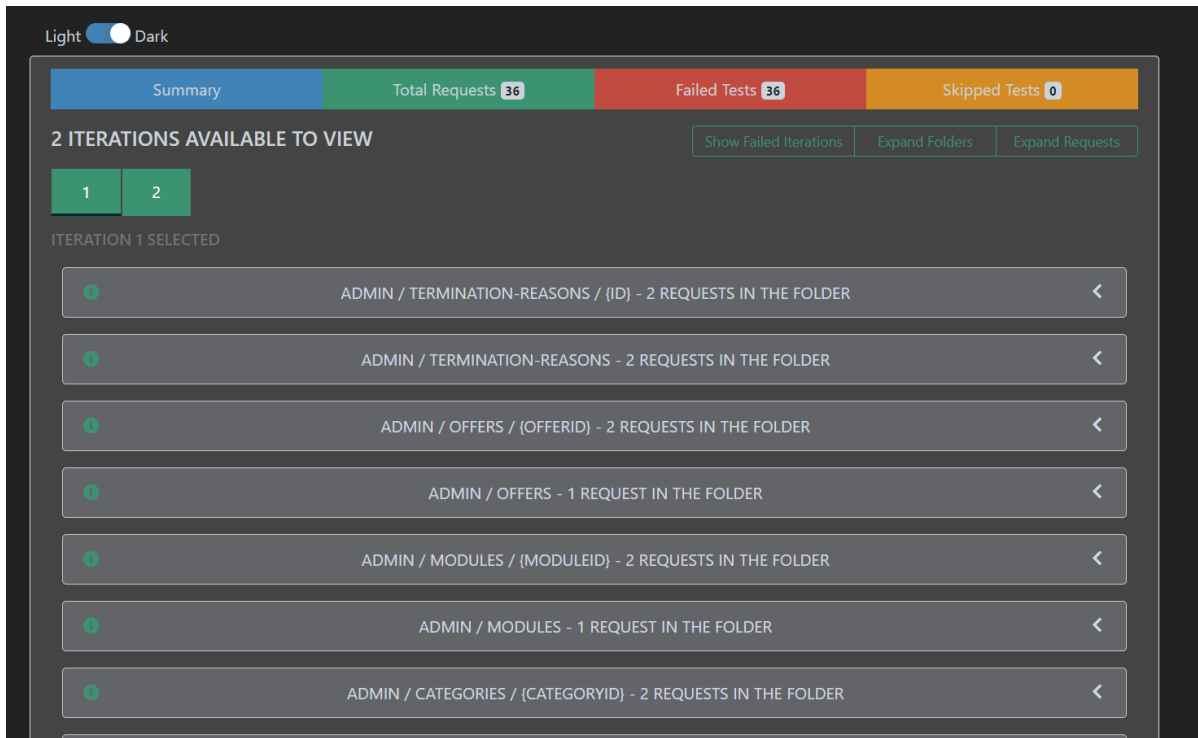


Figure 4.45: L'Onglet Total Requests

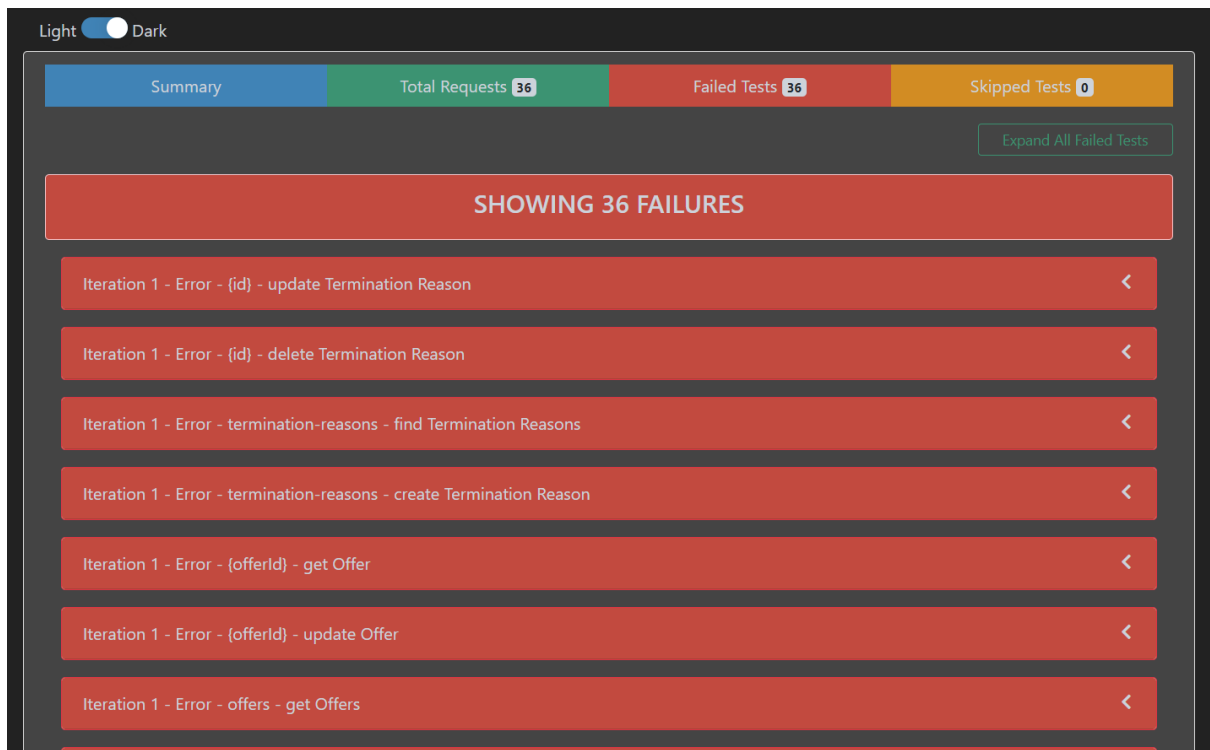


Figure 4.46: L'Onglet Failed Tests

4.2 Présentation de la Pipeline CI/CD

La pipeline CI/CD (Continuous Integration/Continuous Deployment) est un processus automatisé qui permet d'intégrer et de déployer les modifications de code de manière continue et efficace. Elle comprend plusieurs étapes, dont la **construction**, le **test**, et le **déploiement** du code, garantissant ainsi que les nouvelles modifications sont correctement intégrées et fonctionnelles avant d'être mises en production.

De plus, le principe du **scheduler** permet de planifier l'exécution automatique de la pipeline à des intervalles réguliers, assurant ainsi une vérification continue et proactive du système.

Voici une explication de son principe de fonctionnement à travers le script ci-dessous :

```
stages:
  - build

convert_process_generate_run:
  tags:
    - valentine-ipp1
  stage: build

  script:
    - npm install
    - export NO_PROXY=prod,pack,10.0.0.0/8,localhost,127.0.0.1,sfr.com,*.ctn1.pic.services.pack,*.pfv.private.sfr.com
    - export no_proxy=prod,pack,10.0.0.0/8,localhost,127.0.0.1,sfr.com,*.ctn1.pic.services.pack,*.pfv.private.sfr.com
    - npm run download
    - npm run convert
    - npm run process
    - npm run inject
    - npm run populate
    - npm run generate
    - npm run run
  artifacts:
    paths:
      - ./newman
```

Figure 4.47: Script de la Pipeline CI/CD

Dans cette configuration, plusieurs commandes **npm** sont utilisées pour automatiser différentes étapes du processus :

- **npm install** : Installe les dépendances nécessaires pour le projet.
- **npm run download** : Télécharge les spécifications Swagger depuis l'API Pléiade.
- **npm run convert** : Convertit les spécifications Swagger en une collection Postman.

- **npm run process** : Traite la collection Postman.
- **npm run inject** : Injecte les données de test dans la collection Postman.
- **npm run populate** : Remplit la collection avec des données supplémentaires si nécessaire.
- **npm run generate** : Génère les tests basés sur la collection Postman.
- **npm run run** : Exécute les tests avec Newman.

Les rapports générés par Newman sont sauvegardés dans le répertoire `./newman` en tant qu'artifacts pour une analyse ultérieure. En configurant le scheduler, la pipeline peut être planifiée pour s'exécuter automatiquement à des intervalles réguliers, comme quotidiennement ou hebdomadairement, permettant une vérification continue et proactive du système.

Voici comment les différentes exécution (aussi appelée Job) de la pipeline sont organisés au sein de la plateforme Gitlab CI, ainsi que les détails de l'exécution :





















Status	Job	Pipeline	Stage	Name	Duration	Coverage
	#6761569 main -> e11a81a1 valentine-ipp1	#1127593 created by 	build	convert_process_ge...	⌚ 00:00:45	
	#6735755 main -> d6acbdeb valentine-ipp1	#1123628 created by 	build	convert_process_ge...	⌚ 00:36:27 📅 4 days ago	 
	#6732840 main -> 895265a1 valentine-ipp1	#1123216 created by 	build	convert_process_ge...	⌚ 00:40:01 📅 4 days ago	 
	#6728467 main -> 1cbe0298 valentine-ipp1	#1122441 created by 	build	convert_process_ge...	⌚ 00:00:59 📅 5 days ago	
	#6674417 main -> 7b737b37 valentine-ipp1	#1113204 created by 	build	convert_process_ge...	⌚ 00:04:53 📅 1 week ago	
	#6674240 main -> e49aaca5 valentine-ipp1	#1113144 created by 	build	convert_process_ge...	⌚ 00:00:57 📅 1 week ago	



Figure 4.48: Onglets Jobs de Gitlab CI


MIDDLEOFFICE > PLEIADE_SERVICE > test-automation > Jobs > #6735755

```


5468 93. AssertionError Check status code is 200
5469     iteration: 2     expected 404 to be one of [ 200, 204 ]
5470                        at assertion:0 in test-script
5471                        inside "converter / ClsToMid / {cls} / get Converter Mid"
5472
5473 94. AssertionError Check status code is 200
5474     iteration: 2     expected 404 to be one of [ 200, 204 ]
5475                        at assertion:0 in test-script
5476                        inside "converter / Cls2Midc / {cls} / get Converter Midc"
5477
5478 95. AssertionError Check status code is 200
5479     iteration: 2     expected 400 to be one of [ 200, 204 ]
5480                        at assertion:0 in test-script
5481                        inside "admin / users / get Users Global"
5482
5483 96. AssertionError Check status code is 200
5484     iteration: 2     expected 404 to be one of [ 200, 204 ]
5485                        at assertion:0 in test-script
5486                        inside "accounts / {accountId} / get Account Info"
5487 Newman run completed successfully.
5488 All collections run successfully.
✓ 5489 Uploading artifacts for successful job 00:03
5490 Uploading artifacts...
5491 ./newman: found 28 matching artifact files and directories
5492 Uploading artifacts as "archive" to coordinator... 201 Created id=6735755 responseStatus=201 Created token
=64_qrja7
✓ 5494 Cleaning up project directory and file based variables 00:00
5496 Job succeeded
5497 Job succeeded


```

convert_proces...  

Duration: 36 minutes 27 seconds
Finished: 4 days ago
Queued: 16 seconds
Timeout: 1h (from project) 
Runner: #1493 (sk7azjyFj) gittlab-runner-797f54bc6b-7l4fh
Tags: valentine-ipp1

Job artifacts
 These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available.

Commit d6acbdeb 
 change timeout time to 30s

✓ Pipeline #1123628 for main 

build 

→  convert_process_generate_...

Figure 49: Exemple d'écution d'un Job

CONCLUSION

En conclusion du Chapitre 4, nous avons présenté en détail les différentes étapes de notre Framework d'automatisation des tests d'API pour le projet Pléiade. De la configuration des collections à l'exécution des tests avec Newman, chaque étape contribue à garantir une gestion efficace et fiable des tests d'API. L'intégration de fonctionnalités telles que la génération de fichiers CSV, l'injection de valeurs de paramètres et l'ajout d'assertions renforce la robustesse et la qualité des tests, permettant ainsi de détecter et de résoudre rapidement les problèmes. Ce chapitre met en évidence l'importance de notre Framework dans l'amélioration continue des services de Pléiade en assurant des tests d'API complets, cohérents et précis.

Conclusion et Perspectives

Notre projet de fin d'études, mené au sein d'Intelcia IT Solutions, a abouti à la création d'un Framework d'automatisation des tests d'API REST pour le projet Pléiade. L'objectif principal de ce Framework est de faciliter et d'optimiser les tests des API REST, en automatisant les processus pour améliorer la qualité et l'efficacité des tests. Nous avons commencé par une étude approfondie des besoins spécifiques de Pléiade, mettant en évidence les limitations des méthodes de test manuelles existantes. Ces limitations étaient principalement liées à la répétitivité des tests et aux erreurs humaines.

Par la suite, notre responsabilité a été de concevoir et de réaliser le Framework en utilisant des technologies modernes et robustes. Nous avons choisi Node.js, Postman, GitLab CI, Scrum et Swagger pour développer notre solution. Grâce à ces choix technologiques, nous avons pu surmonter les problèmes techniques liés aux tests manuels, en réduisant les temps de test et en augmentant la couverture des tests. L'automatisation a permis de détecter plus rapidement les anomalies et de renforcer la réactivité de l'équipe aux évolutions des services.

À ce jour, bien que notre solution soit déjà opérationnelle et offre des améliorations significatives, elle est toujours en cours de développement et d'amélioration. Nous continuons à ajouter de nouvelles fonctionnalités et à optimiser les performances du Framework.

En plus de notre travail de développement du Framework d'automatisation, j'ai également été chargé de résoudre les anomalies détectées dans les API existantes. Cela comprenait l'identification, l'analyse et la résolution des problèmes rencontrés, afin de garantir le bon fonctionnement des services actuels. Parallèlement, j'ai participé aux activités de maintenance, veillant à ce que les mises à jour et corrections nécessaires soient effectuées régulièrement. Cela incluait la gestion des correctifs, la surveillance et la prise en compte des retours utilisateurs pour garantir une expérience optimale.

Grâce à mon implication dans la résolution des anomalies et la maintenance, nous avons pu améliorer la stabilité et la fiabilité des API de Pléiade, offrant ainsi une meilleure expérience aux utilisateurs finaux. La mise en place du Framework d'automatisation des tests constitue une avancée majeure pour l'équipe de développement et promet des gains de productivité importants à l'avenir. Toutefois, nous restons engagés à poursuivre l'amélioration continue de cette solution pour répondre pleinement aux besoins évolutifs du projet Pléiade.

Bibliographie

- Documentation Postman. Disponible sur : <https://learning.Postman.com/docs/>
- Documentation Node.js. Disponible sur : <https://nodejs.org/en/docs/>
- Documentation GitLab CI. Disponible sur : <https://docs.gitlab.com/ee/ci/>
- Stack Overflow. Disponible sur : <https://stackoverflow.com/>
- GeeksforGeeks. Disponible sur : <https://www.geeksforgeeks.org/>