

# PROJET DE FIN D'ETUDES

En vue de l'obtention du diplôme

**D'INGENIEUR D'ETAT**

**En Ingénierie Informatique et Technologies Emergentes (ZITE)**

## Élaboration d'un Framework de Génération et Automatisation des Tests d'API Rest

*Réalisé à :* Intelcia IT Solutions

***Réalisé par :***

Houssam Eddine Abou El Haoul

***Encadré par :***

---

Prof.

Mme. Khadija Bazoun

***Soutenu le /06/2024 devant le jury :***

Prof.

Prof.

---

Année Universitaire 2023/2024

---

# Dédicaces

# Remerciement

# Liste des Abréviations

# Résumé

# Abstract

# Table de Matières

Dédicaces.....	2
Remerciement.....	3
Liste des Abréviations.....	4
Résumé.....	5
Abstract.....	6
Table de Matières.....	7
Liste des figures.....	9
Liste des tableaux.....	10
1.1    Présentation de l'organisme d'accueil.....	12
1.2    Services d'Intelcia.....	15
1.3    Le pôle IT Solutions.....	15
1.3.1    Services d'Intelcia IT Solutions.....	16
1.3.2    Clients Intelcia IT Solutions.....	18
1.3.3    Structure organisationnelle.....	19
1.4    Présentation Générale du Projet.....	20
1.4.1    La Présentation du Client SFR.....	20
1.4.2    Présentation du projet Pléiade Service.....	20
1.5    Problématique du PFE.....	22
Chapitre 2:Etude préalable.....	23
2.1    Gestion de Projet.....	23
2.3.1    Choix de la méthode.....	23
2.3.2    La méthode SCRUM.....	23
2.3.3    Le processus SCRUM.....	24
2.3.4    Définition de période Sprint.....	24
2.2    Présentation Générale du principe des Tests.....	25
2.2.1    Définition du principe des Tests.....	25
2.2.2    Les Types de Tests.....	26
2.2.3    Comment tester les API Rest.....	27
2.3    Critique de l'existant.....	28

2.4	Comparaison des outils d'Automatisation de Test .....	28
2.5	Objectifs du Framework .....	30
Chapitre 3:Réalisation .....		31
3.1	Stack Technique .....	31
3.1.1	Node.js.....	31
3.1.2	Swagger .....	32
3.1.3	Postman / Newman.....	32
3.1.4	Gitlab CI.....	33
3.2	Conception du Framework .....	34
3.2.1	Planification et diagramme de GANTT .....	34
3.2.2	Architecture du Framework.....	36
3.2.3	Diagramme de Séquence .....	38
3.3	Approche de Développement .....	39
3.4	Fonctionnement du Framework.....	40
3.4.1	Workflow du Framework .....	40
3.4.2	Composants du Framework.....	42
3.4.3	Les Interactions entre les Composants .....	44
Chapitre 4:Présentation de la Solution .....		47
4.1	Présentation des Étapes de l'Automatisation.....	47
4.1.1	Téléchargement des Swagger.....	47
4.1.2	Conversion des Swagger en Collections .....	48
4.1.3	Configuration des Collections .....	50
4.1.4	Génération des fichiers CSV .....	51
4.1.5	Injection des Valeurs dans les Collections .....	51
4.1.6	Ajout des Assertions dans les Collections.....	53
4.1.7	Exécution des Collections .....	54
4.1.8	Génération des Rapports .....	55
4.2	Présentation de la Pipeline CI/CD.....	57
Conclusion et Perspectives .....		60
Bibliographie .....		60



# Liste des figures

Figure 1: Présence d'Intelcia à l'internationale .....	12
Figure 2: Service de Consulting .....	17
Figure 3: Cout relatif de correction d'une anomalie .....	26
Figure 4: Exemple d'outils d'Automatisation de Test .....	28
Figure 5: Benchmark des outils d'Automatisation des Tests .....	<b>Erreur ! Signet non défini.</b>
Figure 6: Node.js .....	31
Figure 7: Swagger .....	32
Figure 8: Postman .....	32
Figure 9: Newman .....	33
Figure 10: GitLab CI .....	33
Figure 11: Diagramme de GANTT .....	34
Figure 12: Test Automation Architecture .....	37
Figure 13: Diagramme de Séquence .....	39
Figure 14: Workflow du Framework .....	40
Figure 15: Processus Détaillé de comment la Collection est Traitée .....	42
Figure 16: Architecture du Framework .....	43
Figure 17: Le Module Data .....	44
Figure 18: Le Module swaggerToPostman .....	45
Figure 19: Le Module Newman .....	46
Figure 20: Execution de npm run download .....	47
Figure 21: Les Swagger Téléchargés .....	48
Figure 22: Execution de npm run convert .....	48
Figure 23: Conversion des Swagger en Collection .....	49
Figure 24: Les Collections Convertis .....	49
Figure 25: Execution de npm run process .....	50
Figure 26: Execution de npm run inject .....	51
Figure 27: Execution de npm run populate .....	51
Figure 28: Injection des données dans les Collections .....	52
Figure 29: Les Collections Configurées avec les Valeurs .....	52
Figure 30: Execution de npm run generate .....	53
Figure 31: Execution de npm run run .....	54
Figure 32: Le rapport local dans la CLI .....	54
Figure 33: Les rapports Générés dans le dossier Newman .....	55
Figure 34: L'interface Principale du Rapport .....	55
Figure 35: L'Onglet Total Requests .....	56
Figure 36: L'Onglet Failed Tests .....	57
Figure 37: Script de la Pipeline CI/CD .....	58
Figure 38: Onglets Jobs de Gitlab CI .....	59
Figure 39: Exemple d'écution d'un Job .....	59

# Liste des tableaux

Tableau 1: Tableau de Benchmark des Outils d'Automatisation des Tests..... 29

Tableau 2: Tableau de GANTT..... 35

# Introduction

# Chapitre 1: Contexte du projet

## 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: 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.

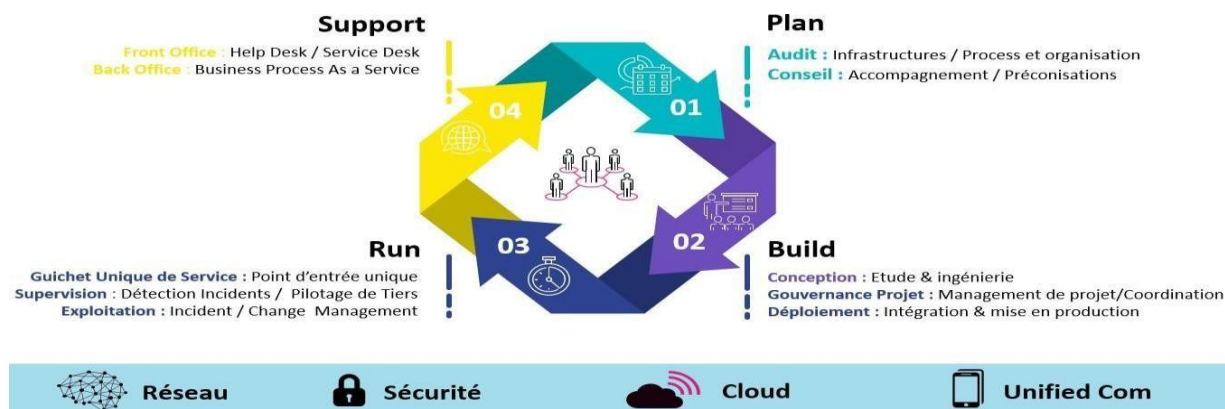


Figure 1.3: Service d'infogérance

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.

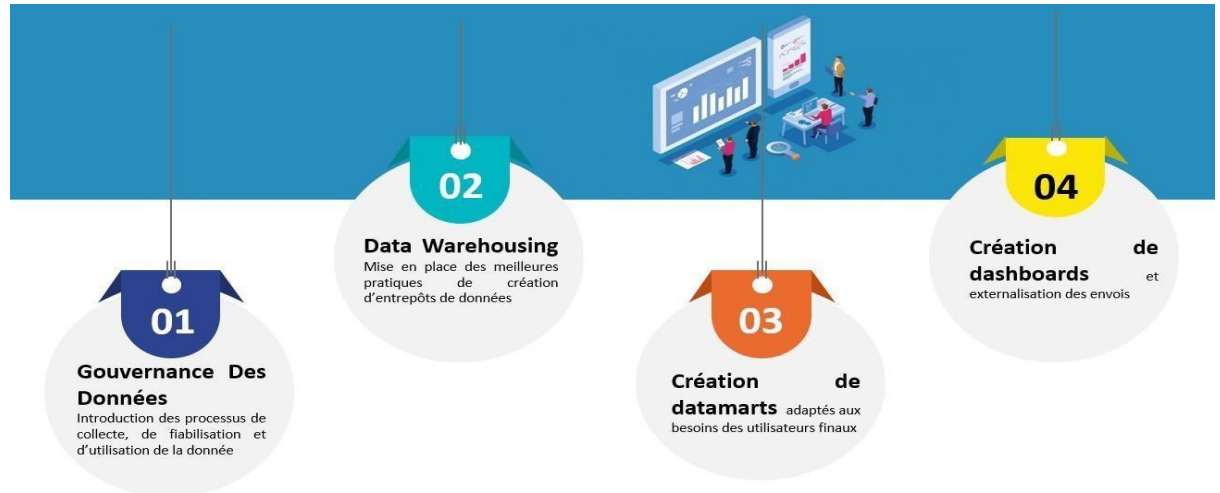


Figure 1.5: Services de Business Intelligence

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.

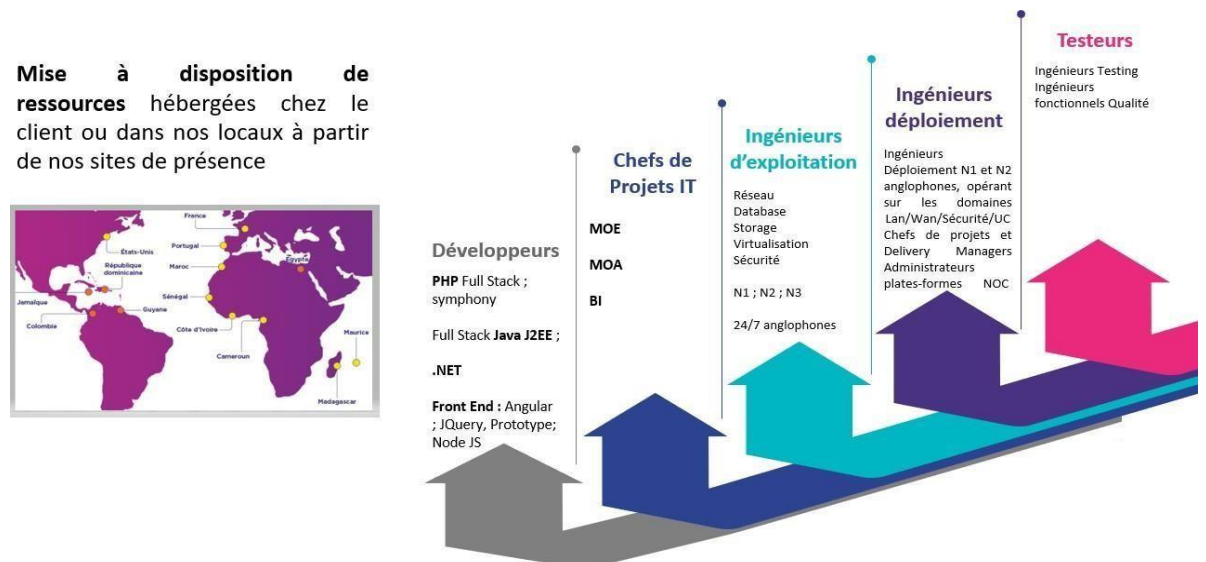


Figure 2: Service de Consulting

### 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.

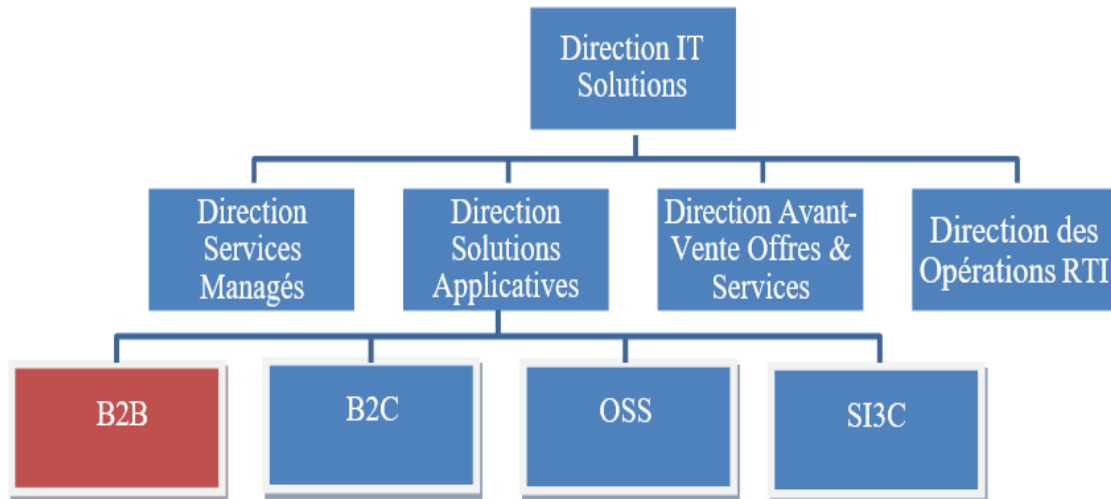


Figure 1.7: Clients d'Intelcia It Solutions



### 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 Solutions*

Mon 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.

## 1.4 Présentation Générale du Projet

### 1.4.1 La Présentation du Client SFR



*Figure 8 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 10 : Pléiade 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 exposées 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 fonctionnaient 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 mon stage PFE, j'ai 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'.

# Chapitre 2: Etude préalable

## 2.1 Gestion de Projet

Avant de passer à l'étude préalable, examinons d'abord comment le projet Pléiade est géré afin de mieux comprendre les enjeux. Cela nous permettra de cerner les défis spécifiques liés à la gestion et au test des nombreuses API REST, et de concevoir des solutions adaptées à ces besoins.

### 2.3.1 Choix de la méthode

Le choix entre une méthode et une autre, dépend de la nature du projet et de sa taille. Pour des projets de petite taille et dont le domaine est maîtrisé, par exemple, un cycle de vie en cascade s'avère largement suffisant. Lorsqu'il s'agit d'un projet où les données ne sont pas réunies dès le départ, où les besoins sont incomplets voire flous, il faut s'orienter vers une méthode itérative ou orientées prototypes.

Parmi les méthodes itératives, nous pouvons distinguer les méthodes AGILE largement utilisées de nos jours à travers le monde. Une méthode AGILE est menée dans un esprit collaboratif et s'adapte aux approches incrémentales. Elle engendre des produits de haute qualité tout en tenant compte de l'évolution des besoins du client. Une méthode AGILE assure une meilleure communication avec le client et une meilleure visibilité du produit livrable. Elle permet aussi de gérer la qualité en continu et de détecter des problèmes le plus tôt au fur et à mesure, permettant ainsi d'entreprendre des actions correctrices sans trop de pénalités dans les coûts et les délais. Il y a moult méthodes AGILE et il ne s'agit pas de choisir la meilleure méthode parmi celles existantes. Il s'agit plutôt de sélectionner la méthode la plus adaptée à notre projet.

La nature de projet qui doit être évolutif et dont tous les besoins n'ont pas encore été totalement identifiés, nous a orientées vers une méthode de type AGILE et plus particulièrement SCRUM.

### 2.3.2 La méthode SCRUM

La méthode SCRUM est un Framework ou cadre de travail de gestion de projets informatiques privilégiant la communication, et facilitant les réorientations opportunes. Ce cadre définit trois rôles principaux :

- **Le Product Owner** qui porte la vision du produit à réaliser (représentant généralement le client) ;
- **Le Scrum Master** garant de l'application de la méthodologie Scrum ;
- **L'équipe de développement** qui réalise le produit.

### 2.3.3 Le processus SCRUM

La vie d'un projet SCRUM est rythmée par un ensemble de réunions clairement définies et strictement limitées dans le temps (time boxing) :

- **Planification du Sprint (Sprint = itération)** : au cours de cette réunion, l'équipe de développement sélectionne les éléments prioritaires du « Product Backlog » (liste ordonnancée des exigences fonctionnelles et non fonctionnelles du projet) qu'elle pense pouvoir réaliser au cours du sprint (en accord avec le « Product Owner »)
- **Revue de Sprint** : au cours de cette réunion qui a lieu à la fin du sprint, l'équipe de développement présente les fonctionnalités terminées au cours du sprint et recueille les feedbacks du Product Owner et des utilisateurs finaux. C'est également le moment d'anticiper le périmètre des prochains sprints et d'ajuster au besoin la planification de release (nombre de sprints restants) ;
- **Rétrospective de Sprint** : la rétrospective qui a généralement lieu après la revue de sprint est l'occasion d'améliorer (productivité, qualité, efficacité, conditions de travail, etc.) à la lueur du "vécu" sur le sprint écoulé (principe d'amélioration continue)
- **Mêlée quotidienne** : il s'agit d'une réunion de synchronisation de l'équipe de développement qui se fait debout (elle est aussi appelée "stand up meeting") en 15 minutes maximum au cours de laquelle chacun répond principalement à 3 questions :  
« Qu'est-ce que j'ai terminé depuis la dernière mêlée ? Qu'est-ce que j'aurai terminé d'ici la prochaine mêlée ? Quels obstacles me retardent ? ».

### 2.3.4 Définition de période Sprint

Le sprint est une période d'un mois au maximum, au bout de laquelle l'équipe délivre un incrément du produit, potentiellement livrable. Une fois la durée choisie, elle reste constante pendant toute la durée du développement. Un nouveau Sprint démarre dès la fin du précédent.

Chaque Sprint possède un but et on lui associe une liste d'éléments du carnet du produit (fonctionnalité) à réaliser.

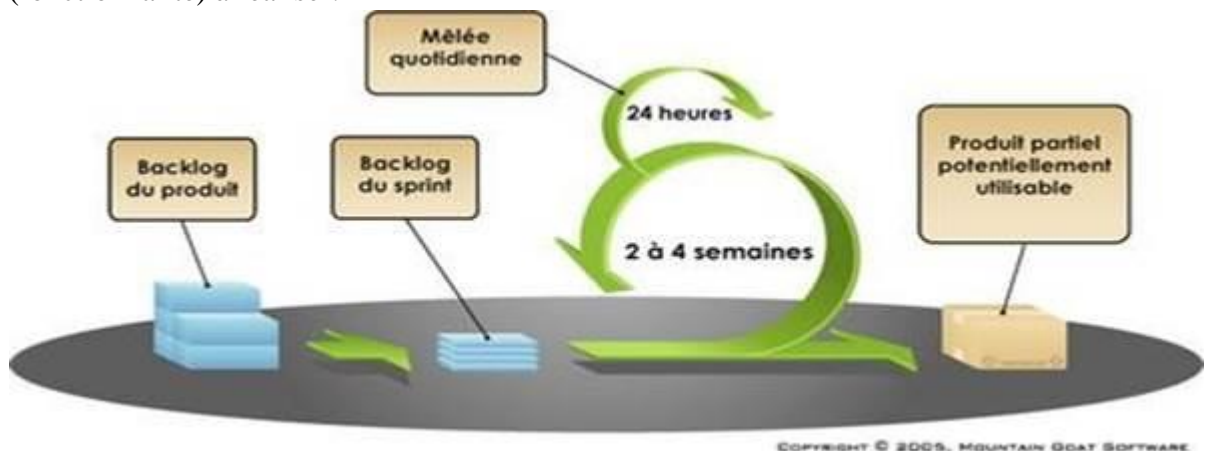


Figure 3.1: Principe de la méthode SCRUM

## 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.

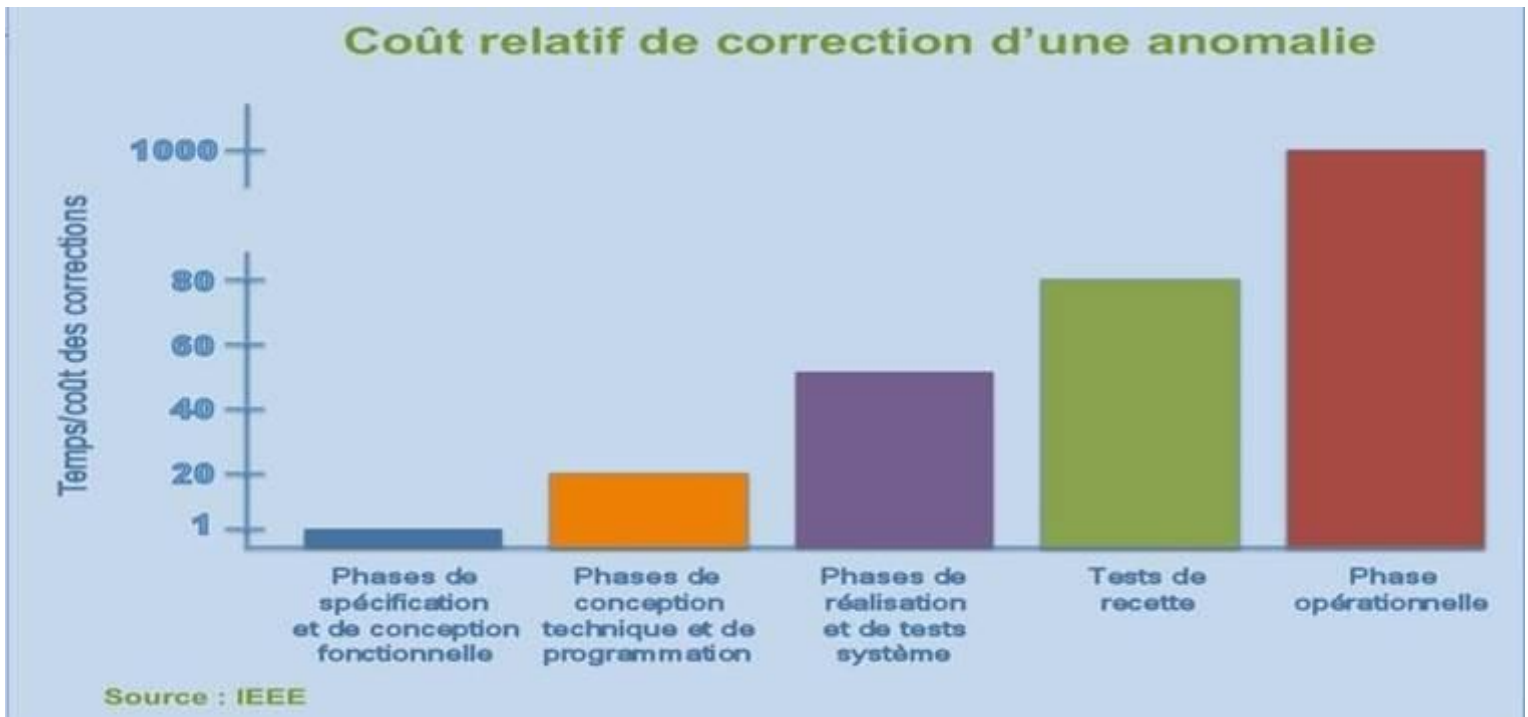


Figure 3: Cout relatif de correction d'une anomalie

### 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.

## 2.3 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 de mon stage PFE, j'ai eu la totale autonomie de choisir avec quel outil j'allais accomplir ma tâche d'automatiser les tests des API. Donc, j'ai dû faire des recherches sur les différents outils d'automatisation pour définir lequel serait le plus adaptés à nos besoins. Voici quelques exemple 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 4: Exemple d'outils d'Automatisation de Test

Et c'est en compilant toutes mes recherches sur ces différents outils, de leur documentation, de leur point forts et points faible que j'ai 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>
<b>Services</b>	<u>Rest</u> + SOAP	<u>Rest</u>	Web + Mobile + <u>Rest</u> + SOAP	<u>Rest</u>	<u>Rest</u> + SOAP	<u>Rest</u> + SOAP
<b>Spécialité</b>	Test de Performance	<u>Rest</u> en JAVA	Polyvalent ( mais plus interface graphique)	<u>Rest</u>	<u>Rest</u> + SOAP	<u>Rest</u> + SOAP
<b>Facilité d'utilisation</b>	Avancé	Avancé	Facile	Moyen	Facile	Moyen
<b>Intégration CI/CD</b>	CLI + Maven + plugins Jenkins	<u>Maven</u>	CLI	API d' <u>Integration</u>	CLI	CLI
<b>Tarification</b>	Open Source	Gratuit	Gratuit + Payant	Gratuit + Payant	<u>Gratuit</u> + <u>Payant</u>	Open Source + Payant

Tableau 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 j'ai dû élaborer mon propre Framework pour répondre spécifiquement aux exigences du projet Pléiade.

## 2.5 Objectifs du Framework

Le framework que j'ai 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.

# Chapitre 3: Réalisation

## 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 5: 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 6: 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 7: 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 8: 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 9: 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 :

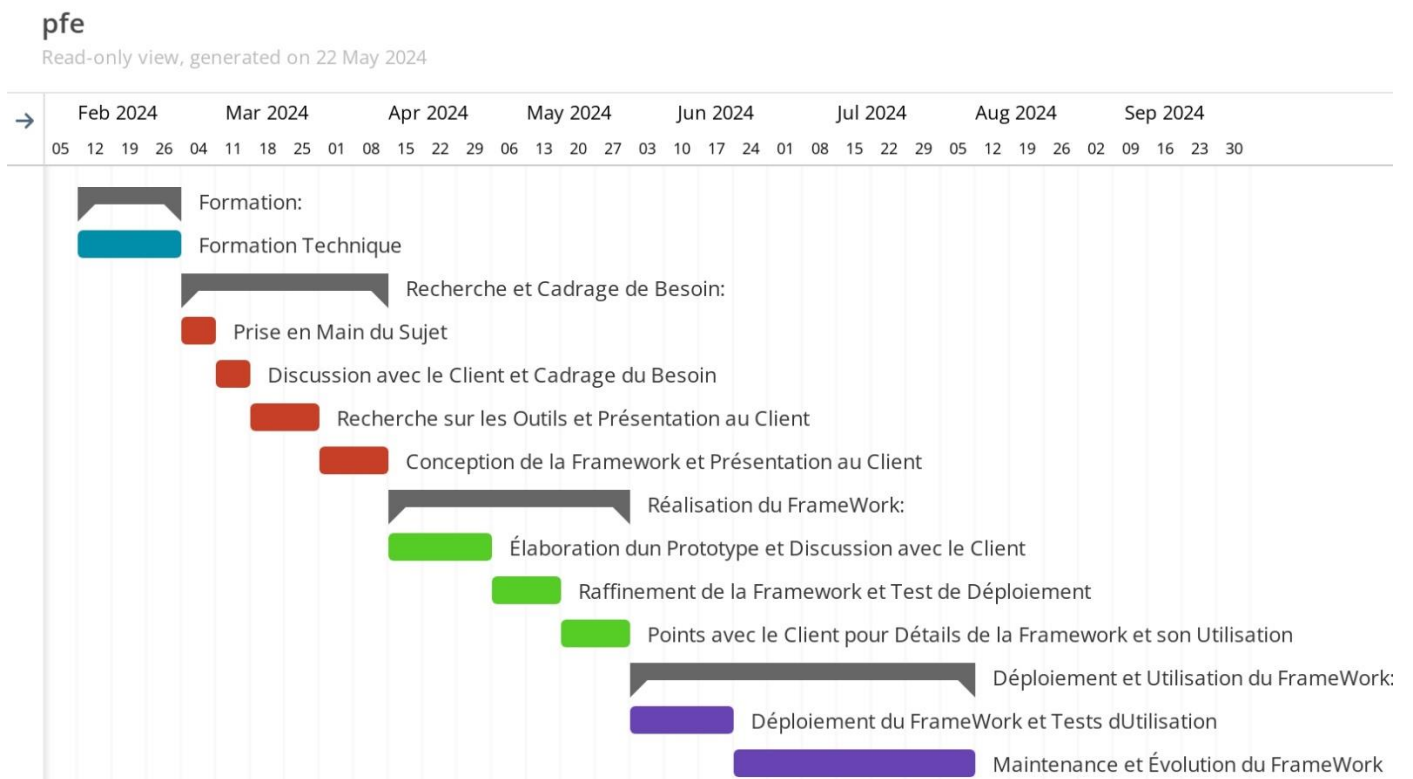


Figure 10: Diagramme de GANTT



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 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 Pleiade.

#### 3.2.2 Architecture du Framework

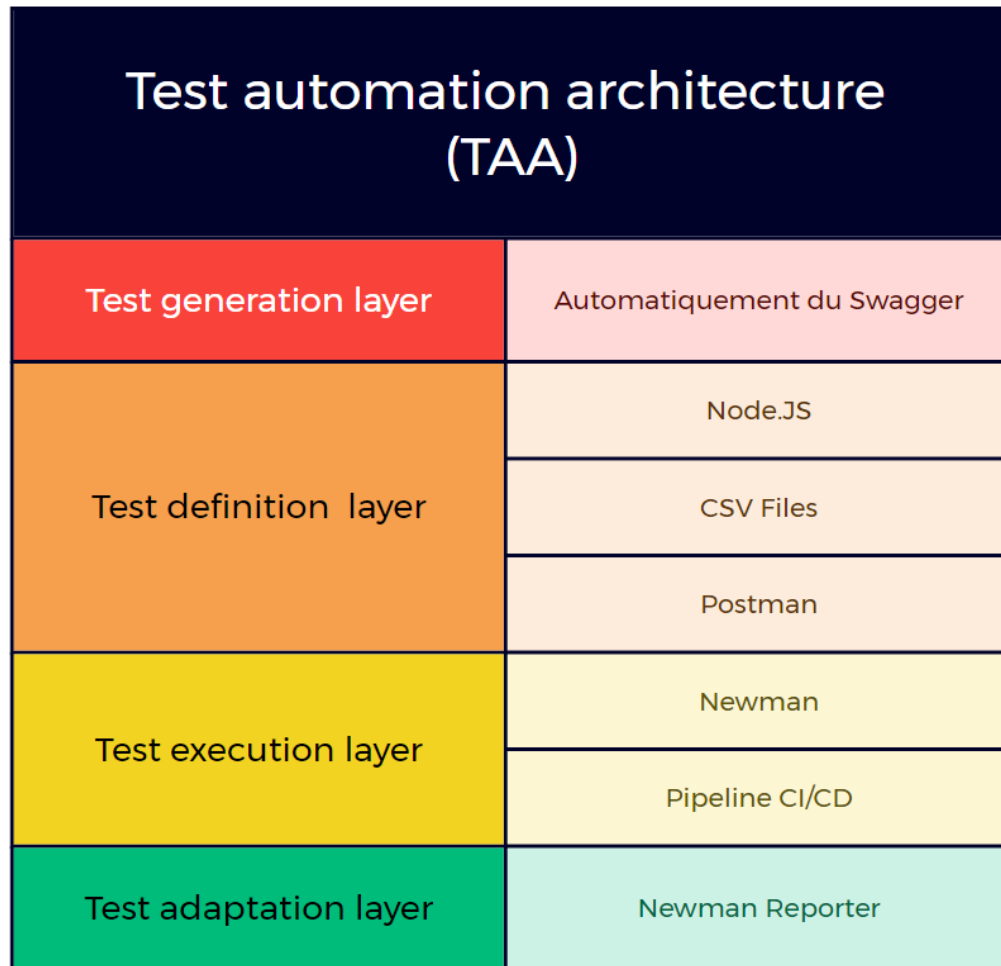


Figure 11: Test Automation Architecture

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

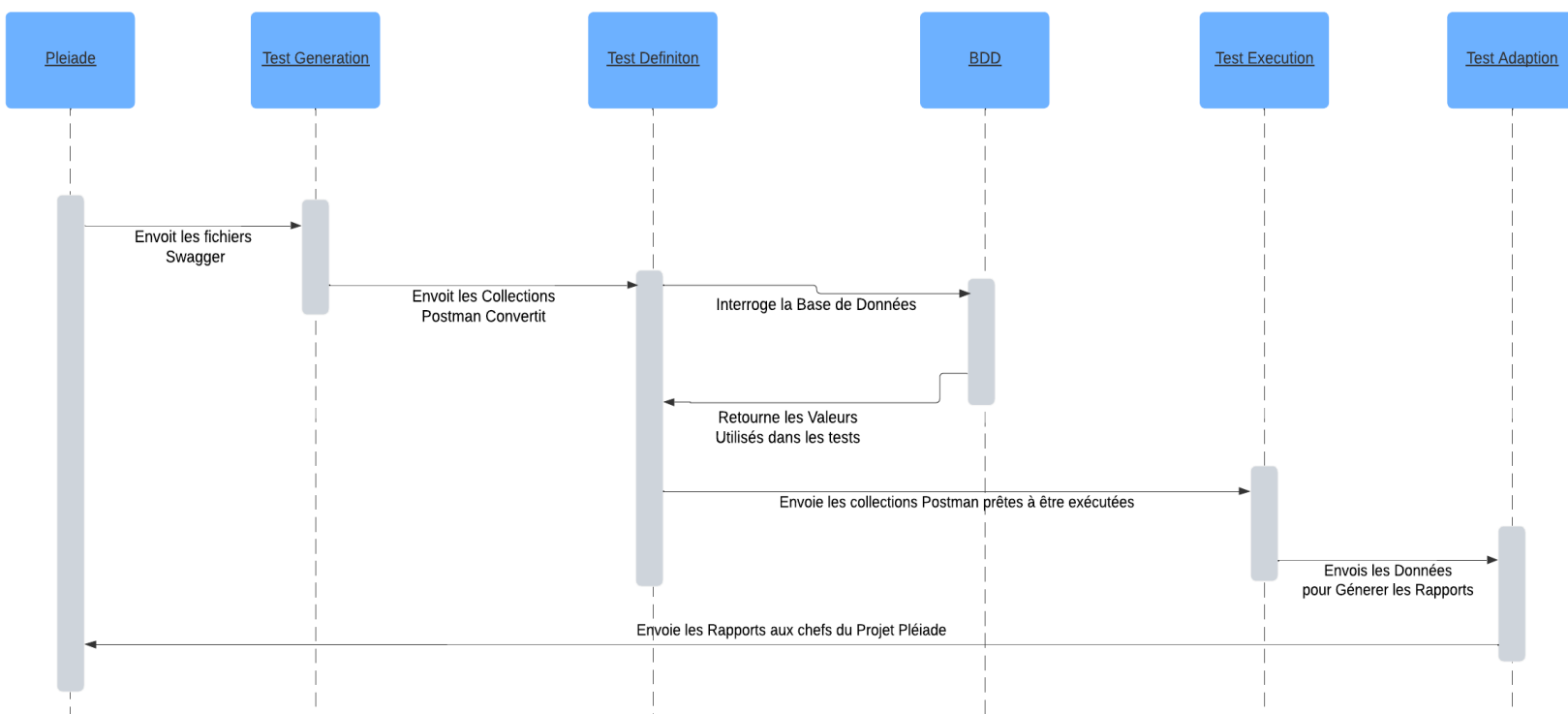


Figure 12: Diagramme de Séquence

### 3.3 Approche de Développement

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

- Je 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.
- Je maintiens 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 me permet d'ajuster les fonctionnalités du framework en fonction des retours et des nouvelles exigences du client.
- Je 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

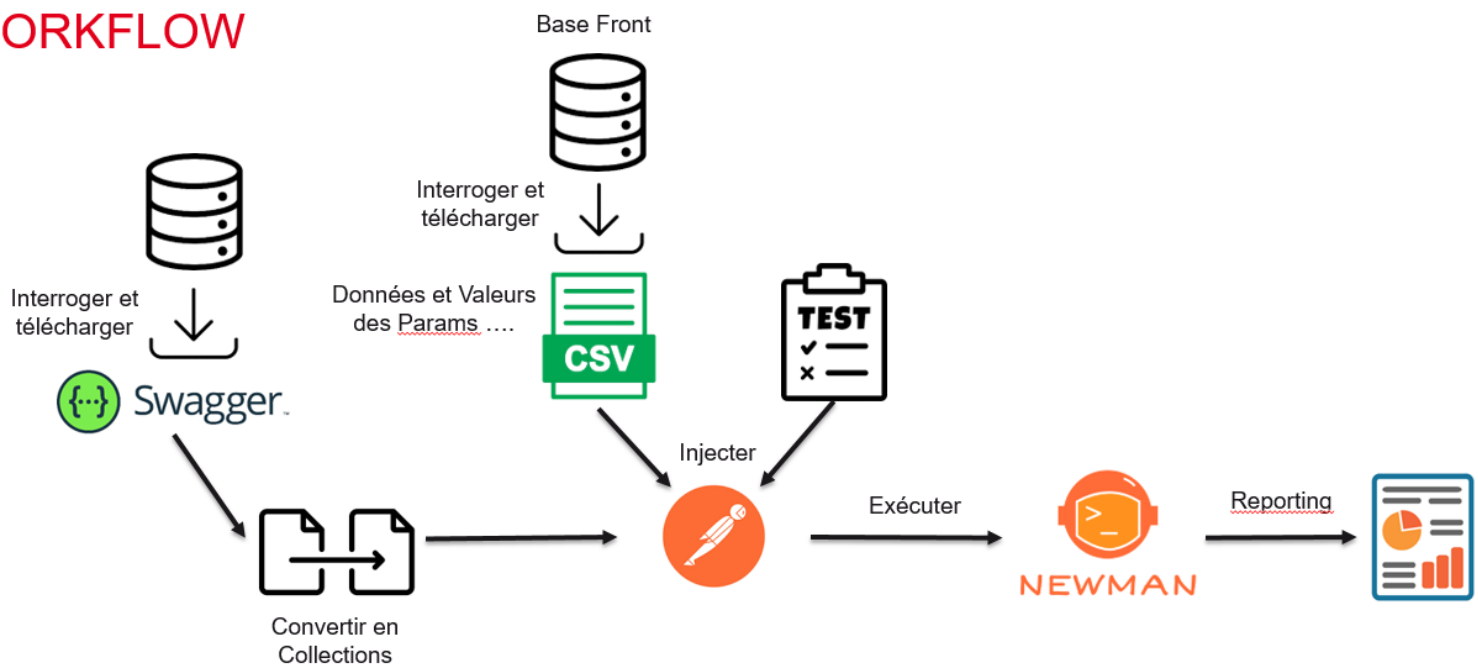


Figure 13: Workflow du Framework

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 Pleiade :

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

#### 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

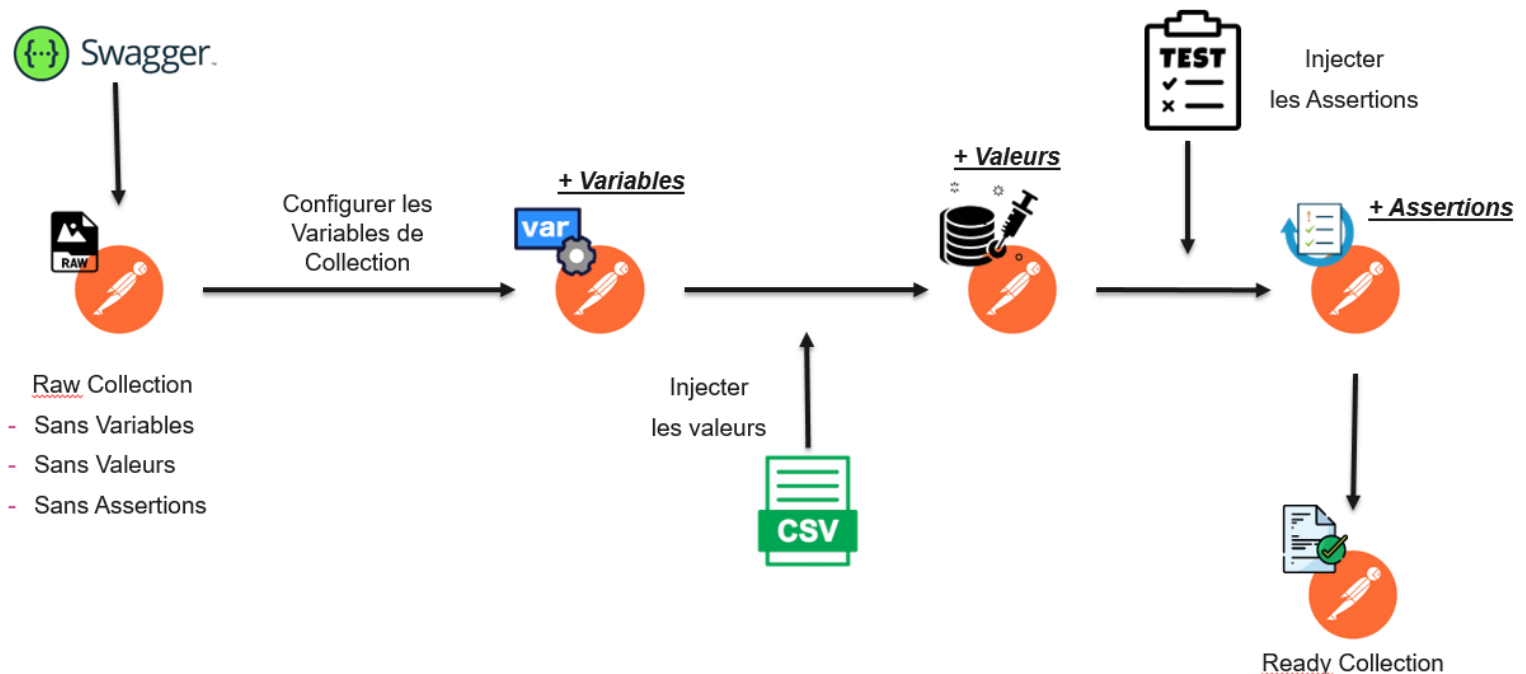


Figure 14: Processus Détaillé de comment la Collection est Traitée

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 utilisé 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 collections 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



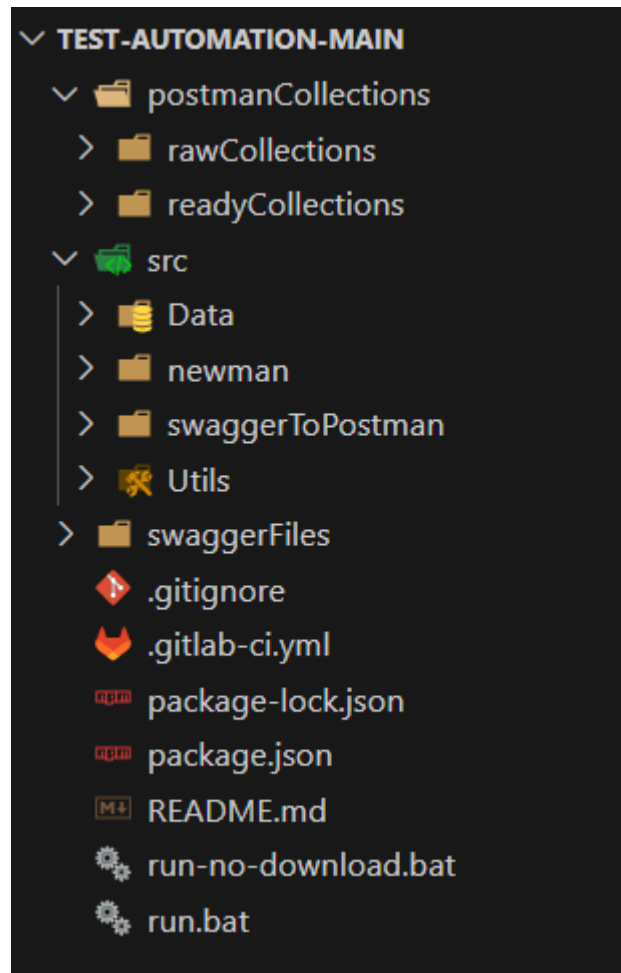


Figure 15: Architecture du Framework

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 :

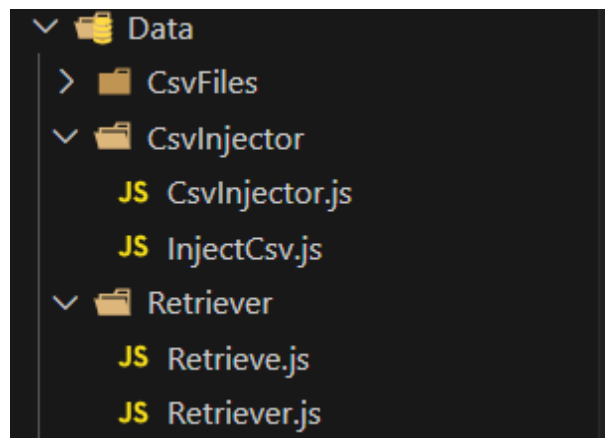


Figure 16: Le Module Data

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

- **Retriever** : 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 :

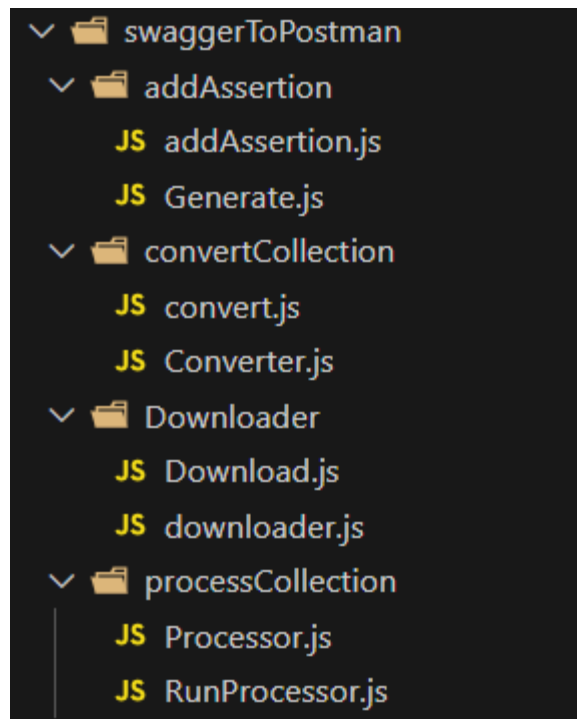


Figure 17: Le Module swaggerToPostman

Le module swaggerToPostman est celui qui passe tous les traitements nécessaires pour passer d'un fichier swagger à 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 18: 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 Pleiade, 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.

# Chapitre 4: Présentation de la Solution

## 4.1 Présentation des Étapes de l'Automatisation

Pour nos exemples et captures d'écrans dans cette partie, j'ai 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 stockées dans le dossier swaggerFiles :

```
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 19: Execution de npm run download

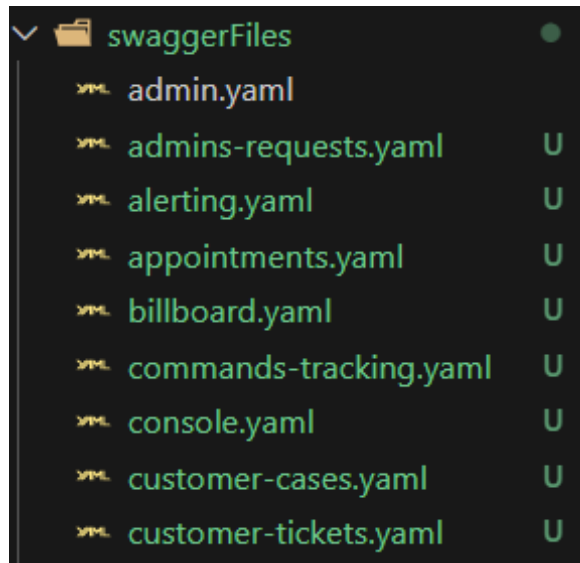


Figure 20: Les Swagger Téléchargés

### 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 :

```
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 21: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 22: Conversion des Swagger en Collection

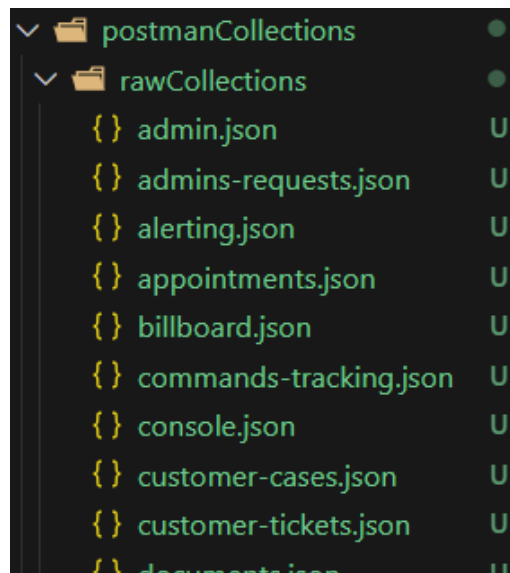


Figure 23: Les Collections Convertis

### 4.1.3 Configuration des Collections

Pour configurer les Collections et y ajouter les variables on utilise la commande **npm run process** :

```
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' => ''
}
```

Figure 24: 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 :

```
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\admin-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 25: 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 :

```
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 26: 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 27: Injection des données dans les Collections

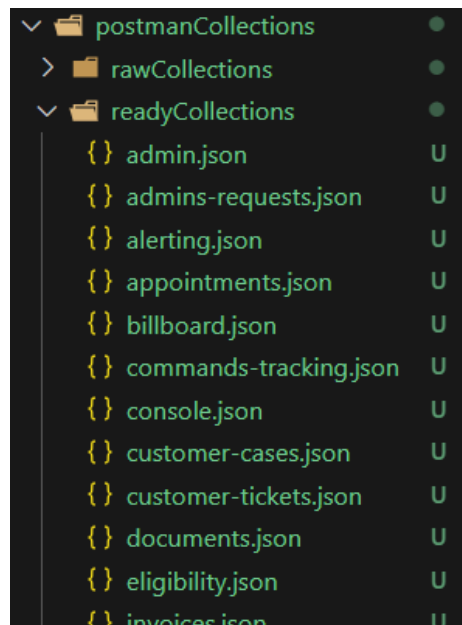


Figure 28: Les Collections Configurées avec les Valeurs

### 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 :

```
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 29: Execution de npm run generate

### 4.1.7 Exécution des Collections

Pour exécuter tous nos collections on utilise la commande **npm run run** :

```
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.ctn1.pic.services.pack/admin/v1.0/admin/termination-reasons/ [errored]
  getaddrinfo ENOTFOUND admin.pleiade-service.ctn1.pic.services.pack

└─ delete Termination Reason
  DELETE http://admin.pleiade-service.ctn1.pic.services.pack/admin/v1.0/admin/termination-reasons/ [errored]
  getaddrinfo ENOTFOUND admin.pleiade-service.ctn1.pic.services.pack

□ admin / termination-reasons
└─ find Termination Reasons
  GET http://admin.pleiade-service.ctn1.pic.services.pack/admin/v1.0/admin/termination-reasons?offerTypes=&configured= [errored]
  getaddrinfo ENOTFOUND admin.pleiade-service.ctn1.pic.services.pack

└─ create Termination Reason
  POST http://admin.pleiade-service.ctn1.pic.services.pack/admin/v1.0/admin/termination-reasons [errored]
  getaddrinfo ENOTFOUND admin.pleiade-service.ctn1.pic.services.pack

□ admin / offers / {offerId}
└─ get Offer
  GET http://admin.pleiade-service.ctn1.pic.services.pack/admin/v1.0/admin/offers/ [errored]
  getaddrinfo ENOTFOUND admin.pleiade-service.ctn1.pic.services.pack
```

Figure 30: 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.ctn1.pic.services.pack at request inside "admin / termination-reasons / {id} / update Termination Reason"
02.	Error iteration: 1	getaddrinfo ENOTFOUND admin.pleiade-service.ctn1.pic.services.pack at request inside "admin / termination-reasons / {id} / delete Termination Reason"
03.	Error iteration: 1	getaddrinfo ENOTFOUND admin.pleiade-service.ctn1.pic.services.pack at request inside "admin / termination-reasons / find Termination Reasons"
04.	Error iteration: 1	getaddrinfo ENOTFOUND admin.pleiade-service.ctn1.pic.services.pack at request inside "admin / termination-reasons / create Termination Reason"

Figure 31: Le rapport local dans la CLI

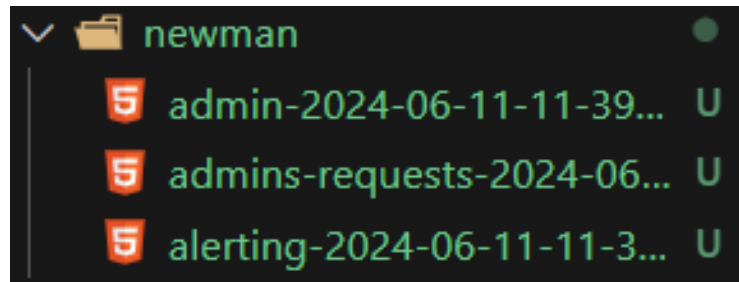


Figure 32: Les rapports Générés dans le dossier Newman

### 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.

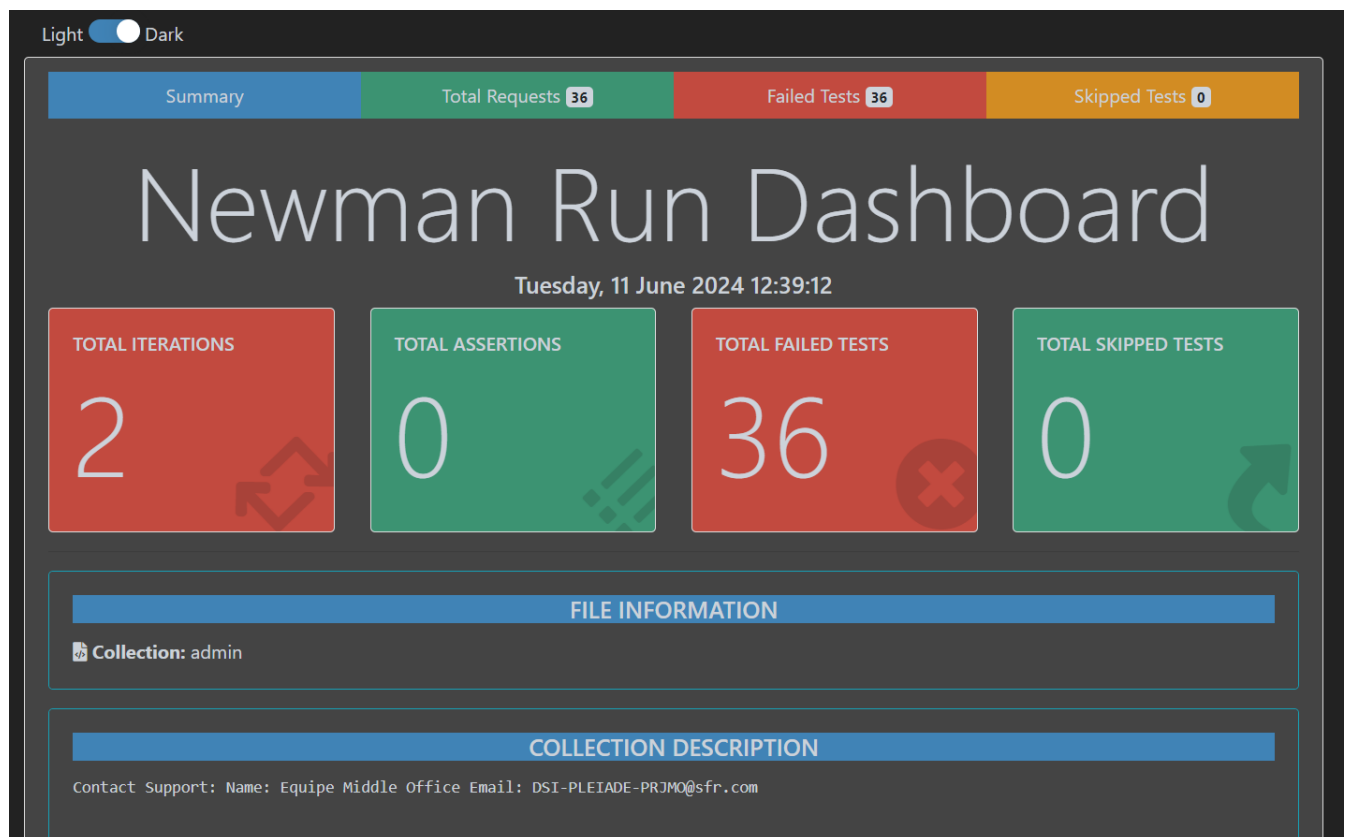


Figure 33: L'interface Principale du Rapport

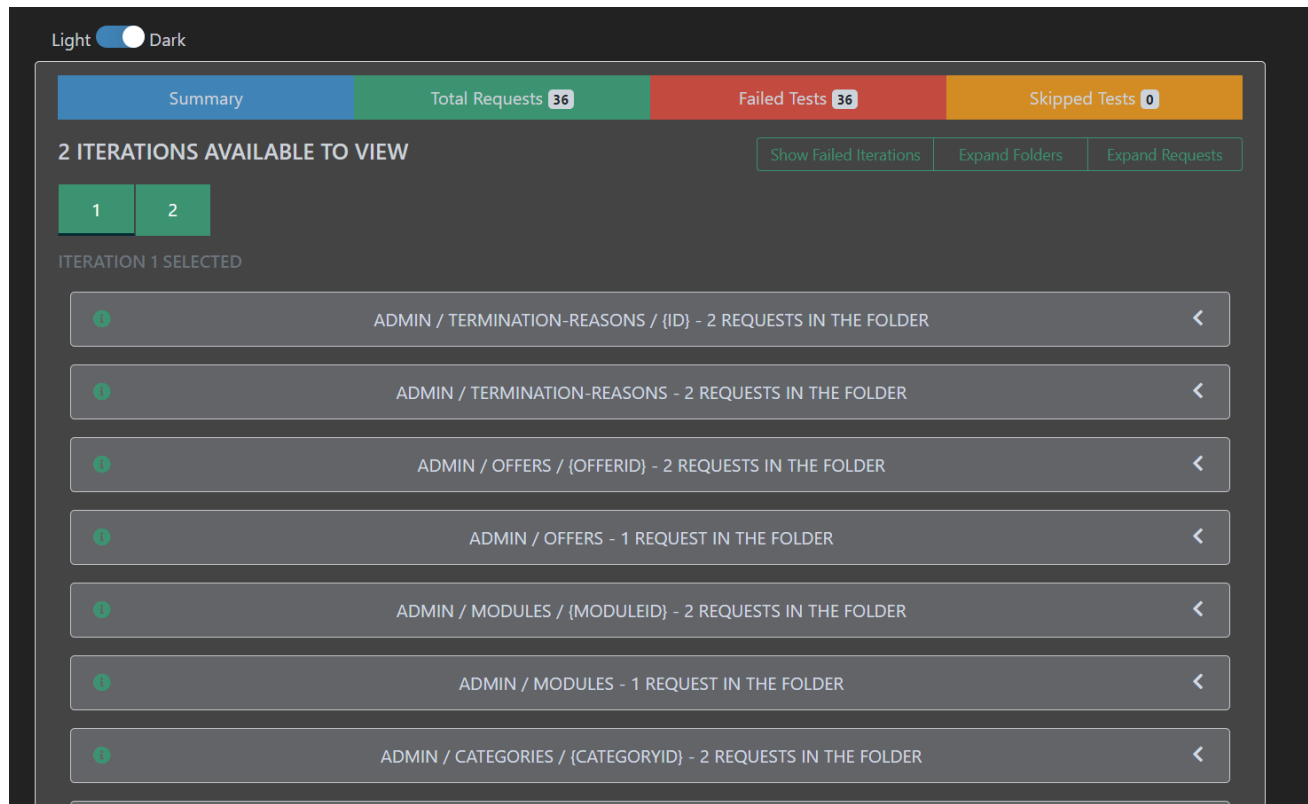


Figure 34: L'Onglet Total Requests

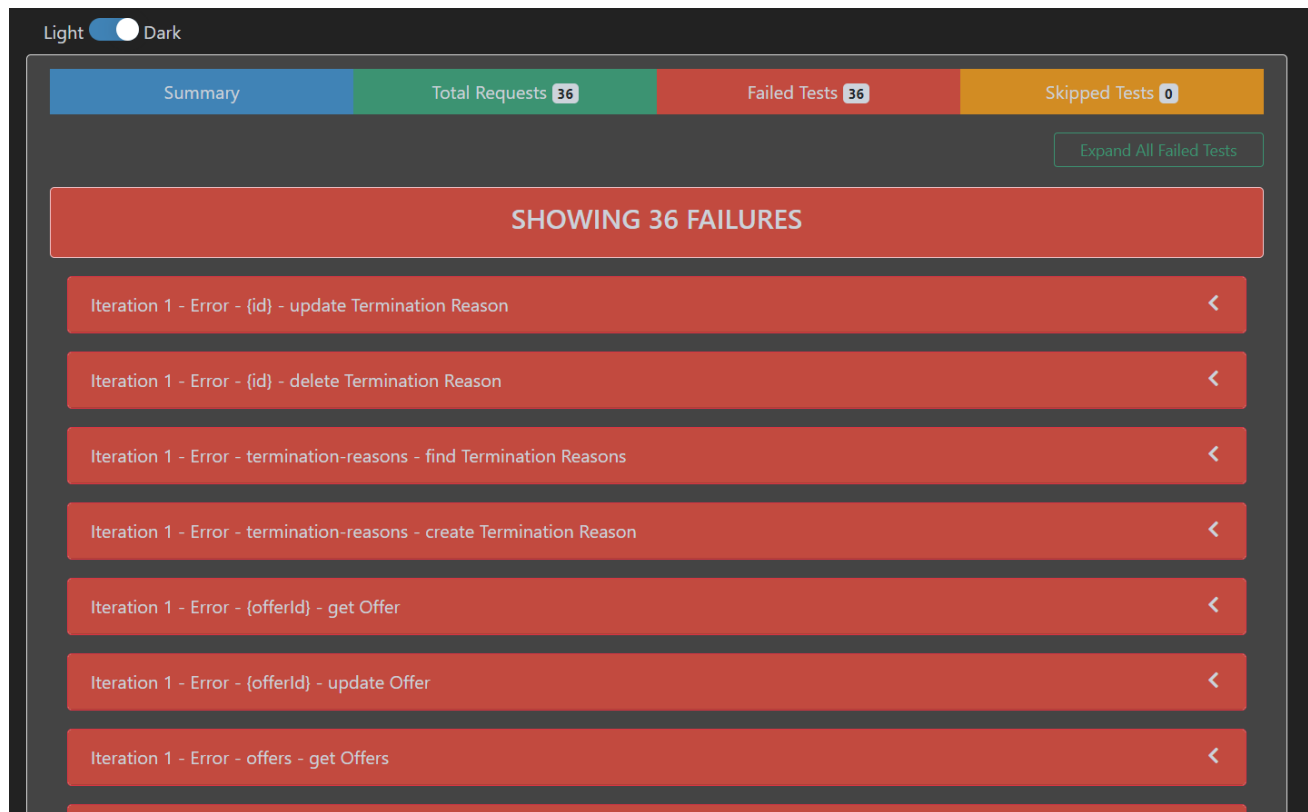


Figure 35: l'Onglet Failed Tests

## 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 efficiente. 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 36: 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 37: 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 / CIsToMid / {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 / CIs2Midc / {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
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
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) gitlab-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.


Keep

Download

Browse

Commit d6acbdeb 

change timeout time to 30s

 Pipeline #1123628 for main 

build

→  convert\_process\_generate....

Figure 38: Exemple d'exécution d'un Job

# Conclusion et Perspectives

## Bibliographie