

2024 - 2025

PROJET DE FIN D'ÉTUDES

DIPLÔME NATIONAL D'INGÉNIEUR

SPÉCIALITÉ : INFORMATIQUE

**Conception et développement du
plateforme Banking-as-a Service**

Réalisé par: Wael Gabsi

Encadré par:

Encadrant ESPRIT: Mme Chaima Messaad Ayari

Encadrant Entreprise: M Taieb Makrem



Je valide le dépôt du rapport PFE relatif à l'étudiant nommé ci-dessous / I validate the submission of the student's report:

- Nom & Prénom /Name & Surname :
- Wael Gabsi

Encadrant Entreprise/ Business site Supervisor

- Nom & Prénom /Name & Surname : Makrem Taieb

Cachet & Signature / Stamp & Signature



Encadrant Académique/Academic Supervisor

- Nom & Prénom /Name & Surname : Chayma Messad Ayari

Signature / Signature

Ce formulaire doit être rempli, signé et scanné/This form must be completed, signed and scanned.

Dédicace

Je dédie honorablement avec toutes mes gracieux remerciements à mon père qui m'a appris que le véritable succès réside dans la persévérance et l'intégrité , ma mère dont l'amour inconditionnel, le sacrifice et le soutien éclairent l'ensemble de toutes mes journées et à toutes les personnes qui m'ont motivé et aidé durant mon parcours .

Remerciements

Avant tout, je rends grâce à Dieu, qui m'a accordé la santé, la patience et la force nécessaires pour mener à bien ce projet.

Je tiens à exprimer ma profonde gratitude à mon encadrante académique, **Madame Chaima Messad Ayari**, pour sa disponibilité, sa rigueur, ses conseils précieux et son accompagnement constant qui ont été déterminants dans l'aboutissement de ce travail.

Mes remerciements s'adressent également à toute l'équipe de la **Société Tunisienne de Banque (STB)** pour son accueil chaleureux et sa collaboration. J'exprime particulièrement ma reconnaissance à mon encadrant professionnel, **Monsieur Makram Taieb**, pour son suivi attentif, son soutien constant ainsi que pour le partage de ses connaissances et ressources, qui ont grandement contribué à l'enrichissement de ce projet.

Sommaire

INTRODUCTION GÉNÉRALE.....	1
1. CHAPITRE 1 : CONTEXTE GÉNÉRALE ET PLANIFICATION DU PROJET	4
1.1. INTRODUCTION	4
1.2. PRÉSENTATION DE L'ORGANISME D'ACCUEIL.....	4
1.2.1. <i>Présentation de la banque</i>	4
1.2.2. <i>Son rôle dans l'économie tunisienne</i>	4
1.2.3. <i>Organigramme ou structure interne</i>	5
1.3. PRÉSENTATION DU PROJET	5
1.3.1. <i>Cadre Générale</i>	5
1.3.2. <i>Problématique</i>	5
1.3.3. <i>Etude et Critique de l'existant</i>	7
1.3.4. <i>Solution Proposée</i>	8
1.4. MÉTHODOLOGIE DE TRAVAIL	9
1.4.1. MÉTHODOLOGIE SCRUM 'AGILE'	9
1.4.2. ORGANISATION DES SPRINTS	10
1.4.3. DIAGRAMME DU GANT.....	10
1.5. CONCLUSION	11
2. CHAPITRE 2 : ÉTUDE THÉORIQUE ET CONCEPTION.....	13
2.1. INTRODUCTION	13
2.2. ENVIRONNEMENT DE DÉVELOPPEMENT ET CHOIX TECHNIQUES	13
2.2.1. <i>Environnement matériel</i>	13
2.2.2. <i>Environnement Logiciel</i>	14
2.2.2.1. Outils de développement	14
2.2.3. Outils de collaboration & DevOps.....	16
2.2.4. <i>Technologies et Frameworks</i>	17
2.2.4.1. Backend et Microservices.....	17
2.2.4.2. Frontend.....	19
2.2.4.3. Api Gateway et Sécurité.....	20
2.3. SPÉCIFICATIONS FONCTIONNELLES ET NON FONCTIONNELLES.....	22
2.3.1. <i>Spécifications fonctionnelles</i>	22
2.3.2. <i>Spécifications non fonctionnelles</i>	23
2.4. ARCHITECTURE DE L'APPLICATION.....	24
2.4.1. <i>Architecture logique</i>	24
2.4.1.1. Composants principaux	24
2.4.1.2. Flux logique	25
2.4.2. <i>Résumé graphique</i>	25
2.4.3. <i>Architecture physique</i>	26
2.5. CONCLUSION	27
3. CHAPITRE 3 : RÉALISATION DU PROJET	29
3.1. SPRINT 0 : ANALYSE ET SPÉCIFICATION DES BESOINS	29
3.1.1. <i>Introduction</i>	29
3.1.2. <i>Analyse des besoins</i>	29
3.1.2.1. Identification des acteurs	29
3.1.3. <i>Backlog Global</i>	29
3.1.4. <i>Conception globale</i>	34
3.2. SPRINT 1: MICROSERVICE COMPTES (ACCOUNT-SERVICE).....	35
3.2.1. <i>Introduction :</i>	35
3.2.2. <i>Backlog du sprint</i>	35

3.2.3.	Conception du sprint.....	37
3.2.4.	Revu du Sprint.....	39
3.2.5.	Démonstration.....	40
3.2.5.1.	Démonstration des apis.....	40
3.2.5.2.	Intégration Account service avec kong.....	40
3.2.6.	Burndown Chart	42
3.3.	SPRINT 2: MICROSERVICE CARTES (CARD-SERVICE)	43
3.3.1.	Introduction :	43
3.3.2.	Backlog du sprint.....	43
3.3.3.	Conception du sprint.....	44
3.3.4.	Revu du Sprint.....	47
3.3.5.	Démonstration.....	47
3.3.6.	Burndown Chart	49
3.4.	SPRINT 3 : MICROSERVICE GUICHETS AUTOMATIQUES (ATM-SERVICE)	50
3.4.1.	Introduction :	50
3.4.2.	Backlog du sprint.....	50
3.4.3.	Conception du sprint.....	51
3.4.4.	Revu du Sprint.....	53
3.4.5.	Démonstration.....	53
3.4.6.	Burndown Chart	54
3.5.	SPRINT 4: MICROSERVICE NOTIFICATIONS.....	55
3.5.1.	Introduction :	55
3.5.2.	Backlog du sprint.....	55
3.5.3.	Revu du Sprint.....	55
3.5.4.	Démonstration.....	56
3.5.5.	Burndown Chart	58
3.6.	SPRINT 5: COMMUNICATION INTERSERVICES (RABBITMQ + GRPC).....	59
3.6.1.	Introduction :	59
3.6.2.	Backlog du sprint.....	59
3.6.3.	Conception du sprint.....	60
3.6.3.1.	Communication asynchrone avec RabbitMQ	60
3.6.3.1.1.	Production d'événements :	60
3.6.3.1.2.	consommation des événements.....	60
3.6.3.1.3.	Mécanisme de routage :	60
3.6.3.2.	Communication synchrone avec gRPC.....	61
3.6.3.2.1.	Cas d'utilisation :	61
3.6.3.2.2.	Définition des contrats :	61
3.6.3.2.3.	avantages.....	61
3.6.4.	Revu du Sprint.....	62
3.6.5.	Démonstration.....	62
3.6.5.1.	Démonstration pour rabbitmq.....	62
3.6.6.	Burndown Chart	64
3.7.	SPRINT 6 : API GATEWAY KONG (SÉCURITÉ, RATE LIMITING, AUTHENTICATION)	65
3.7.1.	Introduction :	65
3.7.2.	Backlog du sprint.....	65
3.7.3.	Conception du sprint.....	66
3.7.4.	Revu du Sprint.....	66
3.7.5.	Démonstration.....	67
3.7.6.	Burndown Chart	70
3.8.	SPRINT 7: MICROSERVICE AUTHENTICATION (JWT, OAUTH, SSL)	71
3.8.1.	Introduction :	71
3.8.2.	Backlog du sprint.....	71
3.8.3.	Conception du sprint.....	72

3.8.4.	<i>Revu du Sprint</i>	74
3.8.5.	<i>Démonstration</i>	74
3.8.5.1.	Démonstration Api.....	74
3.8.5.2.	Intégration Authentication service avec konga	75
3.8.5.3.	Démonstration en vue client.....	76
3.8.5.4.	Consommation documentation.....	77
3.8.5.5.	Intégration helper dans des autres projets	78
3.8.6.	<i>Burndown Chart</i>	79
3.9.	SPRINT 8 : DÉPLOIEMENT & MONITORING (DOCKER, OVH, UFW,SSL)	80
3.9.1.	<i>Introduction :</i>	80
3.9.2.	<i>Backlog du sprint</i>	80
3.9.3.	<i>Revu du Sprint</i>	81
3.9.4.	<i>Démonstration</i>	82
3.9.4.1.1.	Choix du VPS(Serveur virtuel Privé).....	82
3.9.4.1.2.	Paieement et mise en service.....	83
3.9.4.1.3.	Connexion et configuration initiale du VPS	84
3.9.4.1.4.	Sécurisation du VPS.....	85
	Services accessibles depuis l'extérieur (clients, dashboards, monitoring).....	87
3.9.4.1.5.	DNS configuration	88
3.9.4.1.6.	Configuration certificat SSL.....	89
3.9.4.1.7.	Intégration CloudFlare.....	90
3.9.4.1.8.	Déploiement des conteneurs Docker	91
3.9.5.	<i>Visualisation du Système</i>	93
3.9.6.	<i>Burndown Chart</i>	95
4.	CHAPITRE 4 : CONCLUSION & PERSPECTIVE	97
	<i>Perspectives</i>	97

Liste des figures

Figure 1 : Logo STB (https://www.stb.com).....	4
Figure 2: Organigramme de la STB (https://www.stb.com)	5
Figure 3: Méthode Scrum (https://www.tuleap.com).....	9
Figure 4: Diagramme du Gant.....	10
Figure 5: Logo Visual Studio Code.....	14
Figure 6: Logo Visual Studio	14
Figure 7: Logo Insomnia	15
Figure 8: Logo Swagger	15
Figure 9: Logo GitHub	15
Figure 10: Logo Docker Desktop	16
Figure 11: Logo Rabbitmq Management Console	16
Figure 12: Logo MobaXterm	16
Figure 13: Logo Kube Ctl	17
Figure 14: Logo Prometheus et Grafana	17
Figure 15: Logo Elasticsearch et Kibana	17
Figure 16: Logo .NET	18
Figure 17: Logo gRPC	18
Figure 18: Logo RabbitMQ.....	18
Figure 19: Logo PostgreSQL	19
Figure 20: Logo MySQL.....	19
Figure 21: Logo MongoDB.....	19
Figure 22: Logo Angular.....	20
Figure 23: Logo NGINX	20
Figure 24: Logo KONG Api Gateway	20
Figure 25: Logo KONGA	20
Figure 26: Logo CLOUDFARE	21
Figure 27: Logo Let's Encrypt SSL	21
Figure 28: Logo JWT	21
Figure 29: Logo UFW	22
Figure 30: Logo fail2ban.....	22
Figure 31: Logo Namecheap	22
Figure 32: Architecture logique	25
Figure 33: Architecture physique	27
Figure 34: Diagramme de classe (Account Microservice).....	37
Figure 35: Diagramme de séquence (Account Microservice).....	38
Figure 36: Diagramme de cas d'utilisation(Account Microservice).....	39
Figure 37: Documentation Api.....	40
Figure 38: Intégration account service avec kong.....	40
Figure 39: Intégration account service avec kong.....	41
Figure 40: Exemple des fintech (consumers) qui consomment le service	41
Figure 41: Burndown chart, Sprint 1	42

Figure 42: Diagramme de cas d'utilisation (Card Service)	44
Figure 43: Diagramme de classe(Card Service).....	45
Figure 44: Diagramme de séquences(Card Service)	46
Figure 45: Documentation Api.....	47
Figure 46: Intégration de Card-service avec Kong Gateway	48
Figure 47: Route Card service proxisée avec kong	48
Figure 48: Burndown chart, Sprint 2.....	49
Figure 49: Diagramme de classe (ATM Service).....	51
Figure 50: Diagramme de cas d'utilisation.....	51
Figure 51: Diagramme de séquences (ATM Service)	52
Figure 52: Documentation Api.....	53
Figure 53: Intégration Atm-service avec Kong gateway	54
Figure 54: Route Atm-service proxisé avec kong	54
Figure 55: Burndown chart, Sprint 3.....	54
Figure 56: Interface Rabbitmq utilisé pour la communication asynchrone	56
Figure 57: Notifications consommées en conditions réelles.	57
Figure 58: Visualisation du queues	57
Figure 59: Intégration notification-service avec kong	58
Figure 60: Route du service notification-service	58
Figure 61: BurnDown Chart Sprint 4	58
Figure 62:tableau de queues.	61
Figure 63: Démonstration de l'interface RabbitMQ.....	62
Figure 64 : Événement capturé.....	63
Figure 65: Consommation des événements par le service de notification	63
Figure 66: Burndown chart, Sprint 5.....	64
Figure 67: Dashboard konga	67
Figure 68 : intégration des microservices avec kong	67
Figure 69: Route de chaque microservice	68
Figure 70: Liste des consommateurs adéquats	68
Figure 71: les plugins utilisés.....	68
Figure 72: connection Kong avec Konga	69
Figure 73: Consommation de l'api ATM a partir de kong	69
Figure 74: Burndown chart, Sprint 6.....	70
Figure 75 :Diagramme de classe (Authenticator Service)	72
Figure 76:Diagramme de cas d'utilisation (Authenticator Service)	72
Figure 77: Diagramme de séquence (Authenticator Service)	73
Figure 78: Documentation Api.....	74
Figure 79: Intégration Authentication service avec kong	75
Figure 80 : Déclaration de la route à accéder ce service a partir de kong.....	75
Figure 81: Interface Création Compte pour les Fintechs	76
Figure 82:Interface login pour les Fintechs.....	76
Figure 83: Dashboard Gestion Clé pour les Fintechs.....	76
Figure 84: Offres des services	77
Figure 85: Interface clé	77

Figure 86: Téléchargement de documentation	77
Figure 87: consommation du api_helper dans des autres projets	78
Figure 88: application Fintech pour Visualisations des distributeurs actifs avec localisation	78
Figure 89: Burndown Chart, Sprint 7	79
Figure 90: gamme des Vps d' OVH	82
Figure 91: choix du VPS	83
Figure 92 : Achat du VPS	83
Figure 93: Connexion via le portail mobaxterm	84
Figure 94 : Mise à jour Du Système.....	84
Figure 95 : Vérification Du Docker	85
Figure 96 : Modification du port SSH.....	85
Figure 97: redémarrage du service ssh	85
Figure 98:Configuration fail2ban.....	86
Figure 99: statut fail2ban.....	86
Figure 100: statut du jail.....	86
Figure 101:liste des ports a accédé.....	87
Figure 102:Configuration des records DNS	88
Figure 103 : Résolution DNS (nslookup).....	88
Figure 104 : Test de connectivité (ping)	88
Figure 105: Installation de Certbot et du plugin Nginx	89
Figure 106 : Génération du certificat pour le domaine	89
Figure 107 : Test du SSL.....	89
Figure 108 : https://realstataert.it.com	90
Figure 109 : : Configuration des enregistrements DNS	90
Figure 110 : : Application est sécurisé	91
Figure 111 : Conteneurs Docker	91
Figure 112 : Burndown Chart Sprint 8.....	95

Liste des tables

Tableau 1 : Etude et Critique de l'existant.....	7
Tableau 2: Backlog Global.....	33
Tableau 3: Backlog Sprint 1.....	36
Tableau 4: Backlog Sprint 2.....	44
Tableau 5: Backlog Sprint 3.....	50
Tableau 6: Backlog Sprint 4.....	55
Tableau 7: Backlog Sprint 5.....	60
Tableau 8: Backlog Sprint 6.....	65
Tableau 9: Backlog Sprint 7.....	71
Tableau 10: Backlog Sprint 8.....	81
Tableau 11: Détails du Conteneurs	92

Introduction générale

La transformation numérique a profondément bouleversé le secteur bancaire à l'échelle internationale. Elle est devenue un levier essentiel pour répondre aux attentes d'une clientèle de plus en plus exigeante en matière d'accessibilité, de rapidité, de sécurité et d'interconnexion avec d'autres solutions financières.

Dans ce contexte, les banques et institutions financières sont contraintes de moderniser leurs systèmes d'information traditionnels afin de rester compétitives et de se conformer aux nouvelles exigences de sécurité, de transparence et de réglementation.

En **Tunisie**, cette évolution représente un véritable défi pour les acteurs du secteur. Les banques doivent mettre en place des infrastructures technologiques de pointe afin d'améliorer le traitement des opérations, renforcer la confiance de leurs clients et accroître leur attractivité. La **Société Tunisienne de Banque (STB)**, acteur historique du paysage financier national, se trouve ainsi à un tournant stratégique où l'innovation technologique constitue un levier incontournable de compétitivité et de fidélisation.

La digitalisation bancaire ne se limite plus à la mise en ligne de services classiques tels que la consultation des comptes ou la gestion des cartes. Elle implique une refonte globale des systèmes existants en vue de créer des plateformes unifiées, capables de regrouper les services, d'automatiser les processus et de faciliter la communication entre les différents acteurs financiers.

C'est dans cette dynamique que s'inscrit le projet présenté dans ce rapport. Son objectif est de concevoir et de déployer une **plateforme bancaire intégrée**, fondée sur une **architecture en microservices**, offrant une gestion centralisée des comptes, des cartes bancaires, des guichets automatiques et des alertes. La solution expose également des **API conformes aux standards Open Banking**, permettant aux fintechs et services tiers d'accéder, de manière sécurisée et standardisée, aux données et services bancaires. Cette ouverture favorise à la fois la transparence et la compétitivité de la banque dans un contexte international.

Sur le plan technique, la solution s'appuie sur des technologies modernes et robustes : **.NET 8** côté serveur, une combinaison de **bases de données relationnelles et NoSQL** selon les besoins spécifiques des microservices, **RabbitMQ** pour la communication asynchrone et **gRPC** pour les échanges synchrones. Le côté client est développé avec **Angular 16**. Le tout est déployé dans un environnement **conteneurisé avec Docker**, orchestré via **Kubernetes**, et supervisé à l'aide d'outils de monitoring tels que **Elasticsearch, Kibana, Prometheus et Grafana**.

Le présent rapport décrit l'ensemble des étapes ayant conduit à la réalisation de ce projet. Le **premier chapitre** présente le contexte général, la banque d'accueil et la planification du projet. Le **deuxième chapitre** est consacré à l'étude théorique et à la conception du système, incluant les choix technologiques et l'architecture envisagée. Le **troisième chapitre** détaille la réalisation du projet, le développement des microservices, la mise en place du frontend et la communication interservices. Le **quatrième chapitre** traite des aspects DevOps, de la sécurité et du monitoring. Enfin, le **cinquième chapitre** présente la conclusion générale, les apports du projet et les perspectives d'évolution.

Chapitre 1 : Contexte générale et planification du projet

1. Chapitre 1 : Contexte générale et planification du projet

1.1. Introduction

Ce chapitre présente le contexte général du projet , la société d'accueil , ainsi que la planification des différentes étapes de sa réalisation . Il met les points lignes aux niveaux des besoins auxquels il répond , définit la méthodologie adoptée et situe le projet dans son environnement .

1.2. Présentation de l'organisme d'accueil

1.2.1. Présentation de la banque

La Société tunisienne de banque ou STB est une banque tunisienne fondée le 18 janvier 1957. Elle ouvre sa première succursale le 26 mars 1958.[1].

Il s'agit d'une société anonyme régie par la loi n°2001-65 relative aux établissements de crédits et dont le capital s'élève à 776,875 millions de dinars tunisiens, dont l'État tunisien détient 81,83 % d'une façon directe et indirecte.[1].

Créée avec un statut de banque de dépôt et d'investissement, la Société Tunisienne de Banque a représenté, durant des décennies, un instrument privilégié de la politique financière du pays. Avec l'avènement des années 80, le groupe STB se présentait déjà comme une force économique de première importance, constitué de dizaines de filiales appartenant à tous les secteurs de l'économie et considéré comme l'un des plus importants employeurs du pays. [2].



Figure 1 : Logo STB (<https://www.stb.com>)

1.2.2. Son rôle dans l'économie tunisienne

La STB a aussi joué un rôle crucial dans la recherche des sources de financement les plus adaptées aux besoins des investisseurs et la gestion rigoureuse des fonds spéciaux d'origine étrangère ou publique. La transformation de l'environnement économique et financier a nécessité une réforme de la politique monétaire et de crédit mise en œuvre au cours de la décennie 1987-1996 visant particulièrement le renforcement des assises financières des banques par l'introduction des normes prudentielles et la libéralisation des taux d'intérêt, le développement de la concurrence interbancaire et la dynamisation du marché financier. [2].

- ❖ La structure rigide des infrastructures actuelles, fréquemment monolithiques, entrave la possibilité de transition rapide vers des modèles plus souples et orientés vers le client.
- ❖ Des défis liés à l'échelle et au déploiement rapide entravent une réaction efficace face à l'augmentation des demandes des utilisateurs ou à la mise en place de nouveaux services.
- ❖ **Expérience utilisateur souvent limitée**, marquée par une lenteur dans l'accès aux services et une faible personnalisation.

Comment permettre à la STB d'ouvrir ses services bancaires de manière sécurisée et évolutive à travers des API standardisées, afin que les FinTechs puissent développer des solutions innovantes sans interagir directement avec le système bancaire interne ?

1.3.3. Etude et Critique de l'existant

Critère	STB (Tunisie)	BIAT (Tunisie)	BNP Paribas (International)
Architecture	transformation numérique entamée mais infrastructure encore rigide [1]	forte digitalisation avec création d'une filiale dédiée (innovation & technology) [3] [4]	architecture cloud native , API-FIRST , avec joint-venture dans les paiements esteem [5] [6]
Intégration API/Digital	en Progrès , mais encore limitée (STB everywhere)	Plateformes MyBIAT Retail & Corporate centralisant 66 % des virements [3]	FinTech partenaire (Hello bank!), nombreuses API ouvertes
Flexibilité et Agilité	Évolution progressive, mais encore lente	Digitalisation agile avec filiale dédiée, transformation rapide [4]	Agile, investissements massifs en IT (€100M sur 3 ans) [6]
Conformité / Open Banking	Normes en cours d'adoption	Récompensée pour "Meilleure stratégie de digitalisation" en Afrique du Nord [7] [8]	Conformité avancée, ouverture aux FinTechs et innovation [9]
Sécurité	Présente, mais à moderniser	Filiale dédiée renforce la sécurité des systèmes [3]	Infrastructure mature, gestion fine des accès et investissements IT [6]

Tableau 1 : Etude et Critique de l'existant

1.3.4. Solution Proposée

La solution suggérée consiste à établir une plateforme bancaire contemporaine conçue sur la base d'une architecture de microservices, qui encourage la modularité, l'évolutivité et la résilience. Chaque micro-service, dédié à un secteur fonctionnel spécifique (comme les comptes, les cartes, les distributeurs, les notifications et l'authentification), favorise la facilité d'entretien, l'évolutivité ainsi que la résistance aux défaillances.

La communication entre les services se fait selon plusieurs modes complémentaires :

- ❖ APIs REST, utilisées pour l'exposition des services vers l'extérieur et la consommation par les applications clientes (par exemple le front-end Angular ou une fintech).
- ❖ gRPC, privilégié pour les échanges rapides et performants entre microservices internes.
- ❖ RabbitMQ joue le rôle de bus de messages asynchrones pour la diffusion d'événements tels que la création d'un compte ou la mise à jour d'une carte.

Chaque service possède sa propre base de données, sélectionnée en fonction de ses exigences spécifiques (PostgreSQL, MySQL, MongoDB), témoignant ainsi d'une stratégie de persistance polyglotte.

Tous les services sont exposés et protégés par la passerelle API Kong, qui regroupe le routage, l'authentification et la gestion des appels. Un groupe d'outils contemporains (Prometheus, Grafana, Elasticsearch, Kibana) est utilisé pour la supervision et l'observabilité, offrant des fonctionnalités de suivi en temps réel, d'analyse des journaux et de détection préventive des irrégularités.

Pour finir, la plateforme a été entièrement conteneurisée à l'aide de Docker et gérée par Kubernetes, assurant ainsi la portabilité, la disponibilité élevée et la capacité d'évolutivité selon les exigences de la banque.

Cette solution vise donc à offrir à la STB une infrastructure fiable, sécurisée et évolutive, capable de répondre aux nouveaux défis de la digitalisation bancaire et d'intégrer facilement des partenariats avec les fintechs.

1.4. Méthodologie de Travail

1.4.1. Méthodologie SCRUM 'Agile'

La méthodologie Agile SCRUM, largement utilisée dans les projets technologiques modernes, a été choisie pour le développement du projet. Contrairement aux méthodes traditionnelles comme le cycle en V, SCRUM favorise la flexibilité, l'ajustement continu et la collaboration entre les participants. Cette méthode convenait parfaitement au projet pour les raisons suivantes :

- ❖ Les exigences pourraient changer durant le processus de développement (comme l'ajout de nouvelles API, l'intégration du gRPC ou les modifications de sécurité).
- ❖ Le projet englobe divers microservices autonomes, ce qui rend leur déploiement progressif plus aisé.
- ❖ Les retours rapides (tests, validations intermédiaires) ont permis d'affiner constamment la solution.

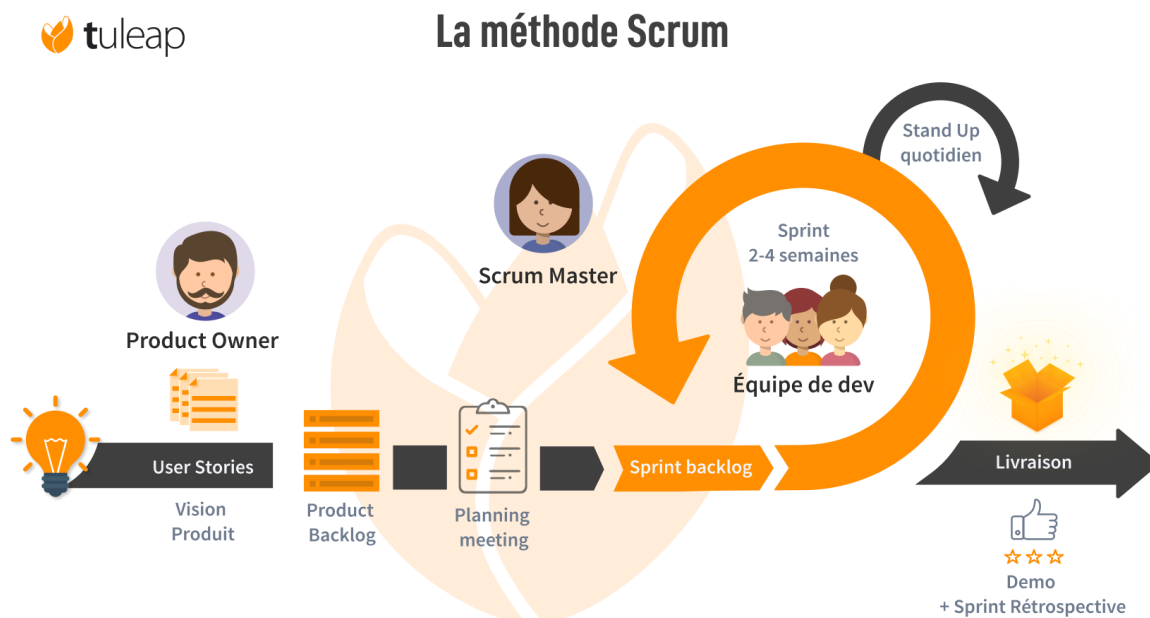


Figure 3: Méthode Scrum (<https://www.tuleap.com>)

1.4.2. Organisation des Sprints

Le développement du projet a été structuré selon la méthodologie **Agile SCRUM**, avec une organisation en huit sprints de trois semaines, répartis sur une durée totale de six mois (du 8 février au 8 août). Chaque sprint a été planifié avec des objectifs clairs et mesurables, permettant d'assurer une progression régulière tout en gardant une flexibilité face aux ajustements nécessaires.

- ❖ **Sprint 0** : Analyse approfondie des besoins, recherche documentaire et choix des technologies (Docker, RabbitMQ, gRPC, Kong Gateway).
- ❖ **Sprint 1**: Développement du microservice **Comptes (Account-Service)**.
- ❖ **Sprint 2**: Implémentation du microservice **Cartes (Card-Service)**.
- ❖ **Sprint 3** : Mise en place du microservice **Guichets automatiques (ATM-Service)**.
- ❖ **Sprint 4**: Développement du microservice **Notifications**.
- ❖ **Sprint 5** : Mise en place de la **communication interservices** via RabbitMQ (asynchrone) et gRPC (synchrone).
- ❖ **Sprint 6** : Intégration de **Kong API Gateway** (sécurité, authentification JWT, rate limiting).
- ❖ **Sprint 7**: Développement du microservice **Authentication** (JWT, OAuth2, SSL).
- ❖ **Sprint 8** : **Déploiement et monitoring** de la plateforme sur un VPS OVH avec Docker, Kubernetes, Prometheus, Grafana, ELK, UFW et SSL.

1.4.3. Diagramme du Gant

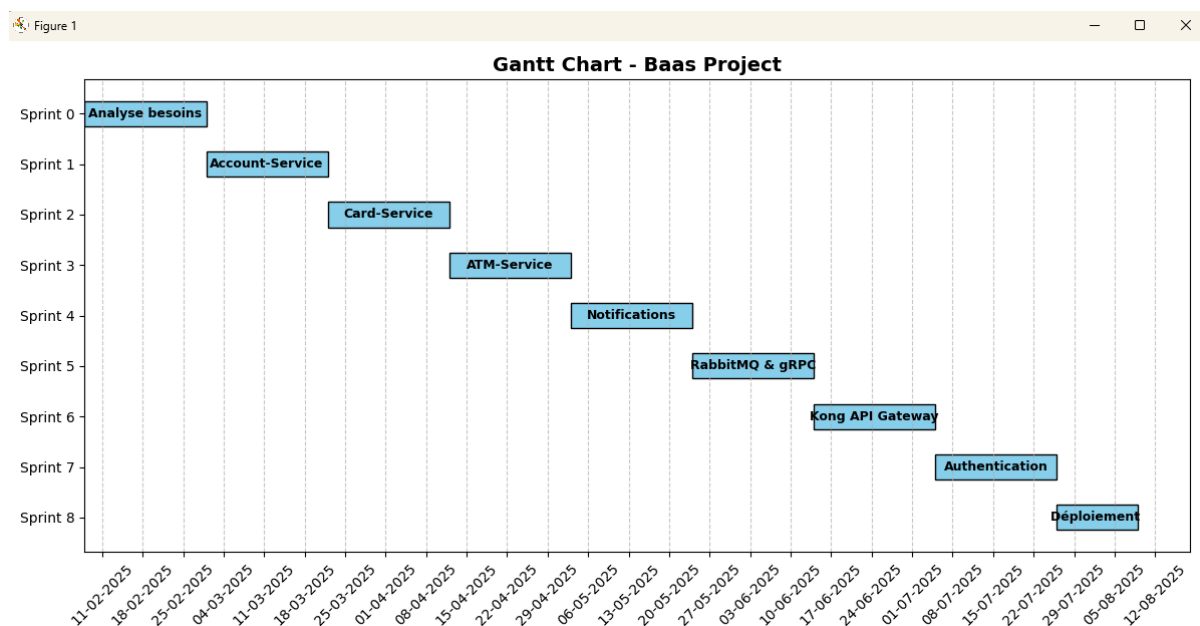


Figure 4: Diagramme du Gant

1.5. Conclusion

Ce premier chapitre a fourni une présentation du cadre global du projet, de l'entité d'accueil et des défis associés à la numérisation bancaire et à la Banque Ouverte. Suite à la détection des contraintes inhérentes aux systèmes actuels, une architecture microservices a été suggérée en tant que solution. Cette dernière incorpore des technologies de pointe comme gRPC, RabbitMQ, Kong Gateway et Docker/Kubernetes.

On a donc opté pour la méthode Agile SCRUM pour assurer une souplesse et un ajustement constant aux exigences. La structuration des sprints sur un semestre a rendu possible l'organisation méthodique des phases de développement, depuis la conception jusqu'à l'implémentation finale.

Chapitre 2 : Étude théorique et conception

2. Chapitre 2 : Étude théorique et conception

2.1. Introduction

Ce chapitre expose l'analyse théorique et la conception de la plateforme bancaire suggérée. C'est le socle technique qui a orienté l'exécution concrète du projet.

Le but est d'établir l'environnement de développement, les décisions technologiques et l'architecture tant logicielle que physique sur laquelle s'appuie la solution.

Dans un environnement de numérisation bancaire, l'élaboration d'une structure robuste, adaptable et sécurisée est essentielle pour assurer l'efficacité du système et satisfaire les exigences des clients et des partenaires fintech.

Par conséquent, nous avons particulièrement veillé à adopter une architecture axée sur les microservices, à exploiter des technologies de communication actuelles comme gRPC et RabbitMQ, ainsi qu'à mettre en place une API Gateway (Kong) pour assurer la sécurité et l'interface des services.

Ce chapitre abordera donc successivement :

- ❖ l'environnement matériel et logiciel utilisé,
- ❖ les technologies et frameworks adoptés,
- ❖ les spécifications fonctionnelles et non fonctionnelles,
- ❖ et enfin l'architecture logique et physique de la solution.

2.2. Environnement de développement et choix techniques

La réussite d'un projet de développement logiciel dépend en grande mesure d'un environnement de développement approprié, tant du point de vue matériel que logiciel. Pour ce projet, la sélection des outils et frameworks s'est faite en prenant en compte des critères de performance, de sécurité, d'interopérabilité et de scalabilité.

2.2.1. Environnement matériel

Le développement et les tests ont été réalisés sur une station de travail disposant des caractéristiques suivantes :

- ❖ **Processeur** : Intel Core i7, 10^e génération
- ❖ **Mémoire vive** : 16 Go RAM
- ❖ **Stockage**: 512 GB SSD NVMe
- ❖ **Système d'exploitation** : Windows 11 Pro pour le développement, Ubuntu 22.04 LTS pour les serveurs de déploiement (VPS OVH)
- ❖ **Réseau** : Connexion haut débit pour les tests distribués et l'intégration continue

Ces caractéristiques assurent une exécution fluide des conteneurs, des outils DevOps ainsi que des environnements de test multi-services

2.2.2. Environnement Logiciel

2.2.2.1. Outils de développement

Visual Studio Code est un éditeur de code source léger, gratuit et open source développé par Microsoft. Il offre une interface moderne, des fonctionnalités puissantes comme l'autocomplétion, le débogage intégré et la gestion des extensions pour divers langages. Grâce à sa rapidité et sa flexibilité, il est largement utilisé par les développeurs pour créer des applications web, mobiles et logicielles.



Figure 5: Logo Visual Studio Code

Visual Studio est un IDE puissant de Microsoft qui prend en charge plusieurs langages comme C#, C++ et Python. Il offre des outils pour coder, tester et déboguer efficacement. Utilisé surtout par les professionnels, il facilite le développement d'applications web, mobiles et desktop.

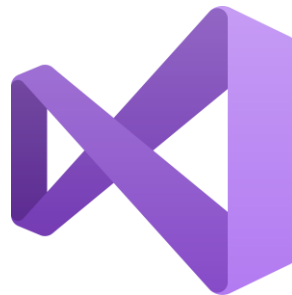


Figure 6: Logo Visual Studio

Insomnia est un outil open source utilisé pour tester et consommer des API. Il permet d'envoyer facilement des requêtes HTTP ou GraphQL, d'analyser les réponses et de gérer l'authentification. Simple et intuitif, il est très apprécié des développeurs pour le développement et le débogage d'APIs.



Figure 7: Logo Insomnia

Swagger est un outil open source qui facilite la conception, la documentation et le test d'API REST. Il permet de décrire les endpoints d'une API avec le standard OpenAPI, offrant ainsi une interface interactive pour tester les requêtes et visualiser les réponses. Très utilisé, il simplifie la collaboration entre développeurs et garantit une meilleure compréhension des services exposés.



Figure 8: Logo Swagger

GitHub est une plateforme en ligne de gestion de code source basée sur Git. Elle permet aux développeurs de stocker, partager et collaborer sur des projets logiciels, avec des fonctionnalités comme le contrôle de version, les issues et les pull-requests. Très populaire dans le monde du développement, elle facilite le travail en équipe et l'open source.

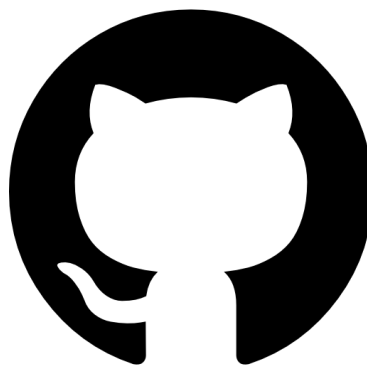


Figure 9: Logo GitHub

Docker Desktop est une application qui permet de créer et de gérer des conteneurs facilement. Elle intègre Docker Engine et une interface simple pour exécuter, tester et déployer des applications. Très pratique, elle aide les développeurs à travailler dans des environnements isolés et reproductibles.



Figure 10: Logo Docker Desktop

RabbitMQ Management Console est une interface web qui permet de gérer et surveiller facilement un serveur RabbitMQ. Elle offre un aperçu des files d'attente, des messages et des connexions en temps réel. Très pratique, elle simplifie le suivi et l'administration du système de messagerie.



Figure 11: Logo Rabbitmq Management Console

2.2.3. Outils de collaboration & DevOps

MobaXterm : application qui fournit un terminal avancé pour accéder à distance aux serveurs via SSH, avec des outils intégrés comme SFTP, X11 et des sessions multiples. Très pratique pour administrer et gérer des systèmes à distance depuis Windows.

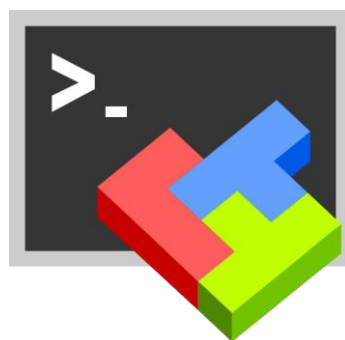


Figure 12: Logo MobaXterm

Kubernetes (kubectl) : outil en ligne de commande utilisé pour interagir avec Kubernetes, permettant de déployer, gérer et superviser des applications conteneurisées sur un cluster. Il facilite l'orchestration, la mise à l'échelle et le suivi des microservices.

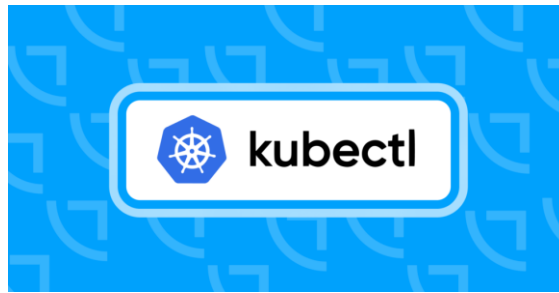


Figure 13: Logo Kube Ctl

Prometheus et Grafana : outils combinés pour surveiller les systèmes et applications. Prometheus collecte et stocke les métriques, tandis que Grafana permet de les visualiser via des tableaux de bord interactifs pour faciliter la supervision en temps réel.



Figure 14: Logo Prometheus et Grafana

Elasticsearch et Kibana : Elasticsearch est un moteur de recherche et d'indexation de données, tandis que Kibana fournit une interface pour visualiser et analyser ces données. Ensemble, ils permettent la centralisation, la recherche et la visualisation des logs et informations système.



Figure 15: Logo Elasticsearch et Kibana

2.2.4. Technologies et Frameworks

2.2.4.1. Backend et Microservices

C# / .NET 8 (ASP.NET Core Web API) : C# est un langage de programmation moderne et orienté objet développé par Microsoft. Avec .NET 8 et ASP.NET Core Web API, il permet de créer des services web performants, sécurisés et multiplateformes, adaptés aux architectures modernes comme les microservices et le cloud.



Figure 16: Logo .NET

gRPC : framework open source de communication à hautes performances développé par Google. Il utilise le protocole HTTP/2 et les fichiers Protocol Buffers (Protobuf) pour définir et échanger des messages, offrant ainsi des appels de procédures distants rapides, sécurisés et adaptés aux microservices.



Figure 17: Logo gRPC

RabbitMQ : système de messagerie open source basé sur le protocole AMQP (Advanced Message Queuing Protocol). Il permet aux applications de communiquer de manière asynchrone via des files d'attente, facilitant la gestion des messages, la scalabilité et la fiabilité dans les architectures distribuées.



Figure 18: Logo RabbitMQ

PostgreSQL : base de données relationnelle open source puissante, idéale pour gérer des données structurées comme les comptes et cartes, avec une forte cohérence et des fonctionnalités avancées.



Figure 19: Logo PostgreSQL

MySQL : base de données relationnelle très utilisée, légère et rapide, adaptée à l'authentification des utilisateurs et à la gestion des agences.



Figure 20: Logo MySQL

MongoDB : base de données NoSQL orientée documents, parfaite pour stocker des données non structurées ou semi-structurées comme les informations des distributeurs automatiques (ATMs).



Figure 21: Logo MongoDB

2.2.4.2. Frontend

Angular 16 : un framework open source conçu par Google pour le développement d'applications web modernes, dynamiques et efficaces. Cette version propose des optimisations au niveau de la performance, de la gestion de la réactivité via les signaux, ainsi qu'une expérience améliorée pour le développeur.



Figure 22: Logo Angular

NGINX : serveur web open source léger et performant, utilisé pour distribuer efficacement des applications frontend comme Angular. Il assure la mise en cache, l'équilibrage de charge et la gestion des requêtes, garantissant une livraison rapide et optimisée des ressources aux utilisateurs.



Figure 23: Logo NGINX

2.2.4.3. Api Gateway et Sécurité

Kong API Gateway : passerelle open source permettant de gérer, sécuriser et surveiller les API. Elle offre des fonctionnalités comme l'authentification, le routage, la limitation de débit et la journalisation, facilitant ainsi la gestion centralisée et la scalabilité des microservices.



Figure 24: Logo KONG Api Gateway

Konga : interface graphique web open source qui facilite la gestion et la configuration de Kong API Gateway. Elle permet d'administrer les services, routes, plugins et consommateurs via une interface visuelle intuitive, sans passer uniquement par la ligne de commande ou l'API de Kong.



Figure 25: Logo KONGA

Cloudflare : service en ligne qui améliore la performance et la sécurité des sites web. Il fournit des fonctionnalités comme le CDN (réseau de diffusion de contenu), la protection contre les attaques DDoS et l'optimisation du trafic, assurant un accès rapide et sécurisé aux utilisateurs.



Figure 26: Logo CLOUDFLARE

Let's Encrypt SSL : fournisseur gratuit de certificats SSL/TLS automatisés, permettant de sécuriser les sites web via le protocole HTTPS. Il simplifie l'installation et le renouvellement des certificats, offrant sécurité et confidentialité aux utilisateurs sans coût supplémentaire.



Figure 27: Logo Let's Encrypt SSL

JWT : JSON Web Token (JWT) est une norme ouverte (RFC 7519) qui définit une méthode compacte et autonome pour les transmissions sécurisées entre tiers d'informations encodées sous forme d'objet JSON [12]



Figure 28: Logo JWT

UFW (Uncomplicated Firewall) : outil de gestion de pare-feu pour Linux, simple et convivial. Il permet de configurer facilement des règles de filtrage pour autoriser ou bloquer le trafic réseau entrant et sortant, assurant ainsi la sécurité du serveur sans avoir à manipuler directement iptables



Figure 29: Logo UFW

Fail2ban : protection contre les attaques par force brute sur SSH et autres services sensibles.



Figure 30: Logo fail2ban

Namecheap est un fournisseur de services de noms de domaine (DNS) et de certificats SSL. Dans ce projet, il est utilisé pour gérer le domaine de l'application et associer les sous-domaines aux services déployés sur le serveur **OVH**. En complément, il facilite l'intégration avec **Cloudflare** afin d'assurer un routage sécurisé et la protection des communications via SSL/TLS.



Figure 31: Logo Namecheap

2.3. Spécifications fonctionnelles et non fonctionnelles

2.3.1. Spécifications fonctionnelles

Les spécifications fonctionnelles décrivent les tâches que le système doit réaliser, en tenant compte des exigences formulées. Dans le contexte de ce projet, cela se manifeste par les capacités fournies par les microservices :

- ❖ Gestion des Comptes
 - Création, modification et suppression de comptes.
 - Consultation du solde et des transactions.
 - Génération d'extraits de compte
- ❖ Gestion des cartes
 - Création et activation de cartes bancaires.
 - Consultation du statut d'une carte.

- ❖ Gestion des guichets automatiques
 - Localisation des distributeurs automatiques via coordonnées GPS.
 - Consultation du statut (en service, hors service).
- ❖ Authentification et sécurité
 - Inscription et authentification par mot de passe ou réseaux sociaux (Google OAuth).
 - Gestion des rôles et permissions (administrateur, client, fintech).
 - Génération et validation de tokens JWT.
- ❖ gestion des branches
 - Localisation des branches de la banque via coordonnées GPS.
 - Consultation du statut .
- ❖ Notifications
 - Envoi de notifications (emails, logs) lors d'événements clés (création de compte, transaction, mise à jour d'ATM).
 - Système basé sur **RabbitMQ** (asynchrone, event-driven).
- ❖ Api Gateway
 - Routage des requêtes vers les microservices.
 - Sécurisation des API (API Key, OAuth2, JWT).
 - Gestion des quotas et rate-limiting.
- ❖ Frontend
 - Interface responsive multilingual (FR/EN/AR).
 - Consultation des documentation des services
 - Tableau de bord centralisé avec authentification utilisateur.

2.3.2. Spécifications non fonctionnelles

Elles se rapportent à la qualité du système et ne se limitent pas aux fonctionnalités :

- ❖ Scalabilité
 - Architecture cloud-native microservices supportée par Docker et Kubernetes
- ❖ Disponibilité & Fiabilité
 - Surveillance constante grâce à Prometheus et Grafana.
 - Consolidation des journaux grâce à ELK (Elasticsearch, Logstash, Kibana).
 - Objectif de taux de disponibilité : 99,5 %.
- ❖ Interopérabilité
 - REST APIs conformes aux spécifications OpenAPI/Swagger.
 - compatibilité avec des FinTech externes (clients Python/Flask, Postman).
- ❖ Maintenabilité
 - Code organisé en microservices indépendants.
 - CI/CD avec GitHub **Actions** pour faciliter les mises à jour.

- ❖ Portabilité
 - déploiement reproductible grâce à Docker Compose (en local) et Kubernetes (sur OVH Cloud).
- ❖ Performance
 - Temps de réponse inférieur à 200 ms pour les opérations courantes
- ❖ Sécurité
 - Transmission sécurisée grâce à TLS/SSL.
 - Authentification améliorée via JWT et OAuth2.
 - Sécurisation côté serveur grâce à CloudFlare , UFW (firewall), Fail2ban et la clé SSH.

2.4. Architecture de l'application

2.4.1. Architecture logique

La structure logique de notre solution bancaire : Bank As A Service (BaaS) est fondée sur une approche de microservices modulaires orchestrée grâce à Kubernetes, et mise en avant via une passerelle d'API (Kong). Cette structure est conçue pour être évolutive, observable et sécurisée, tout en établissant une séparation nette des responsabilités entre les diverses couches logiques de l'application.

2.4.1.1. Composants principaux

★ Frontend & API Gateway

- Frontend Angular (Pod) : Interface web adaptative qui permet aux clients fintech et aux utilisateurs finaux d'interagir avec les services bancaires.
- Kong Ingress Controller et Konga Dashboard : Un point d'accès crucial aux API qui assure le routage, la gestion des clés API, ainsi que la protection contre les requêtes externes.

★ Microservices (Business Logic Layer)

- Account-Service: Gestion des comptes clients, des soldes et transactions.
- Branch-Service : Gestion des agences et des employés.
- Card-Service : Gestion de l'émission et du statut des cartes bancaires.
- ATM-Service : Gestion des distributeurs automatiques, retraits, dépôts et géolocalisation.
- Authenticator-Service : Gestion de l'authentification et des sessions utilisateurs.
- Notification-Service : Gestion des événements internes, déclenchement de notifications et alertes.

★ Bases de données (Data Layer)

Chaque micro service utilise une base de données adaptée à son besoin :

- PostgreSQL pour Account et Card
- MySQL pour Branch et Authenticator
- MongoDB pour ATM
- Redis pour le cache et la gestion des sessions

★ Messaging & Eventing

- RabbitMQ : Facilite la communication asynchrone entre microservices, en particulier pour les notifications basées sur des événements (création de compte, mise à jour de DAB, etc.).

★ Observabilité

- Prometheus et Grafana : Monitoring et tableaux de bord de performance.
- Elasticsearch et Kibana : Indexation des logs et exploration en temps réel.

2.4.1.2. Flux logique

Les microservices sont accessibles via le **Kong API Gateway** par les clients externes, comme les fintech SaaS, en utilisant une clé API sécurisée.

Les microservices échangent des informations entre eux et avec la couche de persistance par le biais d'appels internes et d'événements **RabbitMQ**.

Pour garantir la traçabilité et la robustesse du système, les événements et les données critiques sont surveillés et consignés.

2.4.2. Résumé graphique

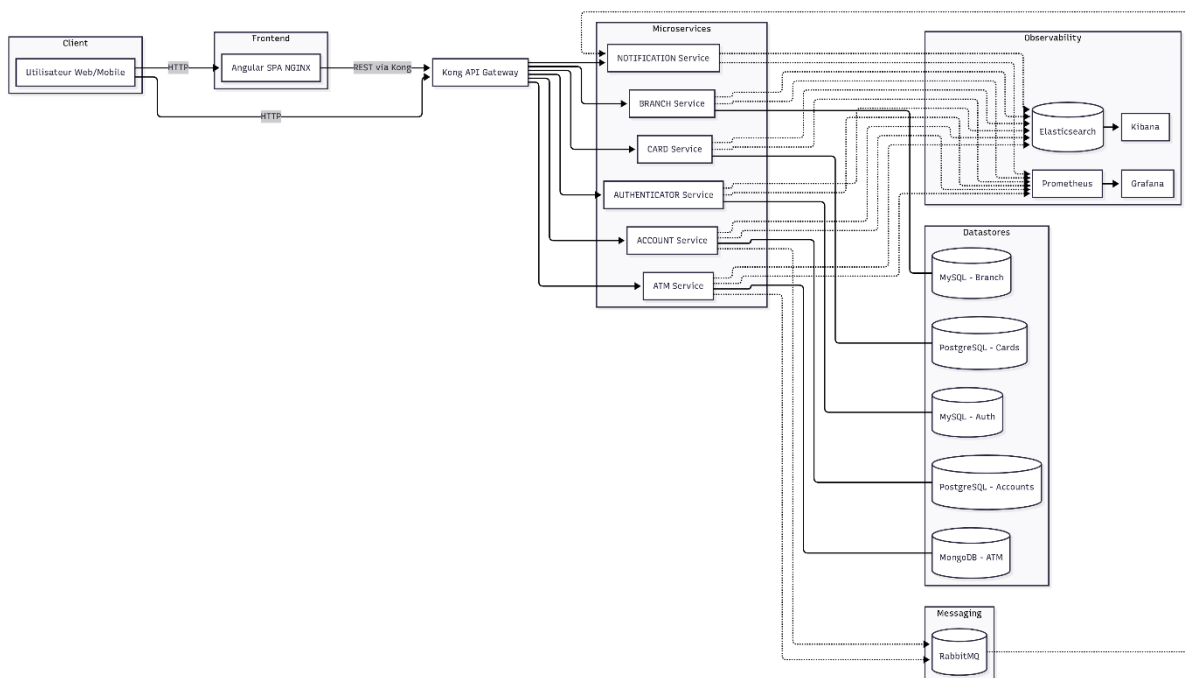


Figure 32: Architecture logique

2.4.3. Architecture physique

L'architecture physique de la solution repose sur un **serveur privé virtuel (VPS)** hébergé chez **OVH**, qui constitue l'environnement matériel principal d'exécution. Ce serveur offre les ressources nécessaires (CPU, mémoire et stockage) pour héberger l'ensemble de l'infrastructure logicielle.

À l'intérieur de ce VPS, un **cluster Kubernetes** est déployé afin d'orchestrer et de gérer l'exécution des différents composants de la plateforme. Kubernetes assure la montée en charge, la tolérance aux pannes et la flexibilité en permettant le déploiement des services sous forme de **Pods** et de **services** interconnectés.

L'ensemble des microservices du système bancaire (Account-Service, Branch-Service, Card-Service, ATM-Service, Notification-Service, Authenticator, etc.) est exécuté dans ce cluster Kubernetes. Chaque microservice est packagé sous forme de **conteneur Docker** et déployé indépendamment, ce qui permet une gestion efficace du cycle de vie des applications.

Les **bases de données** (MySQL, PostgreSQL, MongoDB, Redis, RabbitMQ) sont également déployées dans le cluster Kubernetes, garantissant une communication rapide et sécurisée avec les microservices.

En complément, les outils de **supervision et de journalisation** (Prometheus, Grafana, Kibana, Elasticsearch) sont intégrés dans l'architecture afin d'assurer la visibilité, le suivi et l'analyse des performances du système.

L'accès des utilisateurs finaux s'effectue via un **frontend Angular** hébergé dans Kubernetes et exposé au public à travers un **Kong API Gateway**. La sécurité est renforcée par l'intégration de **Cloudflare** et d'un **pare-feu UFW**, ainsi que par la gestion du trafic chiffré via le protocole **TLS/SSL**.

En résumé, l'architecture physique s'articule autour de :

- ★ **OVH VPS Server** : hébergement de l'infrastructure.
- ★ **Cluster Kubernetes** : orchestration et exécution des services.
- ★ **Microservices conteneurisés** : cœur fonctionnel de l'application.
- ★ **Bases de données distribuées** : stockage et persistance des données.
- ★ **Outils de monitoring et logging** : suivi et observabilité.
- ★ **Passerelles et sécurité réseau** : protection et gestion des accès.

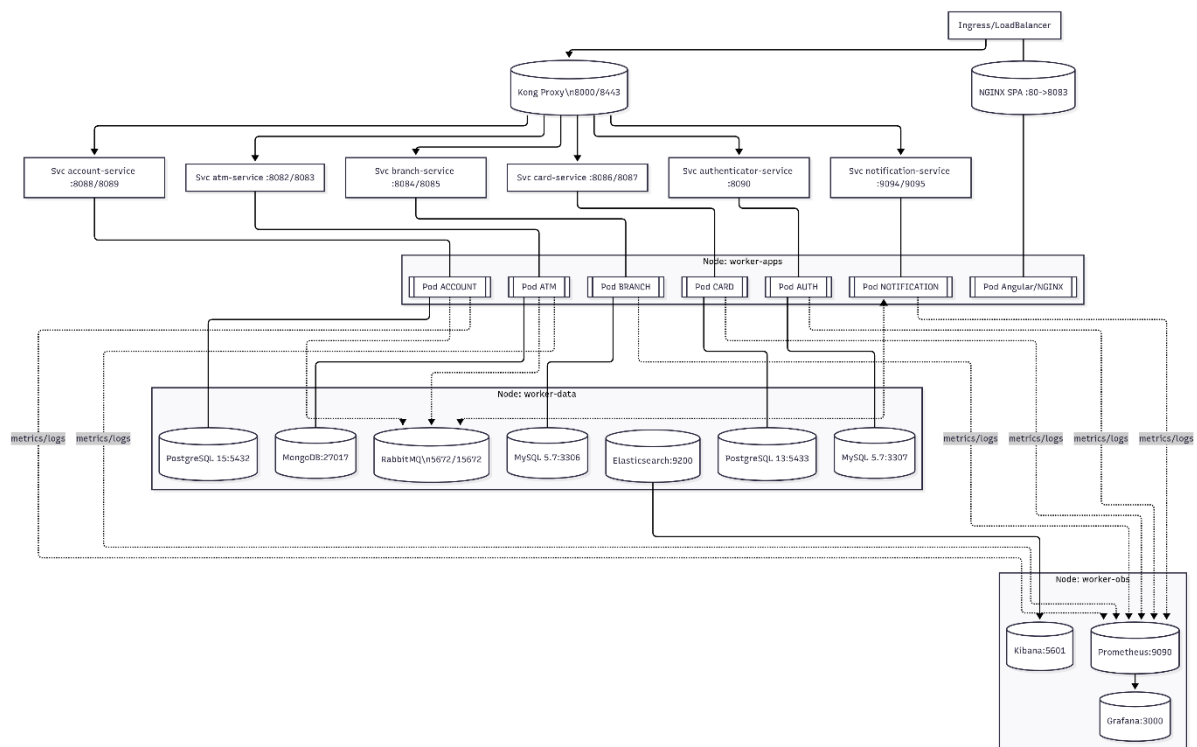


Figure 33: Architecture physique

2.5. Conclusion

Grâce à cette recherche théorique et à la phase de conception, nous avons réussi à formuler une vision organisée et précise de la solution à réaliser, en déterminant tant son architecture physique que logique ainsi que les technologies les plus appropriées pour sa mise en œuvre. Cette approche a conduit à la conversion des exigences fonctionnelles et non fonctionnelles en une structure architecturale unifiée, apte à satisfaire les restrictions liées à la performance, à la sécurité et à l'évolutivité du projet.

Cette étape initiale est donc le fondement sur lequel se basera la phase suivante. Dans le chapitre suivant, nous traiterons de la mise en œuvre pratique du système, faisant la transition de la conception théorique vers l'implémentation réelle des divers éléments et services, tout en respectant les décisions architecturales prises.

Chapitre 3 : Réalisation du projet

3. Chapitre 3 : Réalisation du projet

3.1. Sprint 0 : Analyse et spécification des besoins

3.1.1. Introduction

Le Sprint 0 représente l'étape préparatoire du projet Banking Solution – Banking as a Service (BAAS).

Il sert à définir les buts, à repérer les intervenants, à détailler les exigences fonctionnelles et non fonctionnelles, ainsi qu'à établir les fondements techniques de la solution.

Cette phase est cruciale pour minimiser les risques et assurer une planification efficace des prochains sprints.

3.1.2. Analyse des besoins

3.1.2.1. Identification des acteurs

Dans le contexte de ce projet de solution bancaire - Banking as a Service (BAAS), diverses parties prenantes participent pour garantir l'efficacité du système et satisfaire aux divers besoins.

Chaque participant a un rôle précis, qu'il s'agisse d'un client qui utilise les services bancaires, d'un administrateur en charge de la sécurité, ou d'un ingénieur DevOps chargé du déploiement et de la surveillance technique.

Il est primordial de reconnaître précisément ces acteurs pour guider la création du système et garantir une distribution appropriée des responsabilités.

- ★ **Client bancaire** : utilise les services (compte, carte, ATM, transactions).
- ★ **Employé de banque** : gère les agences et supervise certaines opérations.
- ★ **Administrateur (Admin)** : supervise la sécurité, la gestion des API et l'observabilité.
- ★ **DevOps** : déploie et maintient l'infrastructure (Kubernetes, CI/CD, monitoring).
- ★ **Fintech (partenaire externe)** : consomme les API exposées par la plateforme.
- ★ **Système** : assure la résilience, la sécurité (TLS, DNS, HTTPS) et la haute disponibilité.

3.1.3. Backlog Global

Role	User story	tâches principales	estimation
Fintech	En tant que fintech, je veux consommer les APIs bancaires (compte, cartes, ATM, branches) afin d'intégrer ces services dans mon application	Développer microservices (Compte, Carte, ATM, Transactions) en .NET 8 - Connecter aux bases PostgreSQL / MongoDB / MySQL - Exposer via Kong API Gateway - Rédiger documentation Swagger/OpenAPI	13 pts
client	En tant que fintech, je veux	Développer page "API Catalog" (UI Angular) - Lister endpoints exposés par Kong	5 pts

	parcourir un catalogue d'API disponibles afin de choisir les services pertinents.	- Lier avec documentation Swagger interactive	
fintech	En tant que fintech, je veux souscrire à une offre d'API afin d'obtenir mes clés d'accès sécurisées.	- Implémenter système de gestion des plans/offres - Génération automatique d'API Keys / Tokens JWT - Dashboard UI Angular pour gestion d'abonnement	8 pts
fintech	En tant que fintech, je veux tester facilement les APIs afin de vérifier leur intégration	- Intégrer Insomnia/Postman Collections - Créer client de test (Flask/Python sandbox) - Fournir environnement de test (sandbox)	5 pts
fintech	En tant que fintech, je veux recevoir des notifications (webhooks, RabbitMQ, emails) en cas d'événements importants (paiement, échec de la transaction).	- Implémenter service Notifications (RabbitMQ) - Webhooks REST/gRPC callbacks - Intégrer Email/SMS providers	8 pts
fintech	En tant que fintech, je veux avoir un portail développeur pour gérer mes accès, consulter la documentation et monitorer mon utilisation d'API.	- Développer un Developer Portal (Angular) - Affichage quotas, logs d'appels - Gestion des clés d'accès (rotation, révocation)	8pts
fintech	En tant que fintech, je veux avoir des métriques de consommation (API usage,	- Implémenter API usage tracking - Intégration Grafana pour reporting fintech - Prévoir export PDF/Excel	5pts

	facturation, SLA) afin de suivre mes coûts		
Employé banque	En tant qu'employé, je veux gérer les agences bancaires afin de maintenir des données à jour	En tant qu'employé, je veux gérer les agences bancaires afin de maintenir des données à jour	5pts
Admin	En tant qu'admin, je veux gérer les API via Kong afin d'assurer la sécurité et la gouvernance.	Déployer Kong + Konga - Créer services, routes, consumers - Activer plugin API Key, JWT et rate-limiting	8pts
Admin	En tant qu'admin, je veux assurer l'observabilité du système afin de surveiller la performance	Déployer Prometheus + Grafana - Centraliser logs avec Elasticsearch/Kibana - Configurer alertes	8pts
Admin	En tant qu'admin, je veux sécuriser l'infrastructure afin de protéger les données	Activer TLS sur Kong - Configurer UFW firewall - Sécuriser SSH + Fail2ban - DNS Cloudflare + WAF	8pts
Admin	En tant qu'admin, je veux assurer la conformité Open Banking / PSD2 afin que la plateforme soit utilisable légalement par des FinTechs	- Implémenter OAuth / OpenID Connect	5pts

Devops	En tant qu'ingénieur DevOps, je veux déployer le système sur Kubernetes OVH afin d'assurer scalabilité et résilience	Écrire manifests (Deployment, Service, Ingress) - Configurer volumes persistents - Monitoring cluster	8pts
Devops	En tant qu'ingénieur DevOps, je veux mettre en place un pipeline CI/CD afin d'automatiser le déploiement.	- GitHub Actions ou Jenkins - Build & push Docker images - Automatiser déploiement sur K8s	8pts
Devops	En tant que système, je veux assurer sécurité réseau & disponibilité afin de garantir un accès fiable aux fintechs	- Enregistrer domaine (Namecheap) - Configurer DNS Cloudflare - Gérer certificats TLS (Let's Encrypt)	5pts
Fintech	En tant que fintech, je veux consommer les APIs bancaires sécurisées (TLS, HTTPS) afin d'intégrer les services sans risque	- Activer HTTPS sur Kong avec TLS (Let's Encrypt) - Forcer redirection HTTP→HTTPS - Tester via Postman/Insomnia	5pts
Admin	En tant qu'admin, je veux protéger l'accès au serveur via un pare-feu UFW afin de bloquer les accès non autorisés.	- Configurer UFW (autoriser uniquement 22/80/443/8000) - Bloquer ports sensibles - Vérifier règles persistantes	3pts

Admin	En tant qu'admin, je veux renforcer la sécurité SSH avec Fail2ban afin d'empêcher les attaques par force brute.	- Installer Fail2ban sur VPS - Configurer jail SSH	3pts
Admin	En tant qu'admin, je veux protéger le domaine et les API via Cloudflare afin de bénéficier d'un WAF et de la protection DDoS.	- Configurer domaine (Namecheap → Cloudflare) - Activer SSL/TLS flexible/full strict - Activer protection DDoS & WAF Cloudflare	5pts
Admin	En tant qu'admin, je veux assurer une observabilité complète du système pour détecter les anomalies.	Déployer Prometheus + Grafana (métriques) - Centraliser logs via Elasticsearch/Kibana - Configurer alertes de sécurité (ex: intrusion)	8pts
Système	En tant que système, je veux assurer une infrastructure résiliente et sécurisée afin de garantir la disponibilité et la conformité.	TLS end-to-end (client → Cloudflare → Kong → microservices) - Rotation automatique des certificats (Let's Encrypt)	8 pts

Tableau 2: Backlog Global

3.1.4. Conception globale

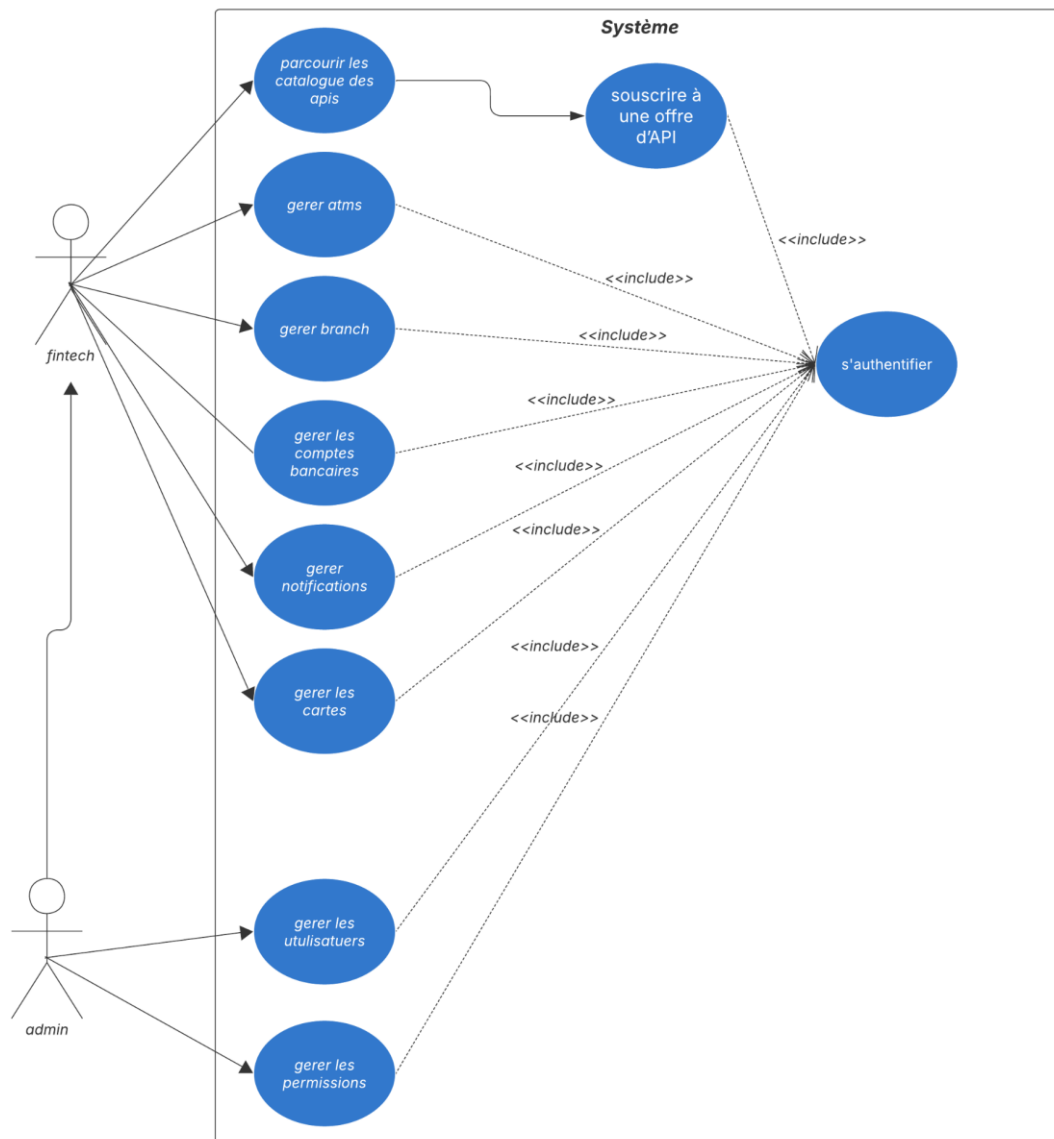


Figure 34:Diagramme de cas d'utilisation

3.2. Sprint 1: Microservice Comptes (Account-Service)

3.2.1. Introduction :

Ce premier sprint est consacré à l'implémentation du microservice Comptes (Account-Service).

Ce service est le fondement du système Open Banking, car il gère toutes les informations liées aux comptes bancaires : mise en place, consultation, solde, routage, caractéristiques et titulaires. Il présente des API REST conformes à la norme Open Banking et utilise Entity Framework Core pour assurer la persistance des données au sein d'une base de données PostgreSQL.

3.2.2. Backlog du sprint

Rôle	User Story	Tâche	Estimation
Fintech	En tant que fintech, je veux recupérer la liste des comptes d'une banque pour intégrer les données dans mon app	Implémenter+ <u>GET/banks/{bankId}/accounts</u> - Exposer via Kong - Ajouter Swagger Doc	5 pts
Fintech	En tant que fintech, je veux consulter les soldes globaux d'une banque pour afficher la situation financière.	- Implémenter <u>GET/banks/{bankId}/balances</u> - Mapper la logique DB - Ajouter tests unitaires	8 pts
Fintech	En tant que fintech, je veux obtenir les détails d'un compte spécifique pour alimenter mes tableaux de bord.	Implémenter <u>GET/banks/{bankId}/accounts/{accountId}</u>	5 pts
Fintech	En tant que fintech, je veux créer un compte bancaire via API pour proposer des services automatisés.	Implémenter la méthode <u>POST/banks/{bankId}/accounts</u>	8 pts
Fintech	En tant que fintech, je veux ajouter un attribut produit à un compte pour enrichir les données de mes clients.	- Implémenter <u>POST/banks/{bankId}/accounts/{accountId}/products/{productCode}/attribute</u>	5 pts
Fintech	En tant que fintech, je veux recupérer les comptes détenus (held accounts) afin de gérer les contrats.	- Implémenter <u>GET/banks/{bankId}/accounts-held</u>	5 pts
Fintech	En tant que fintech, je veux accéder aux agents bancaires afin de connecter mon app aux	- Implémenter <u>GET/banks/{bankId}/agents</u> - Implémenter <u>GET/banks/{bankId}/agents/{agentId}</u>	8 pts

	interlocuteurs humains.	- Sécuriser accès	
Fintech	En tant que fintech, je veux utiliser le mode fast-firehose pour streamer des comptes en temps réel.	- Implémenter <u>GET /management/banks/{bankId}/fast-firehose/accounts</u> - Gérer performance et pagination	13 pts
Admin	En tant qu'admin, je veux superviser l'accès aux API afin de garantir la sécurité.	Activate plugins Kong (API Key / JWT) - Définir RBAC - Ajouter logs via Elasticsearch/Kibana	8 pts
DevOps	En tant que DevOps, je veux déployer ces endpoints sur Kubernetes pour la scalabilité.	Créer <u>deployment.yaml</u> et <u>service.yaml</u> - Configurer persistance DB	8 pts
Sécurité	En tant que responsable de la sécurité, je veux sécuriser toutes les API avec TLS, Firewall et Fail2ban.	Certificat TLS (Let's Encrypt / Cloudflare) - Configurer UFW - Activer Fail2ban sur VPS OVH	3 pts

Tableau 3: Backlog Sprint 1

3.2.3. Conception du sprint

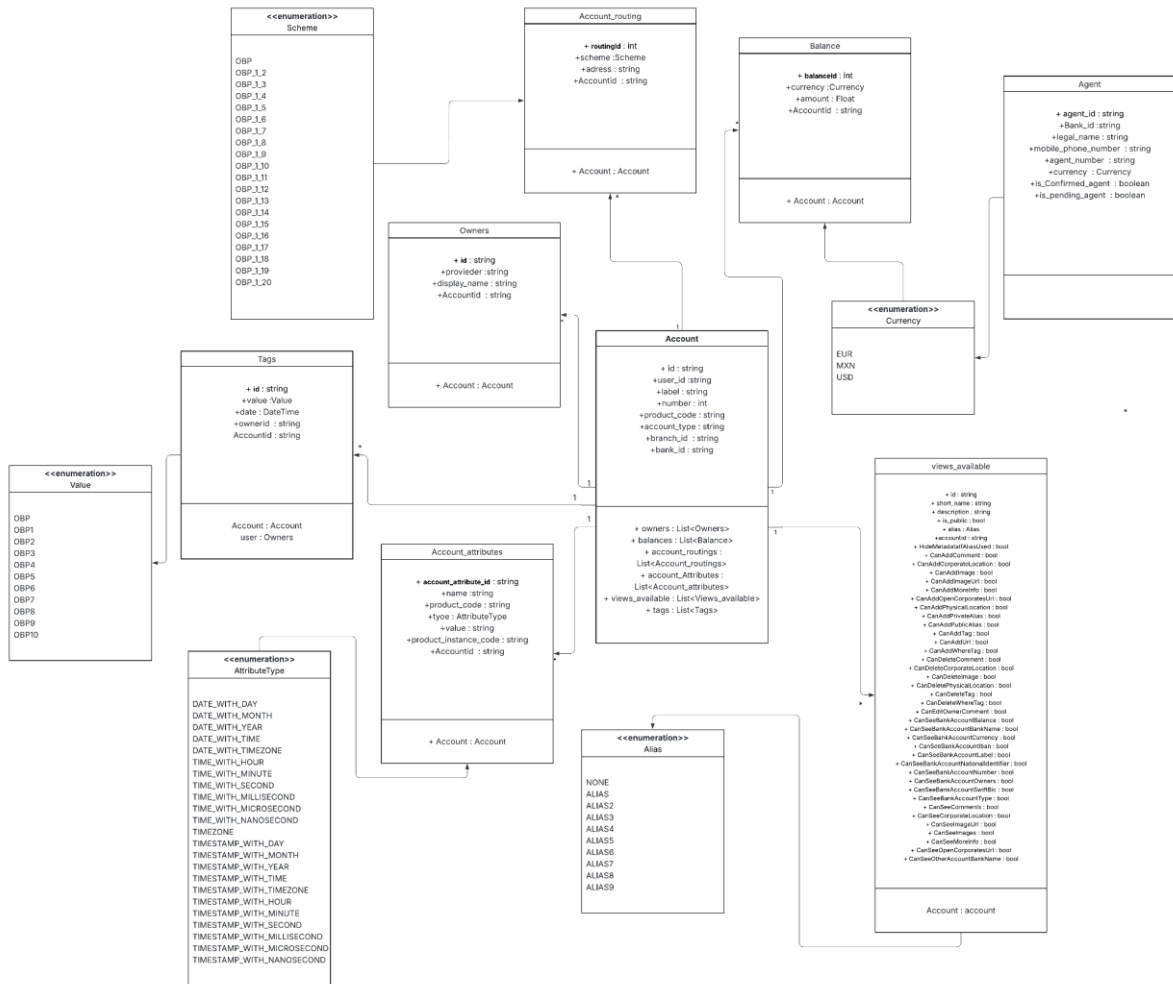


Figure 35: Diagramme de classe (Account Microservice)



Figure 36: Diagramme de séquence (Account Microservice)

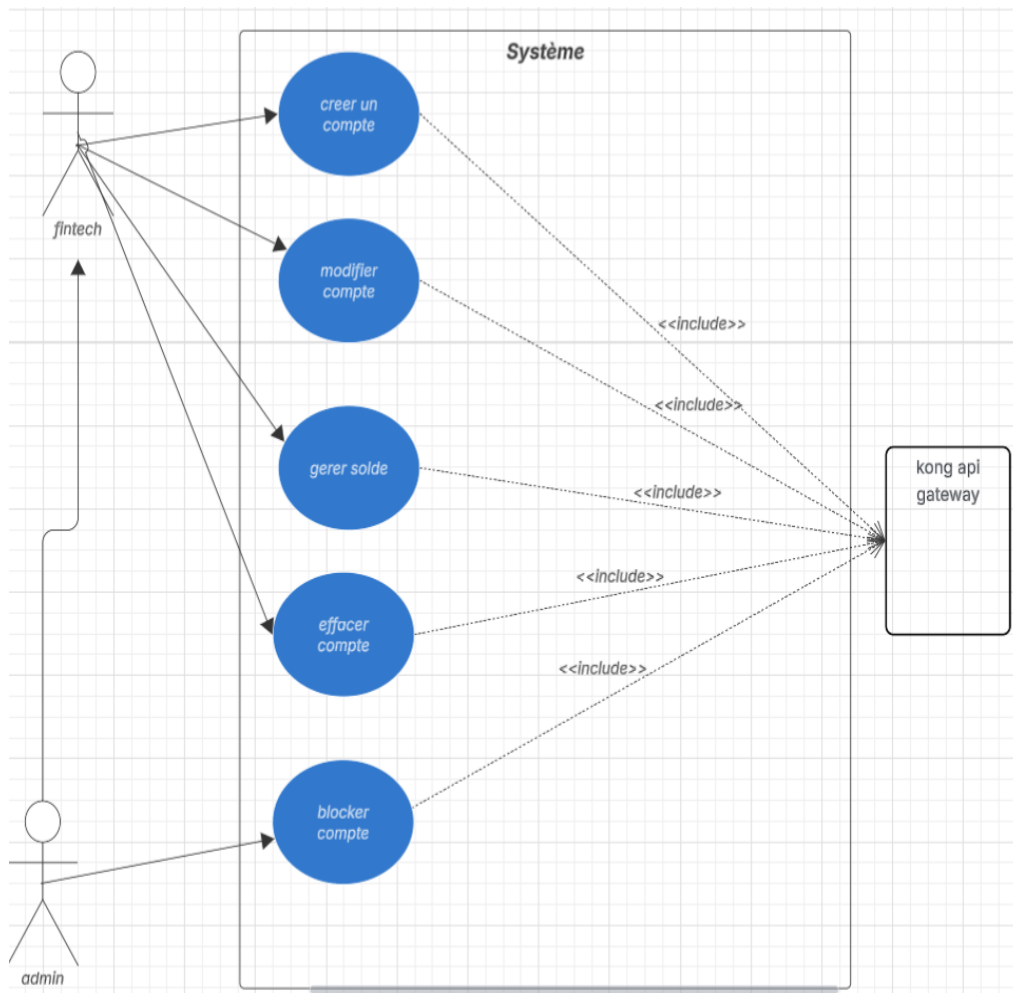


Figure 37: Diagramme de cas d'utilisation(Account Microservice)

3.2.4. Revu du Sprint

Au terme de ce premier sprint, l'équipe a pu livrer une première version stable et opérationnelle du **microservice Comptes (Account-Service)**. Ce service constitue un élément central du système Open Banking, puisqu'il permet de gérer l'ensemble des opérations liées aux comptes bancaires, qu'il s'agisse de leur création, de la consultation des soldes, de l'historique des transactions ou encore de l'accès aux agents bancaires.

D'un point de vue fonctionnel, les principales **user stories du backlog** ont été réalisées avec succès. Les API de base ont été implémentées et exposées via **Kong API Gateway**, garantissant ainsi une sécurisation et un routage cohérent des requêtes. Les endpoints permettant la création d'un compte bancaire, la récupération de la liste des comptes d'une banque, la consultation des soldes globaux, ou encore la consultation des détails d'un compte spécifique, ont été développés, testés et validés à travers Swagger et Postman. La fonctionnalité permettant l'ajout d'attributs produits à un compte a également été intégrée, offrant une flexibilité accrue dans la gestion des données clients.

3.2.5. Démonstration

3.2.5.1. Démonstration des apis

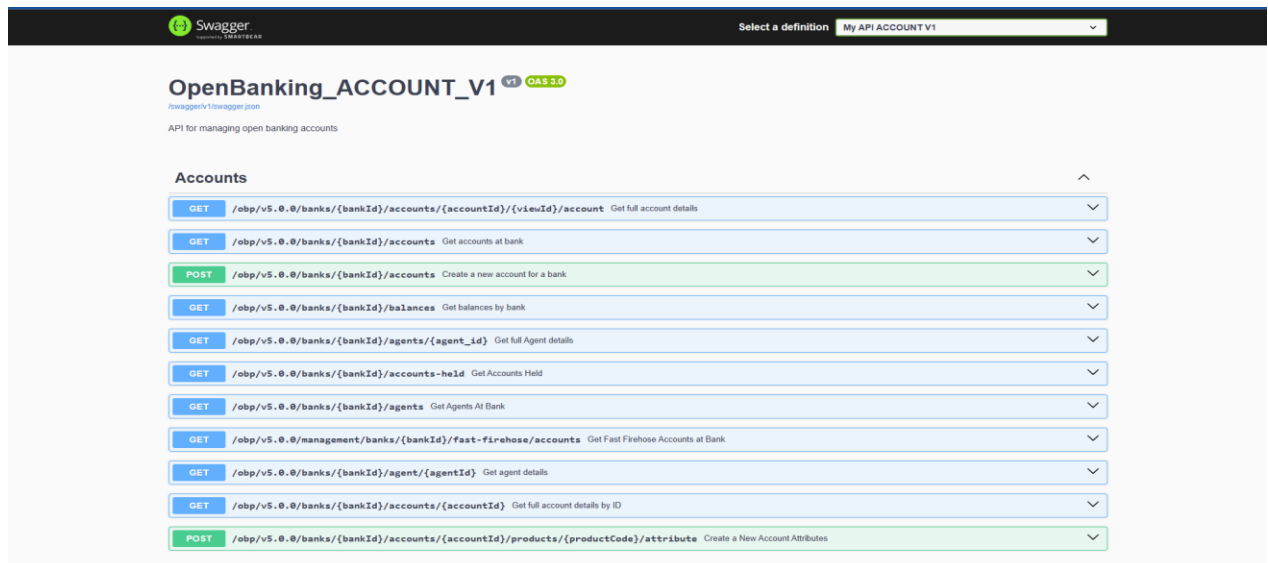


Figure 38: Documentation Api

3.2.5.2. Intégration Account service avec kong

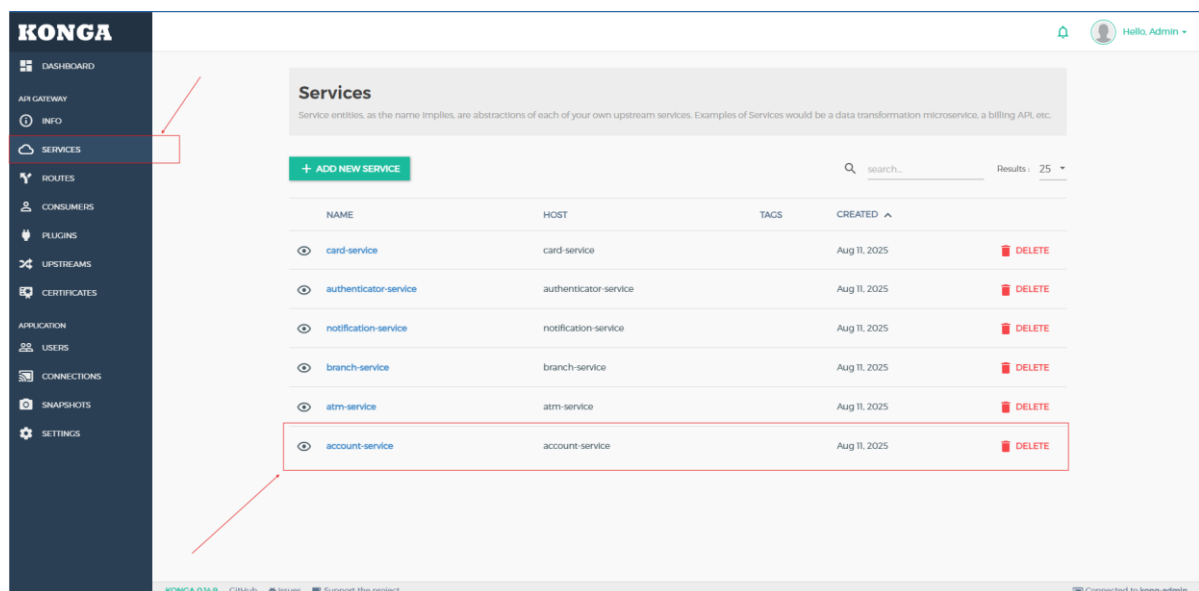


Figure 39: Intégration Account service avec kong

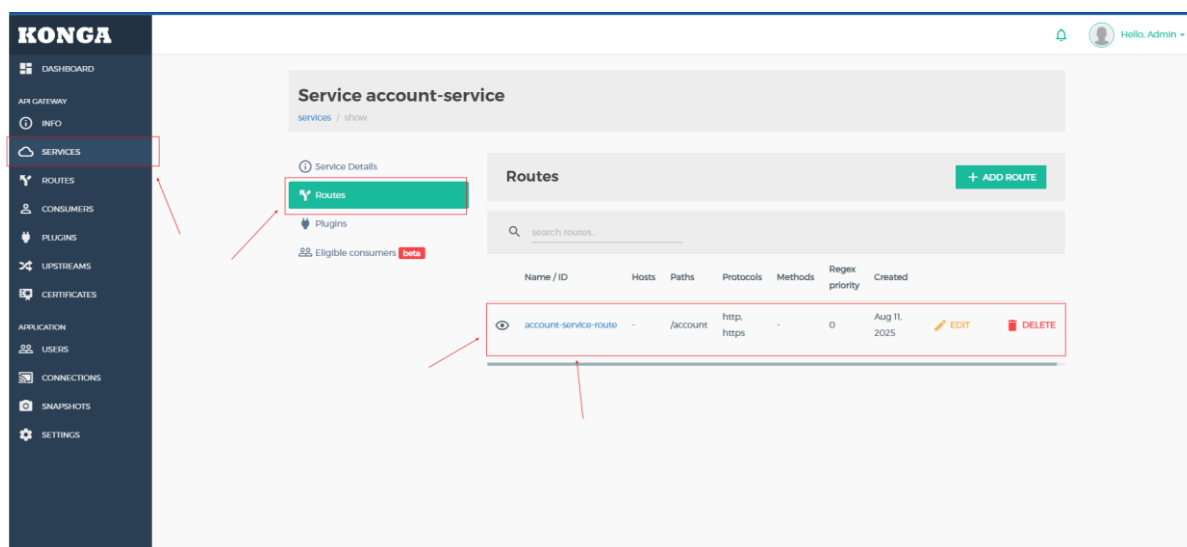


Figure 40: Intégration Account service avec kong

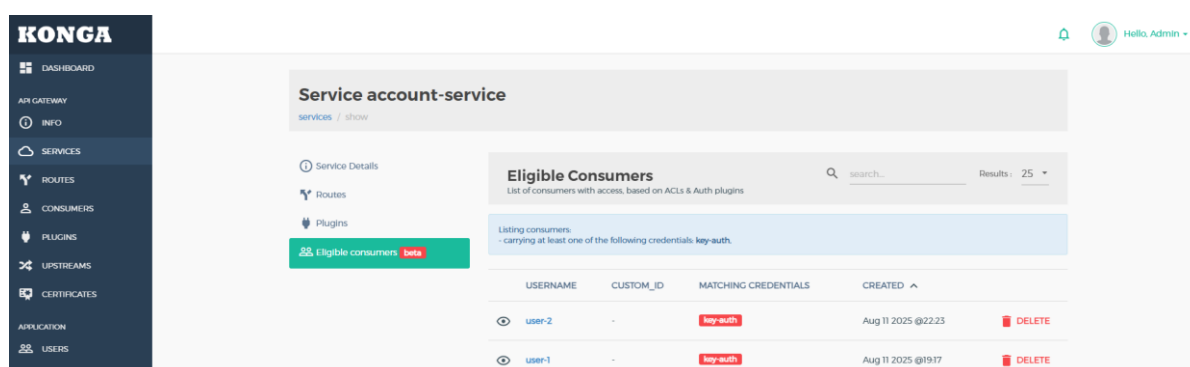


Figure 41: Exemple des fintech (Consumers) qui consomment le service

3.2.6. Burndown Chart

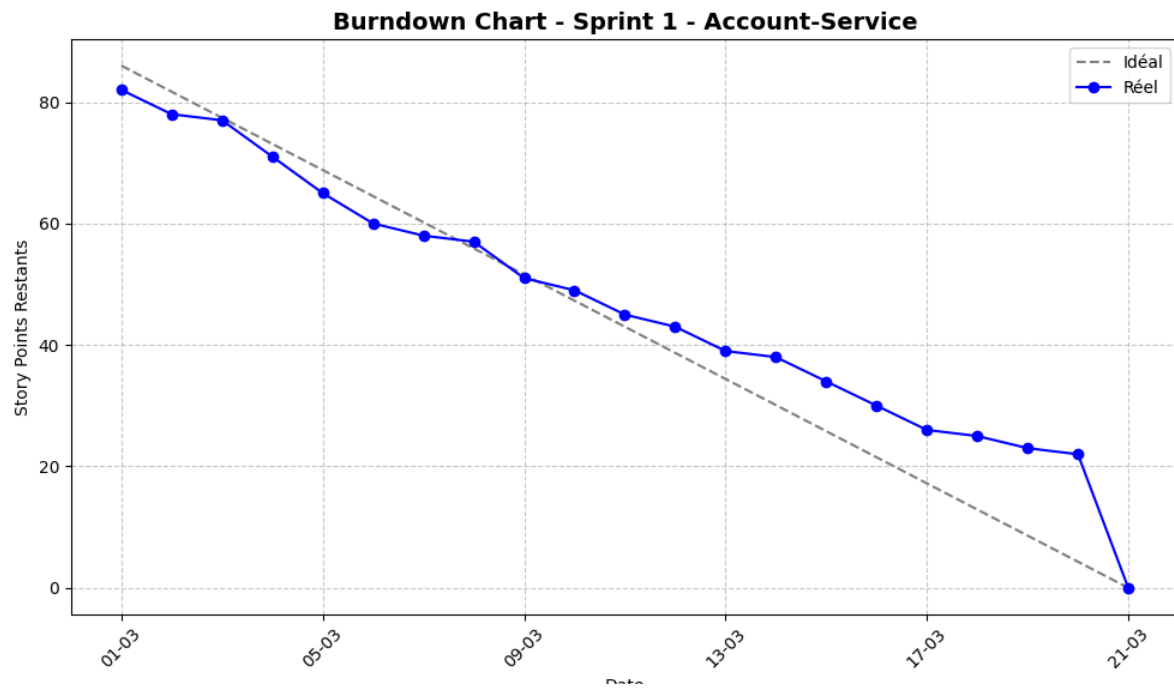


Figure 42: Burndown chart, Sprint 1

3.3. Sprint 2: Microservice Cartes (Card-Service)

3.3.1. Introduction :

Ce sprint s'inscrit dans le cadre du développement de la solution **Banking-as-a-Service (BAAS)**, une plateforme bancaire cloud-native composée de microservices (Account, ATM, Branch, Card, Authentication, Notification, etc.), déployée via **Docker** et **Kubernetes**.

L'objectif principal du sprint est de mettre en place le **microservice Cartes (Card-Service)**.

Il permet la gestion des cartes bancaires (émission, statut, association aux comptes) et leur consommation via une **API REST exposée par Kong API Gateway**.

Ce sprint ne se limite pas à la mise en place du Card-Service mais inclut aussi :

- ❖ La communication inter-services avec **gRPC** (Card-Service ↔ Account-Service).
- ❖ L'exposition sécurisée via **Kong Gateway** avec **API Key Authentication**.
- ❖ L'intégration dans la pile d'observabilité (Prometheus, Grafana, Elasticsearch, Kibana).

3.3.2. Backlog du sprint

Rôle	User Story	Tâche	Estimation
Développeur Backend	En tant qu'utilisateur, je veux que mes cartes soient stockées et liées à mes comptes	Créer le modèle Card et la base PostgreSQL	8 pts
Développeur Backend	En tant que service, je veux pouvoir récupérer toutes les cartes d'un utilisateur	Implémenter le CardRepository	5 pts
Développeur Backend	En tant que service, je veux enrichir mes cartes avec les infos des comptes associés	Implémenter la communication gRPC avec Account-Service	8 pts
Développeur Backend	En tant qu'utilisateur, je veux consulter mes cartes et détails de comptes	Développer les endpoints REST (GET /cards, GET /cards/{id})	10 pts
Développeur Backend	En tant que client externe, je veux consommer l'API de manière sécurisée	Exposer Card-Service via Kong (service, route, plugin key-auth)	7 pts
Testeur QA	En tant qu'utilisateur, je veux être informé en cas d'erreur	Tester les erreurs gRPC + API Gateway (clé invalide, compte introuvable)	5 pts

Scrum Master	En tant qu'équipe, je veux visualiser l'avancement et l'état du microservice	Suivi via Burndown Chart, monitoring avec Prometheus/Grafana	3 pts
Product Owner	En tant que PO, je veux vérifier que le service est exploitable via API Gateway	Démonstration avec Swagger + consommation via client Python Flask	4 pts

Tableau 4: Backlog Sprint 2

3.3.3. Conception du sprint

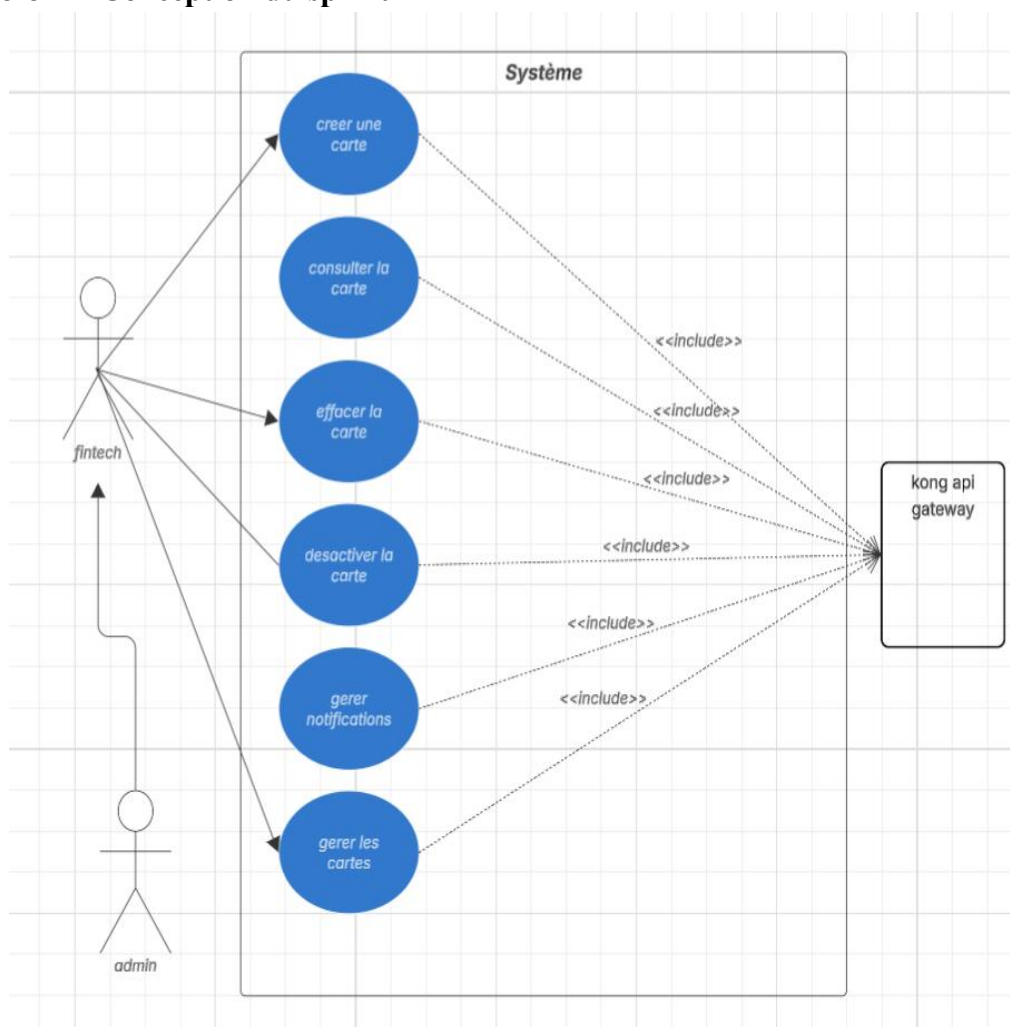


Figure 43: Diagramme de cas d'utilisation (Card Service)

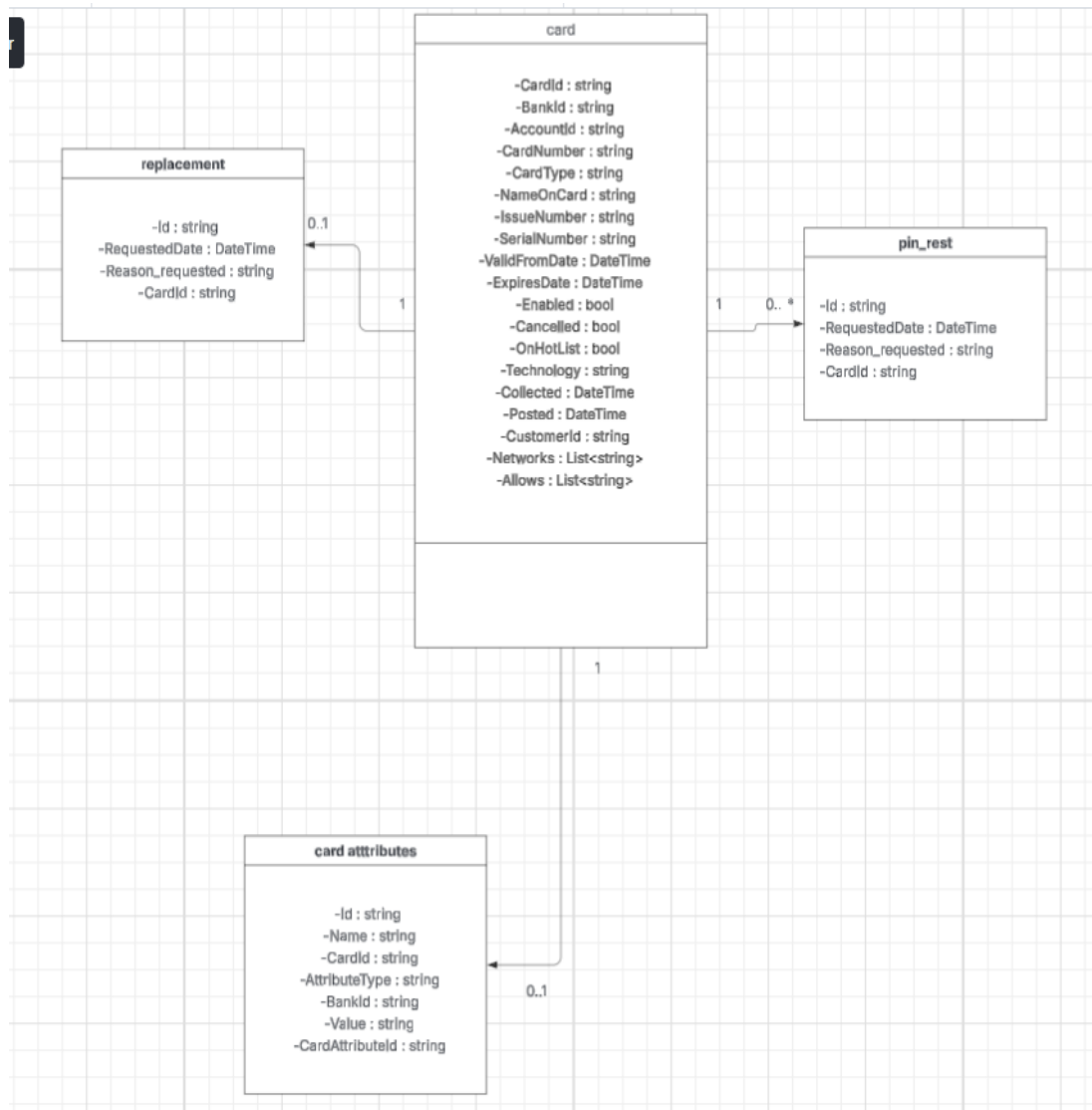


Figure 44: Diagramme de classe(Card Service)

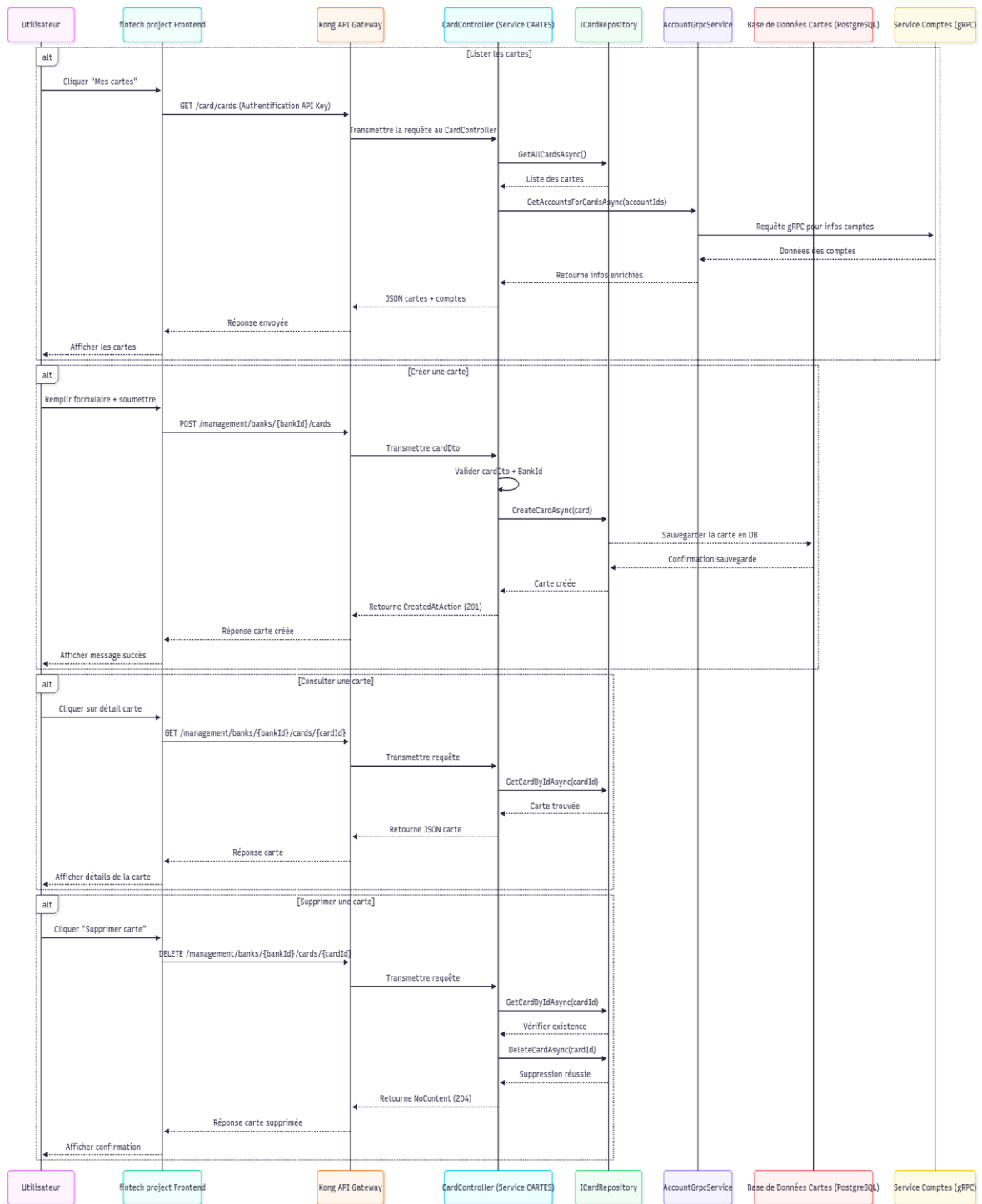


Figure 45: Diagramme de séquences(Card Service)

3.3.4. Revu du Sprint

La revue du sprint 2 a permis de valider la mise en place du microservice Cartes (Card-Service) au sein de la solution Banking-as-a-Service (BAAS). L'équipe a débuté par la conception du modèle de données, en définissant les entités principales telles que Card, Remplacement, Pin Reset et Card Attributes, puis en intégrant ces modèles dans une base PostgreSQL. Cette étape a été essentielle pour garantir la cohérence des informations bancaires et leur association correcte aux comptes utilisateurs, tout en respectant les exigences métier du domaine bancaire.

Le développement s'est poursuivi avec l'implémentation du repository Card Repository, assurant la gestion des opérations CRUD sur les cartes. L'équipe a également mis en place la communication inter-services via gRPC, permettant au Card-Service d'enrichir les données des cartes avec les informations des comptes provenant du Account-Service. Cette intégration a été testée et validée, garantissant la fiabilité des échanges et la cohérence des données entre microservices.

Côté exposition des fonctionnalités, le Card-Service a été publié derrière Kong API Gateway, avec la configuration du plugin key-auth pour sécuriser l'accès à l'API. Les endpoints REST (GET, POST, DELETE) ont été testés via Swagger et des clients externes, assurant la conformité aux spécifications et la robustesse du service face aux différents scénarios d'utilisation. Les tests QA ont porté une attention particulière à la gestion des erreurs, notamment celles liées à la communication gRPC et à l'authentification via l'API Gateway.

Enfin, l'intégration du Card-Service dans la pile d'observabilité a été réalisée avec Prometheus et Grafana, permettant un suivi en temps réel des métriques et de l'état du service. L'équipe a utilisé le Burndown Chart pour visualiser l'avancement du sprint et identifier les points d'amélioration. La démonstration finale a mis en avant la capacité du service à répondre aux besoins métier, à sa sécurité et à son intégration dans l'écosystème BAAS, validant ainsi tous les objectifs fixés pour ce sprint.

3.3.5. Démonstration

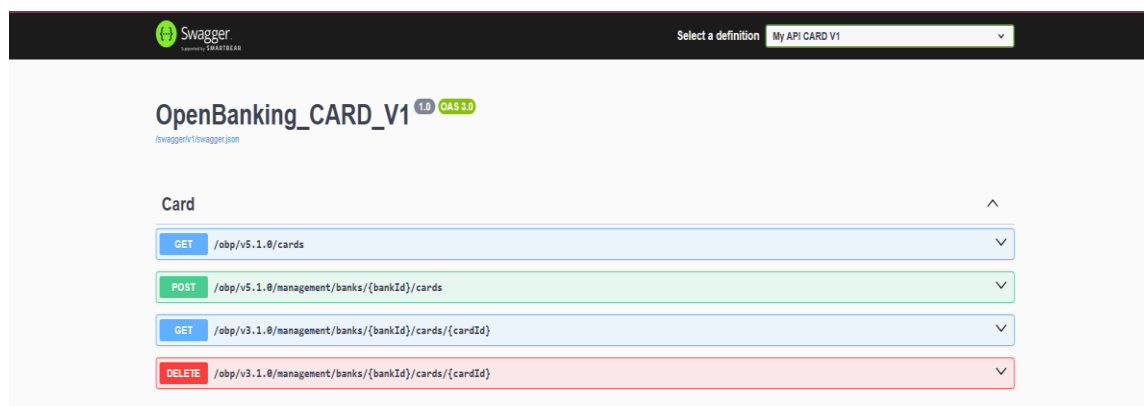


Figure 46: Documentation Api

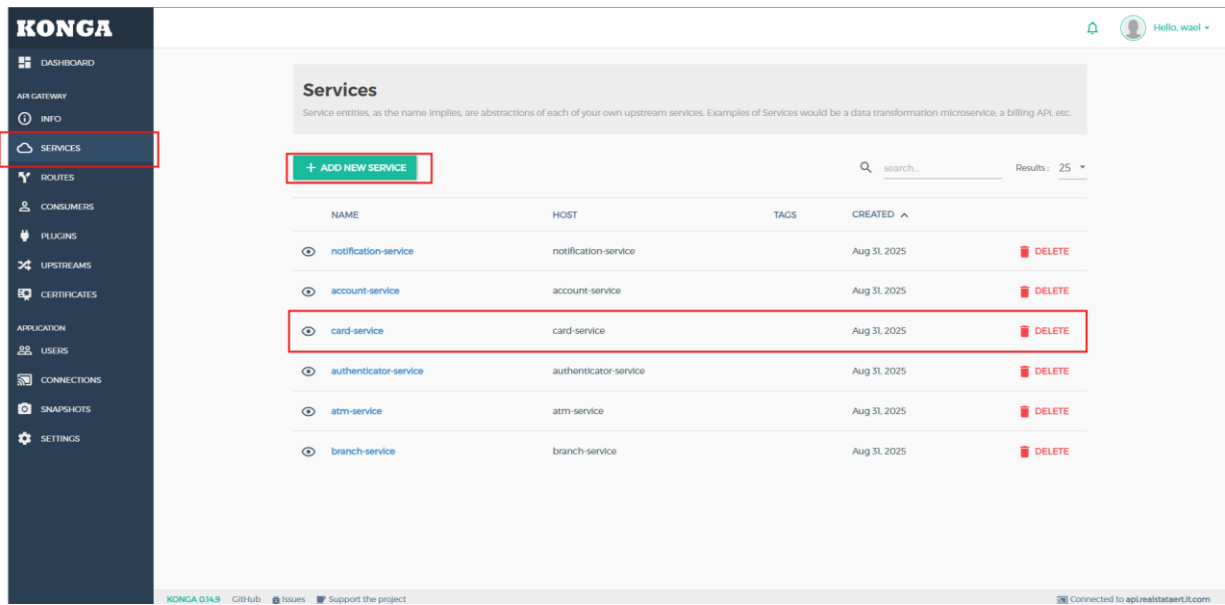


Figure 47: Intégration de Card-service avec Kong Gateway

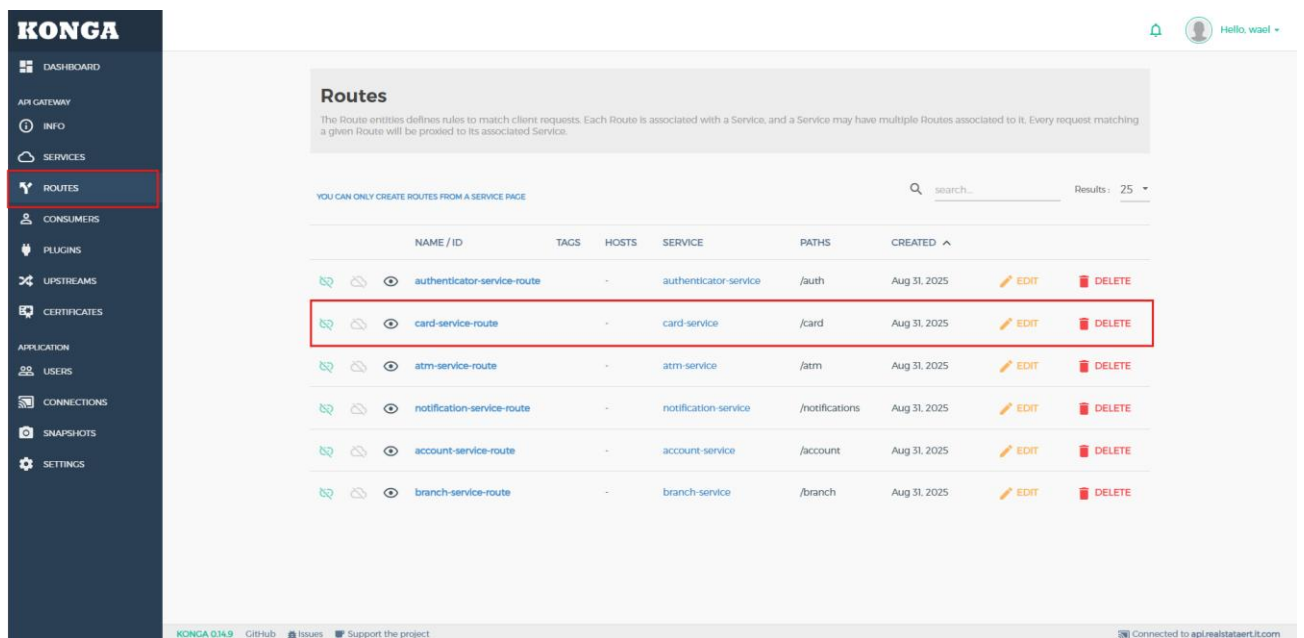


Figure 48: Route Card service proxisée avec kong

3.3.6. Burndown Chart

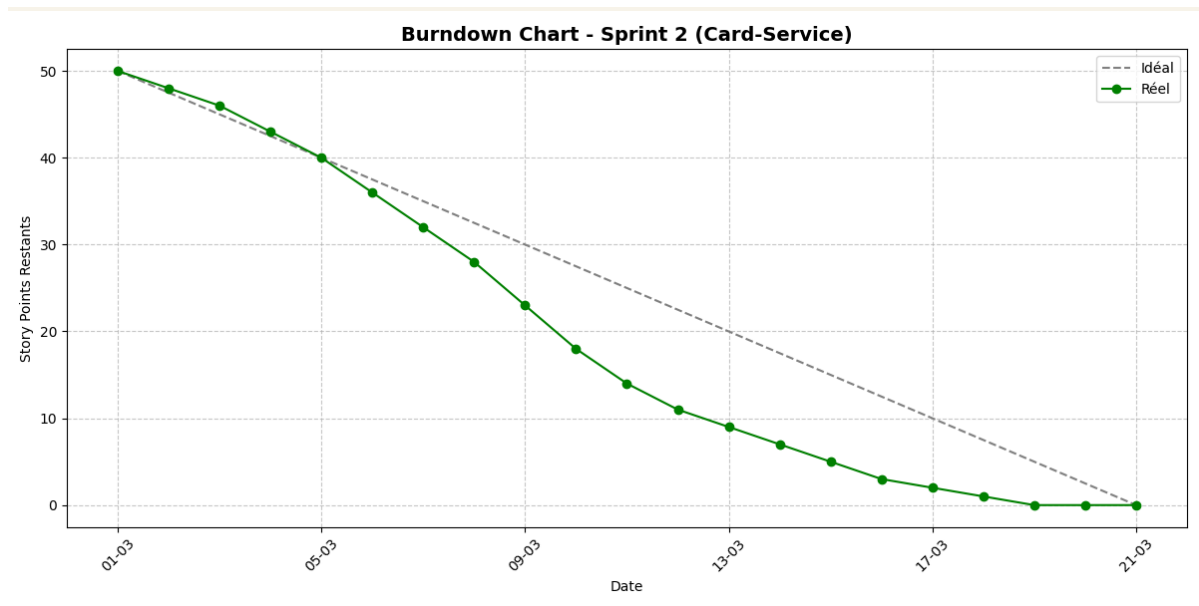


Figure 49: Burndown chart, Sprint 2

3.4. Sprint 3 : Microservice Guichets automatiques (ATM-Service)

3.4.1. Introduction :

Ce sprint est dédié à l'implémentation du microservice Guichets automatiques (ATM-Service) dans la solution Open Banking. Ce service gère les informations des distributeurs automatiques de billets (DAB), leur localisation, accessibilité, horaires, services offerts et caractéristiques techniques. L'ATM-Service expose des API REST conformes à la norme Open Banking, utilise MongoDB pour la persistance et publie des événements via RabbitMQ.

3.4.2. Backlog du sprint

Rôle	User Story	Tâche	Estimation
Développeur Backend	En tant qu'utilisateur, je veux consulter la liste des guichets automatiques d'une banque	Implémenter l'endpoint GET /banks/{bankId}/atms	8 pts
Développeur Backend	En tant qu'utilisateur, je veux consulter les détails d'un guichet automatique	Implémenter l'endpoint GET /banks/{bankId}/atms/{atmId}	5 pts
Développeur Backend	En tant qu'administrateur, je veux créer un nouveau guichet automatique	Implémenter l'endpoint POST /banks/{bankId}/atms	13 pts
Développeur Backend	En tant qu'administrateur, je veux ajouter des attributs à un guichet	Implémenter l'endpoint POST /banks/{bankId}/atms/{atmId}/attributes	8 pts
Développeur Backend	En tant que service, je veux publier un événement lors de la création d'un ATM	Intégrer RabbitMQ Publisher	8 pts
FinTech	En tant que FinTech, je veux accéder aux données ATM pour enrichir mes services	Implémenter des endpoints ouverts et documentés (Swagger)	5 pts
Testeur QA	En tant qu'utilisateur, je veux être informé en cas d'erreur	Tester la gestion des erreurs et la validation des entrées	5 pts
Scrum Master	En tant qu'équipe, je veux visualiser l'avancement et l'état du microservice	Suivi via Burndown Chart, monitoring	3 pts
Product Owner	En tant que PO, je veux vérifier que le service est exploitable via API	Démonstration avec Swagger + tests clients	5 pts

Tableau 5: Backlog Sprint 3

3.4.3. Conception du sprint

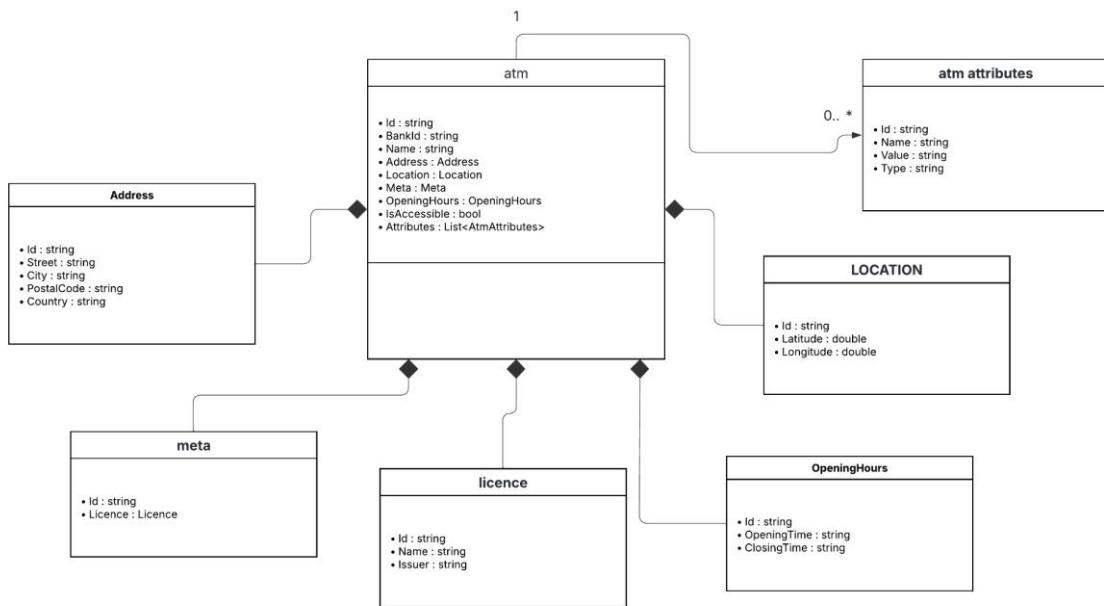


Figure 50: Diagramme de classe (ATM Service)

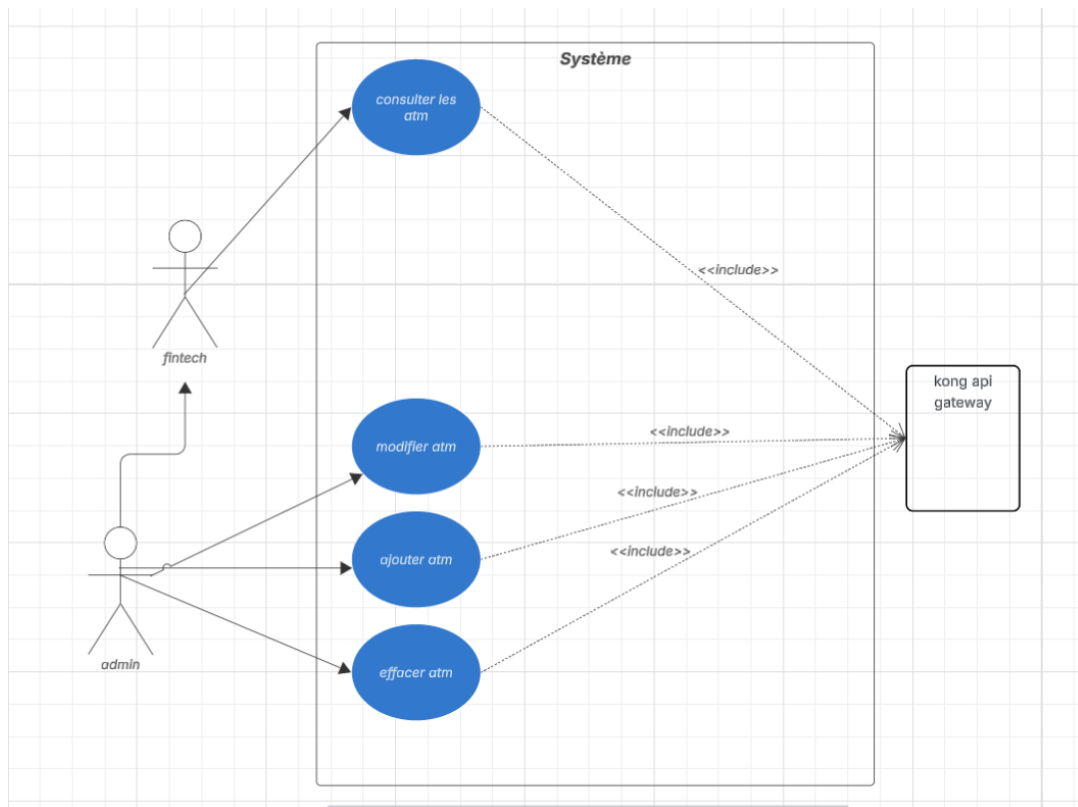


Figure 51: Diagramme de cas d'utilisation

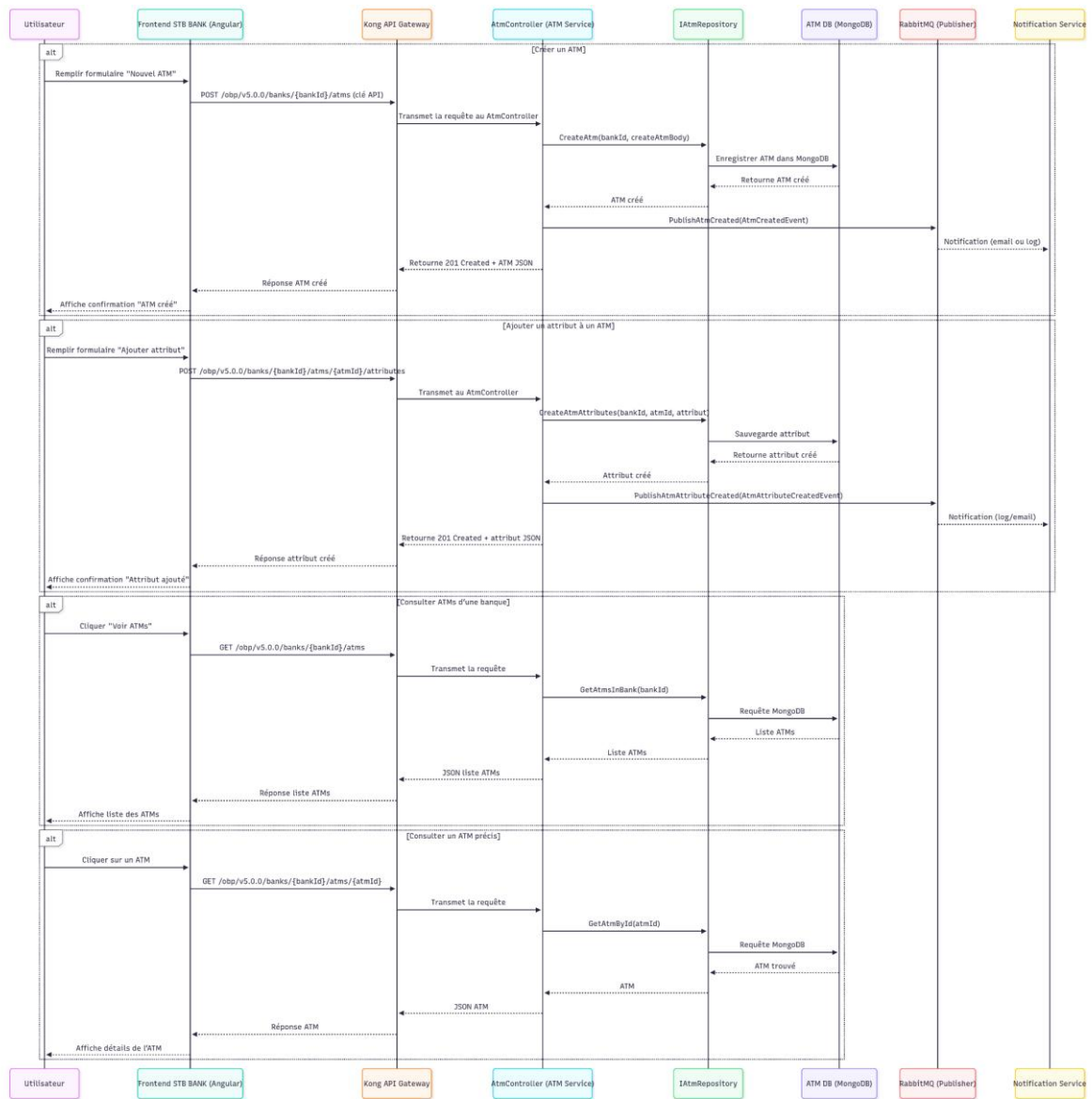


Figure 52: Diagramme de séquences (ATM Service)

3.4.4. Revu du Sprint

Ce sprint a permis de concevoir et de développer le microservice ATM-Service, essentiel pour la gestion des guichets automatiques dans l'écosystème Open Banking. L'équipe a commencé par la modélisation des entités principales (Atm, Address, Location, Meta, AtmAttributes, etc.), en veillant à représenter toutes les caractéristiques métier nécessaires. La persistance des données a été assurée via MongoDB, permettant une gestion flexible et évolutive des ATMs.

Les endpoints REST ont été implémentés conformément à la norme Open Banking, offrant aux utilisateurs et aux FinTechs la possibilité de consulter la liste des ATMs, leurs détails et d'ajouter de nouveaux guichets ou attributs. La gestion des erreurs et la validation des entrées ont été renforcées pour garantir la fiabilité du service, avec des retours explicites en cas de données manquantes ou invalides.

L'intégration de RabbitMQ a permis de publier des événements lors de la création d'un ATM, facilitant la communication inter-services et l'extension du système vers des usages avancés (notifications, analytics, etc.). La documentation Swagger a été enrichie pour permettre aux FinTechs et partenaires d'intégrer facilement le service dans leurs propres solutions.

Le suivi du sprint s'est appuyé sur le Burndown Chart, permettant à l'équipe de visualiser l'avancement, d'identifier rapidement les obstacles et d'ajuster le plan de travail. La démonstration finale a mis en avant la robustesse du service, sa conformité aux standards et son intégration réussie dans la plateforme Open Banking.

3.4.5. Démonstration

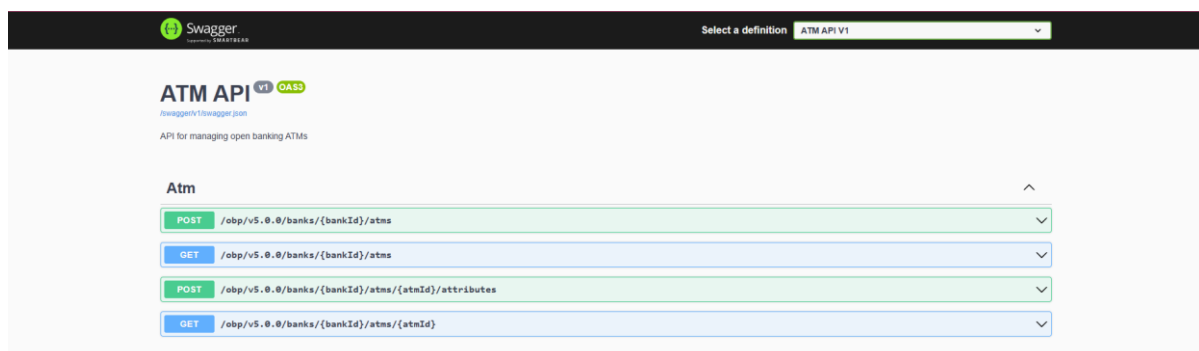


Figure 53: Documentation Api

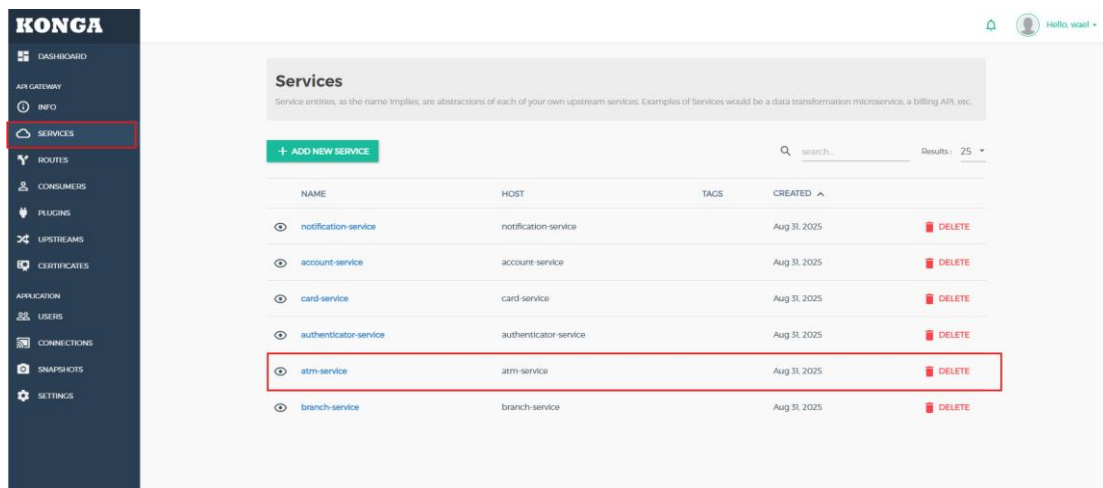


Figure 54: Intégration Atm-service avec Kong gateway

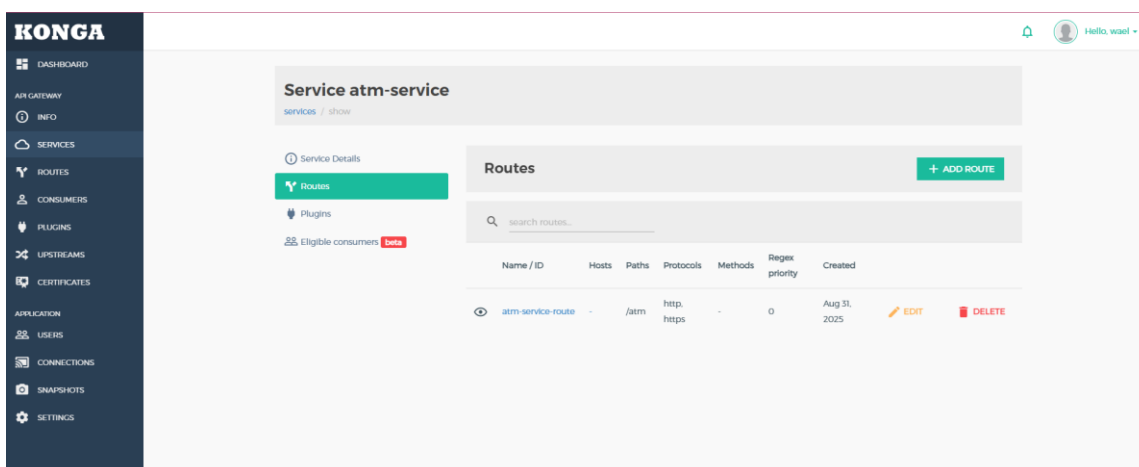


Figure 55: Route Atm-service proxisé avec kong

3.4.6. Burndown Chart

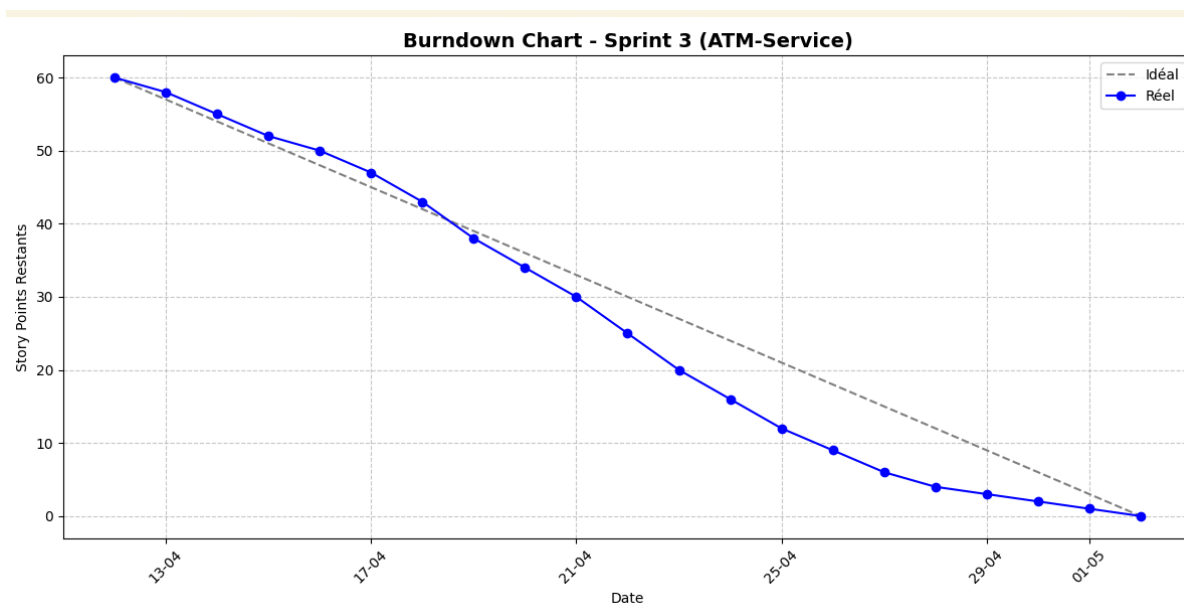


Figure 56: Burndown chart, Sprint 3

3.5. Sprint 4: Microservice Notifications

3.5.1. Introduction :

L'objectif de ce sprint était l'implémentation d'un microservice de notifications pour centraliser et diffuser les événements générés par les autres microservices du système Open Banking (par exemple, création de comptes, ajout d'attributs, mise en place d'ATM).

Le défi majeur consiste à fournir aux fintechs et à leurs équipes un système fiable permettant d'obtenir en direct des notifications utiles, plutôt que de se contenter de simples journaux techniques.

3.5.2. Backlog du sprint

Rôle	User Story	Tâche	Estimation
Fintech	En tant qu'utilisateur, je veux être notifié lorsqu'un compte est créé pour réagir rapidement.	- Consommer l'événement account.created . - Générer notification (log + email).	5 pts
Fintech	En tant qu'utilisateur, je veux être notifié lorsqu'un attribut est ajouté à un compte.	- Consommer account.attribute.created . - Publier une notification.	5 pts
Admin	En tant qu'admin, je veux que les notifications soient monitorées.	Exposer /metrics pour Prometheus. - Intégration possible de Grafana.	3 pts
développeur	En tant que développeur, je veux un mécanisme extensible de notification pour de futurs canaux.	- Implémenter la couche de service Notification.	7 pts
Product Owner	En tant que PO, je veux pouvoir démontrer la génération de notifications en conditions réelles.	- Préparer scénario de test (création d'un compte + attribut). - Logs + email de test.	3 pts

Tableau 6: Backlog Sprint 4

3.5.3. Revu du Sprint

À l'issue de ce sprint, le micro-service de notification a atteint un premier niveau fonctionnel. Les événements **comptables.created**, **account.attribute.created**, **atm.created** et **atm.attribute.created** sont correctement consommés, assurant la détection en temps réel des changements au sein du système.

Les notifications sont actuellement émises sous forme de **logs applicatifs** et peuvent également être envoyées par email via un serveur SMTP (par exemple Gmail, configuré dans *appsettings.json*). Ce mécanisme permet aux fintechs et collaborateurs de recevoir une information immédiate, bien que le format reste pour l'instant limité.

Un endpoint de monitoring a été exposé sur **/metrics**, rendant le service compatible avec **Prometheus** et **Grafana** pour le suivi en production.

La principale limitation identifiée réside dans l'absence de **persistance des notifications**. Celles-ci ne sont pas encore stockées en base de données et ne peuvent donc pas être historisées ni consommées par d'autres systèmes de manière fiable.

Ce sprint constitue ainsi une étape importante vers une plateforme de notification plus robuste. Les prochaines itérations visent à introduire un stockage persistant et une gestion multicanal (API, fichiers de messages, base de données), permettant une meilleure intégration avec les différents partenaires et applications clientes.

3.5.4. Démonstration

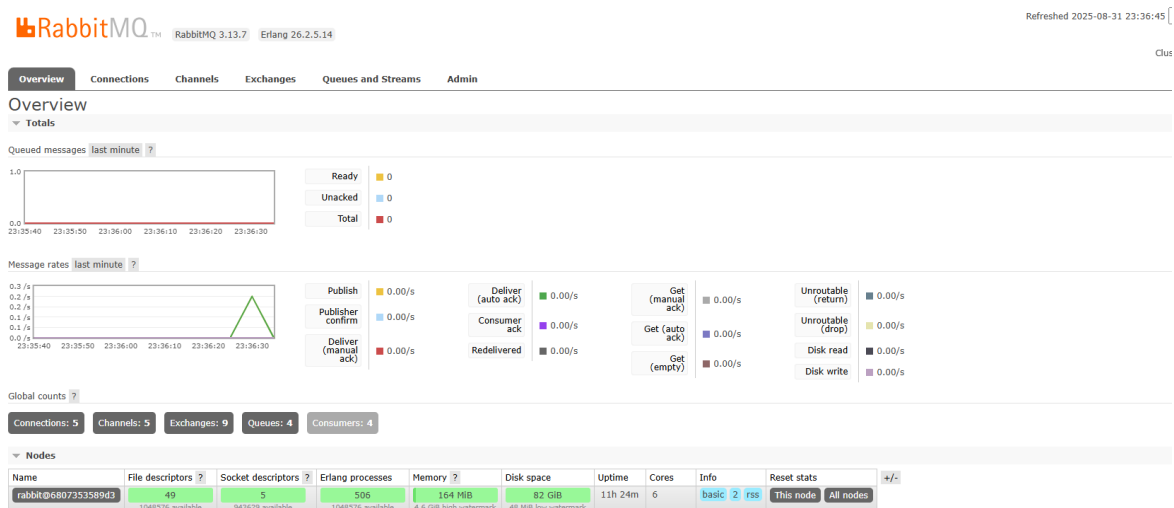


Figure 57: Interface Rabbitmq utilisé pour la communication asynchrone

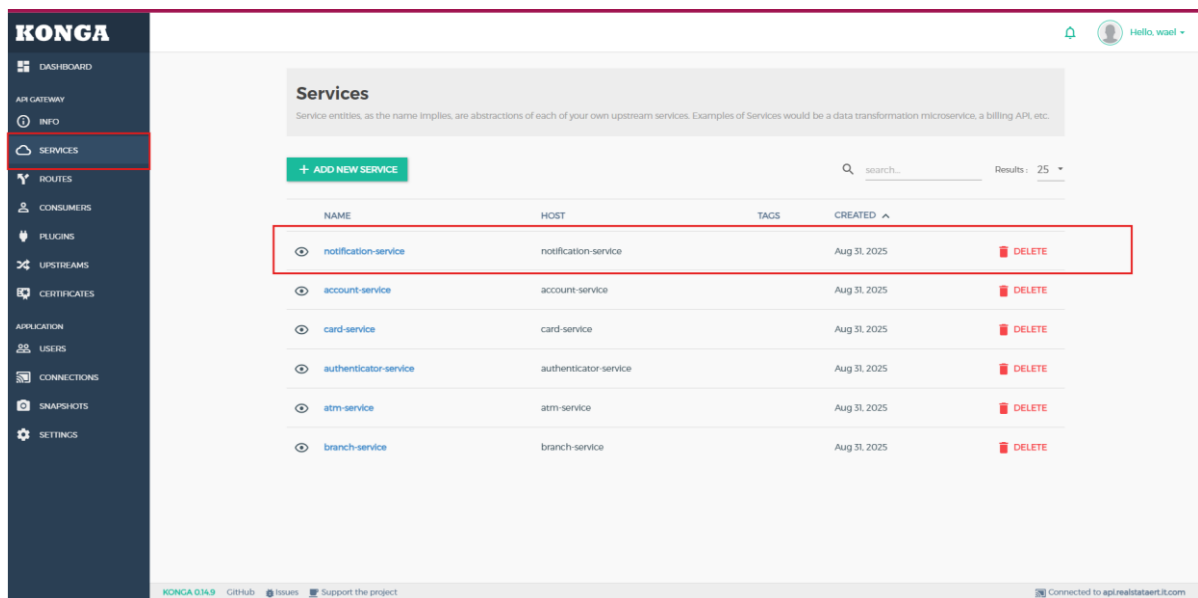


Figure 60: Intégration notification-service avec kong

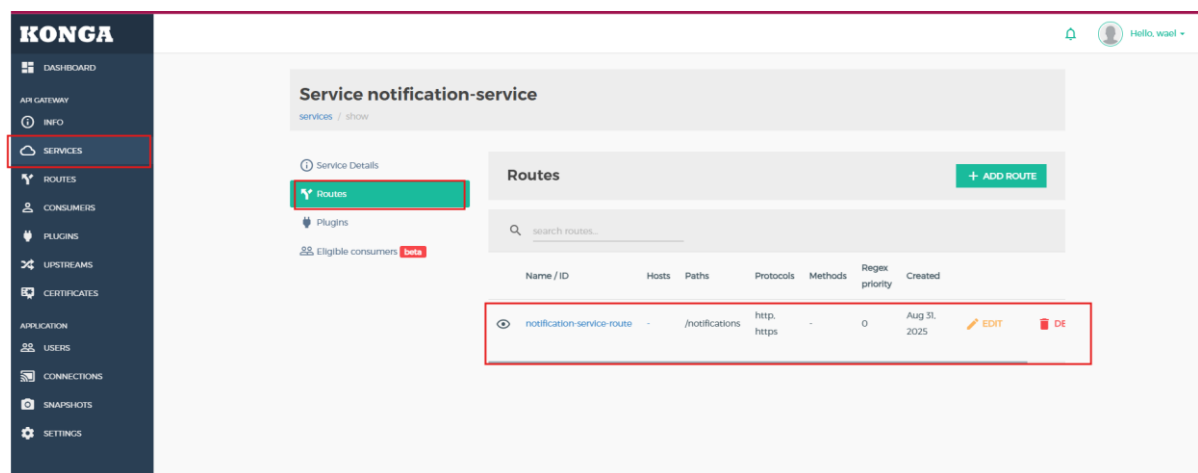


Figure 61: Route du service notification-service

3.5.5. Burndown Chart

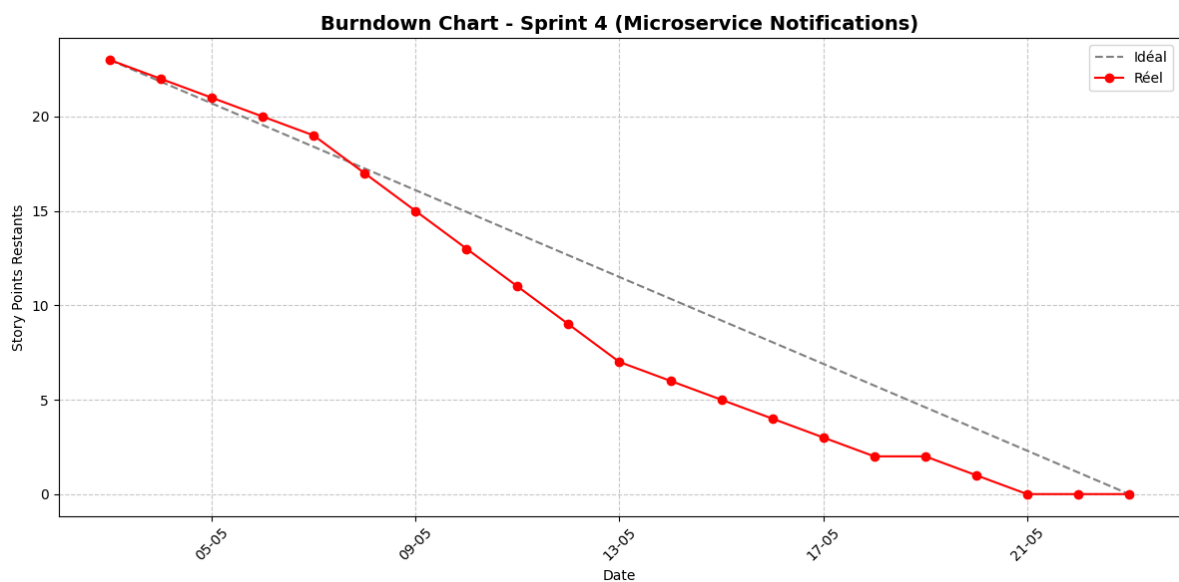


Figure 62: BurnDown Chart Sprint 4

3.6. Sprint 5: Communication interservices (RabbitMQ + gRPC)

3.6.1. Introduction :

Le but premier de ce sprint était d'établir une communication robuste, efficace et scalable entre les divers microservices de notre infrastructure Open Banking. Jusqu'à présent, chaque service interagissait directement par le biais d'appels REST, ce qui entraînait des dépendances importantes et restreignait la possibilité d'évoluer.

Ainsi, nous avons intégré RabbitMQ pour la communication basée sur les messages et les événements (asynchrone) et gRPC pour une communication synchrone à haute performance (RPC).

Cette association répond à deux besoins :

L'utilisation de RabbitMQ pour notifier de manière asynchrone les événements métiers (par exemple, création d'un compte, envoi d'alerte, mise à jour des cartes).

L'établissement d'une communication directe entre microservices se fait par le biais de gRPC, une méthode d'échange plus rapide, typée et efficace que l'HTTP.

3.6.2. Backlog du sprint

Rôle	User Story	Tâche	estimation
Développeur Backend	En tant que développeur, je veux que les microservices communiquent via RabbitMQ pour publier/consommer des événements	installer RabbitMQ via Docker, configurer les échanges et queues, implémenter un producteur et un consommateur.	10 pts
Développeur Backend	En tant que développeur, je veux utiliser gRPC pour les appels synchrone entre services critiques	Définir les fichiers .proto , générer les stubs, implémenter un service gRPC côté serveur et client.	10 pts
Architecte logiciel	En tant qu'architecte, je veux que la communication reste découplée et tolérante aux pannes	Définir les patterns d'intégration RabbitMQ et les schémas gRPC pour chaque cas.	5 pts
QA	En tant que testeur, je veux valider la résilience de la communication interservices	Écrire des tests unitaires et d'intégration pour vérifier la livraison des messages et les appels gRPC.	5 pts

Product Owner	En tant que PO, je veux démontrer aux parties prenantes la fluidité de la communication	Créer un scénario complet : une demande de création de compte génère un événement notifié par RabbitMQ, et un appel gRPC met à jour un autre service.	5 pts
---------------	---	---	-------

Tableau 7: Backlog Sprint 5

3.6.3. Conception du sprint

Au cours de ce sprint, la conception s'est centrée sur deux piliers majeurs en matière de communication interservices : RabbitMQ pour la communication asynchrone et gRPC pour la communication synchrone. Le but était d'établir une architecture de communication solide, souple et répondant aux exigences opérationnelles de notre plateforme bancaire.

3.6.3.1. Communication asynchrone avec RabbitMQ

La communication asynchrone permet aux microservices d'échanger des informations sans dépendre d'une réponse immédiate. Pour ce besoin, nous avons retenu **RabbitMQ** comme broker de messages.

3.6.3.1.1. Production d'événements :

Chaque microservice est capable d'émettre des événements métiers lorsqu'une action significative est réalisée.

Exemples :

- ❖ ***AccountCreatedEvent*** : émis lorsqu'un nouveau compte bancaire est créé.
- ❖ ***CardBlockedEvent*** : émis lorsqu'une carte bancaire est bloquée.

3.6.3.1.2. consommation des événements

D'autres microservices peuvent souscrire à ces événements et exécuter des actions en conséquence.

Exemple :

Le **Notification Service** consomme l'événement *account.Created* et envoyer automatiquement un email ou un SMS au client pour l'informer.

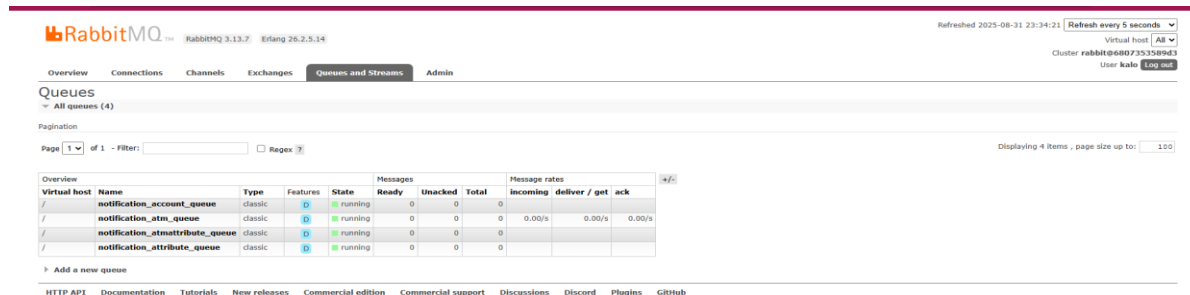
3.6.3.1.3. Mécanisme de routage :

Afin de gérer efficacement la distribution des messages, nous avons opté pour un **échange de**

type

topic.

Ce type d'échange offre une grande flexibilité puisqu'il permet de router les événements vers une ou plusieurs queues selon des patterns définis (exemple : notification account.* pour tous les événements liés aux comptes).



The screenshot shows the RabbitMQ web interface with the 'Queues and Streams' tab selected. It displays a table of queues for a specific virtual host. The table includes columns for queue name, type, state, and message counts. The queues listed are notification_account_queue, notification_atm_queue, notification_atmtribute_queue, and notification_attribute_queue.

Virtual host	Name	Type	Features	State	Ready	Unacked	Total	Message rates
								incoming deliver / get / ack
/	notification_account_queue	classic	D	running	0	0	0	
/	notification_atm_queue	classic	D	running	0	0	0	0.00/s 0.00/s 0.00/s
/	notification_atmtribute_queue	classic	D	running	0	0	0	
/	notification_attribute_queue	classic	D	running	0	0	0	

Figure 63:tableau de queues.

Avantages obtenus :

- Découplage fort entre les services producteurs et consommateurs.
- Mise en place d'une architecture réactive et extensible.
- Possibilité d'ajouter de nouveaux consommateurs sans impacter les producteurs existants.

3.6.3.2. Communication synchrone avec gRPC

Certains scénarios métier nécessitent une communication directe, rapide et fortement typée entre deux services. Dans ce contexte, nous avons intégré **gRPC** comme protocole de communication synchrone.

3.6.3.2.1. Cas d'utilisation :

Le **Account Service** doit interroger en temps réel le **Card Service** afin de valider une carte bancaire avant une opération.

3.6.3.2.2. Définition des contrats :

Tous les échanges sont définis dans des fichiers **.proto** qui décrivent les services, les méthodes disponibles ainsi que la structure des messages échangés. Ces fichiers permettent :

- ❖ Une **forte cohérence** entre le client et le serveur.
- ❖ Une **génération automatique** de stubs pour différents langages (Java, Go, Python, etc.).

3.6.3.2.3. Avantages

parmi les avantages de communication grpc :

- ❖ **performances élevées** grâce à l'utilisation de Protobuf (format binaire plus compact que JSON).

- ❖ **Streaming bidirectionnel** supporté, permettant des échanges continus entre client et serveur.
- ❖ **Communication fortement typée** et versionnée, minimisant les risques d'incompatibilité.
- ❖ **Sécurité renforcée** grâce au support natif de TLS.

3.6.4. Revu du Sprint

L'intégration de **RabbitMQ** et **gRPC** a transformé notre architecture en un système **plus modulaire, réactif et résilient**.

Les tests réalisés montrent que :

- ❖ Les microservices peuvent désormais **s'abonner dynamiquement à des événements** sans dépendances fortes. Lorsqu'un compte est créé, plusieurs services reçoivent l'événement en parallèle (notification, authentification, reporting).
- ❖ Le système supporte la **tolérance aux pannes** : si un consommateur RabbitMQ tombe, les messages restent dans la queue et sont traités dès sa remise en ligne.
- ❖ Les appels gRPC offrent une **latence très faible**, en moyenne 30% plus rapide que REST classique.
- ❖ Le couplage fort entre microservices a été réduit : un service peut évoluer sans impacter directement les autres.

Ce sprint a permis de poser les bases d'une **communication interservices robuste**, ouvrant la voie à une montée en charge sans dégradation des performances.

3.6.5. Démonstration

3.6.5.1. Démonstration pour rabbitmq

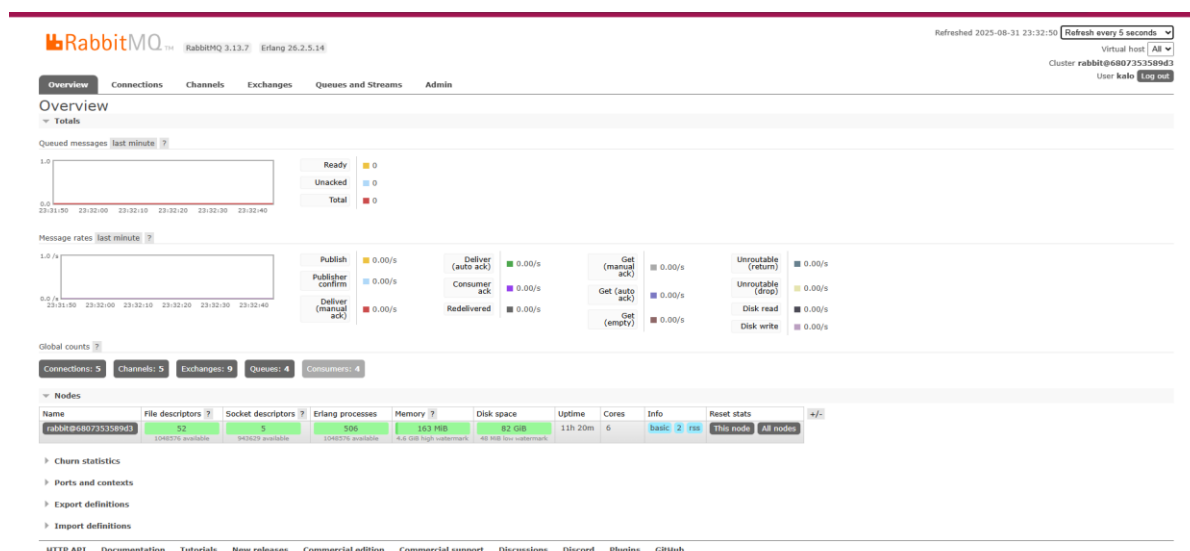


Figure 64: Démonstration de l'interface RabbitMQ

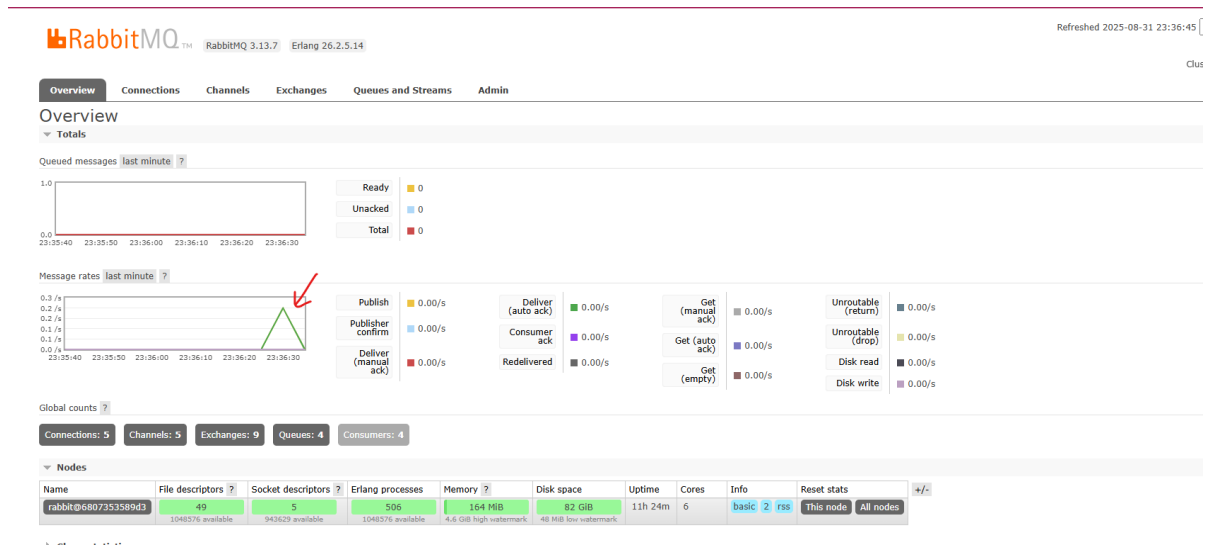


Figure 65 : Événement capturé

```

debian@vps-3020b30b:~/projet/Banking_Solution_PFE_2025$ docker logs -f notification-service
info: Microsoft.Hosting.Lifetime[14]
  Now listening on: http://[::]:8080
info: Microsoft.Hosting.Lifetime[0]
  Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
  Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
  Content root path: /app
=== [LOG] Atm Attribute Created Notification ===
To: System
Subject: New Atm Created
Body:
  ✓ A new atm attribute was added:
  ID: d7ed7483-9ccf-45f0-959c-9bfdc26d62aa
  Name: bank-001
  Value: string
=====
=== [LOG] Atm Attribute Created Notification ===
To: System
Subject: New Atm Created
Body:
  ✓ A new atm attribute was added:
  ID: b81881ad-6a5c-445e-8a8d-87d3087d9b79
  Name: bank-001
  Value: zitouna
=====
=== [LOG] Atm Attribute Created Notification ===
To: System
Subject: New Atm Created
Body:
  ✓ A new atm attribute was added:
  ID: f5008be4-bbcd-455b-a198-1f87be1de440
  Name: bank-001
  Value: POSTE
=====

```

Figure 66: Consommation des événements par le service de notification

3.6.6. Burndown Chart

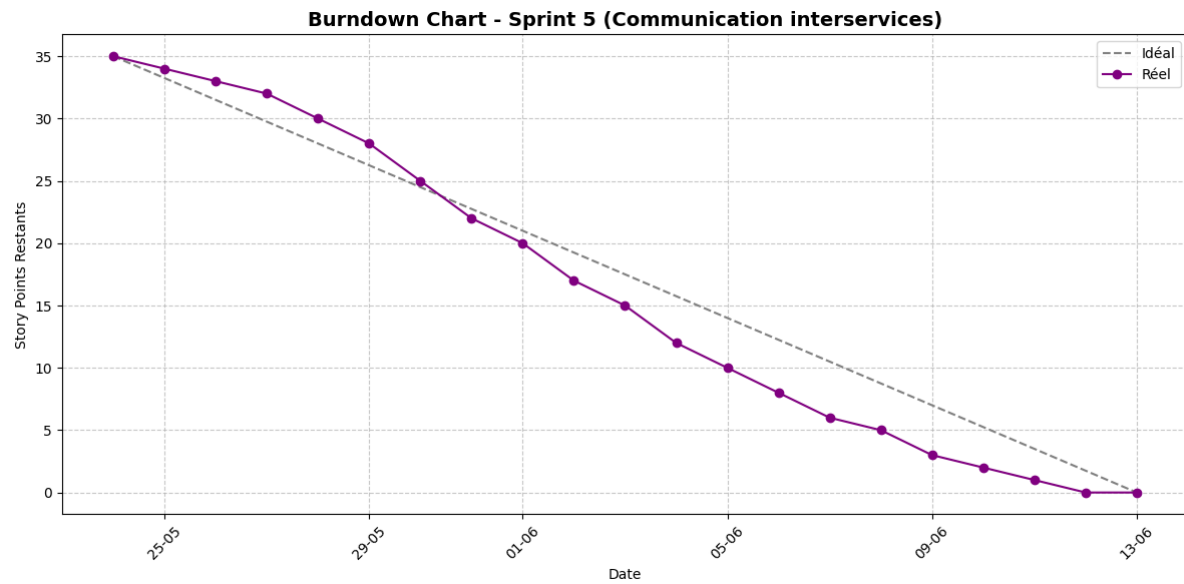


Figure 67: Burndown chart, Sprint 5

3.7. Sprint 6 : API Gateway Kong (sécurité, rate limiting, authentication)

3.7.1. Introduction :

L'objectif de ce sprint est de mettre en place **Kong API Gateway** pour centraliser et sécuriser l'accès aux microservices de notre architecture Open Banking. Kong permettra d'appliquer des politiques de sécurité (authentification, autorisation, JWT), de contrôler le **rate limiting** (protection contre abus/excès de requêtes) et de faciliter l'exposition unifiée des API à travers un seul point d'entrée.

Microservices Concernés :

account-service	→	/account
atm-service	→	/atm
branch-service	→	/branch
card-service	→	/card
authenticator-service	→	/auth
notification-service	→ /notifications	

3.7.2. Backlog du sprint

Role	User Story	Tâches	Estimation
Admin	En tant qu'administrateur, je veux configurer Kong pour centraliser les API	- Déployer Kong et Konga (interface UI) - Configurer les services et routes - Vérifier accessibilité	8 pts
Développeur	En tant que développeur, je veux sécuriser les microservices via JWT	- Activer le plugin JWT sur Kong - Configurer clés publiques/privées - Tester token d'accès	8 pts
Devops	En tant qu'DevOps, je veux limiter les requêtes pour éviter la surcharge	- Mettre en place Rate Limiting (par IP, par consommateur) - Tester avec scénarios de charge	5 pts
Fintech	En tant que consommateur d'API, je veux utiliser une seule URL sécurisée	- Configurer un upstream unique - Documenter les endpoints publics unifiés	4 pts

Tableau 8: Backlog Sprint 6

3.7.3. Conception du sprint

Architecture réseau : tous les microservices exposés via Kong (<http://kong-gateway/api/...>).

Sécurité :

- JWT obligatoire pour l'authentification.
- API Keys créées pour les fintechs (consumers).

Rate Limiting : quotas appliqués par clé d'API pour protéger les services sensibles.

Automatisation : script Bash pour enregistrer automatiquement les services et routes auprès de Kong Admin API.

Observabilité : intégration de logs et métriques pour suivi du trafic.

3.7.4. Revu du Sprint

L'intégration de **Kong API Gateway** dans notre architecture a marqué une étape essentielle vers la sécurisation et l'optimisation des échanges entre les microservices. Durant ce sprint, plusieurs objectifs majeurs ont été atteints.

Tout d'abord, l'ensemble des microservices a été regroupé derrière un point d'entrée unique via l'API Gateway. Cette centralisation a permis de simplifier la consommation des services et d'assurer une cohérence dans la gestion des accès. Les tests ont confirmé que chaque service est accessible à travers l'URL standardisée <https://api.realstataert.it.com/...> , réduisant ainsi la complexité côté client.

Sur le plan de la sécurité, l'authentification basée sur **JWT** a été mise en place avec succès. Cette implémentation garantit que seuls les utilisateurs disposant d'un jeton valide peuvent interagir avec les API. Cela renforce la protection des données et prévient tout accès non autorisé.

De plus, la configuration du **rate limiting** a permis de prévenir efficacement les risques de surcharge des microservices. Les requêtes excédentaires sont désormais bloquées, ce qui assure la stabilité et la disponibilité des services, même en cas de forte sollicitation.

Enfin, cette intégration a eu un impact direct sur l'expérience des utilisateurs finaux, notamment les fintechs et les collaborateurs. Grâce à une documentation centralisée et une gestion unifiée des accès, l'utilisation des services est devenue plus fluide et plus fiable.

En conclusion, ce sprint a permis non seulement de renforcer la sécurité et la résilience du système, mais également d'apporter une meilleure gouvernance technique sur l'ensemble des microservices

3.7.5. Démonstration

3.7.5.1. Dashboard konga

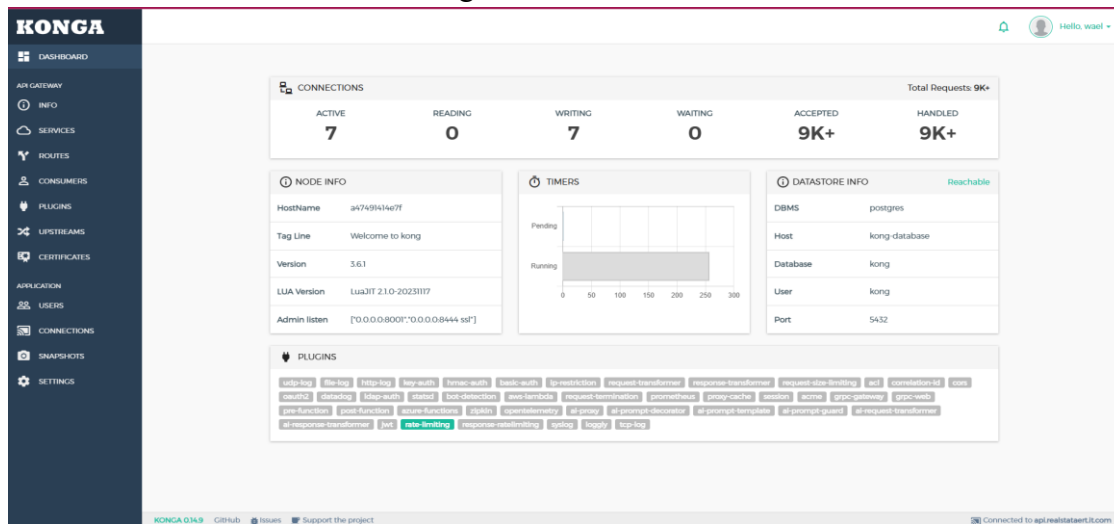


Figure 68: Dashboard konga

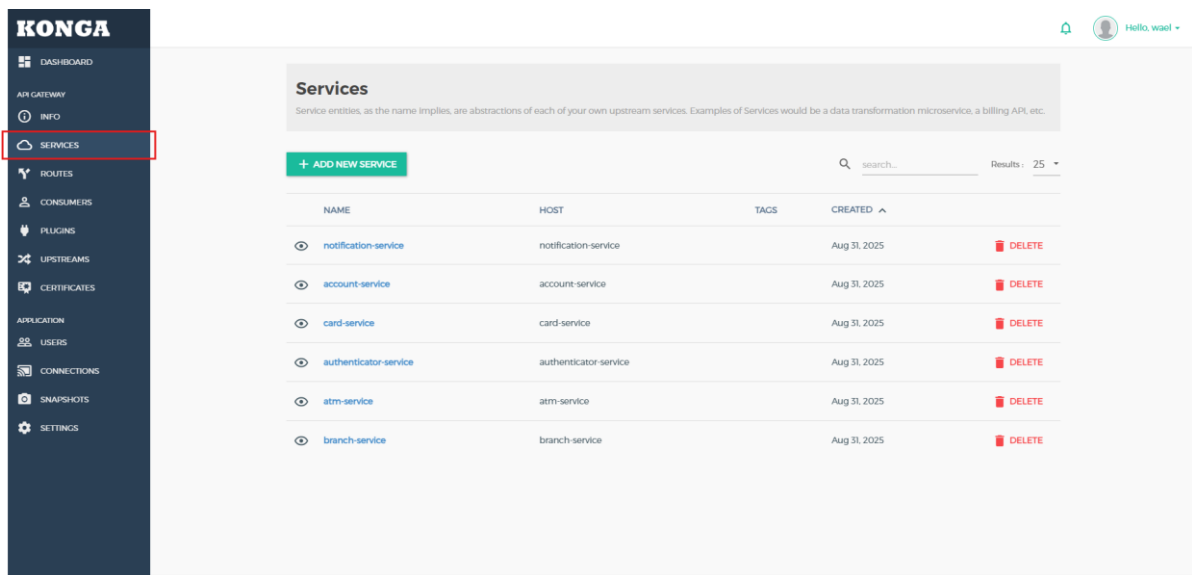


Figure 69 : intégration des microservices avec kong

KONGA

DASHBOARD

API GATEWAY

INFO

SERVICES

ROUTES

CONSUMERS

PLUGINS

UPSTREAMS

CERTIFICATES

APPLICATION

USERS

CONNECTIONS

SNAPSHOTS

SETTINGS

Routes

The Route entity defines rules to match client requests. Each Route is associated with a Service, and a Service may have multiple Routes associated to it. Every request matching a given Route will be proxied to its associated Service.

YOU CAN ONLY CREATE ROUTES FROM A SERVICE PAGE

search... Results: 25

NAME / ID	TAGS	HOSTS	SERVICE	PATHS	CREATED		
authenticator-service-route	-	-	authenticator-service	/auth	Aug 31, 2025	EDIT	DELETE
card-service-route	-	-	card-service	/card	Aug 31, 2025	EDIT	DELETE
atm-service-route	-	-	atm-service	/atm	Aug 31, 2025	EDIT	DELETE
notification-service-route	-	-	notification-service	/notifications	Aug 31, 2025	EDIT	DELETE
account-service-route	-	-	account-service	/account	Aug 31, 2025	EDIT	DELETE
branch-service-route	-	-	branch-service	/branch	Aug 31, 2025	EDIT	DELETE

KONGA 0.14.9 GitHub Issues Support the project

Connected to api.realdataent.it.com

Figure 70: Route de chaque microservice

KONGA

DASHBOARD

API GATEWAY

INFO

SERVICES

ROUTES

CONSUMERS

PLUGINS

UPSTREAMS

CERTIFICATES

APPLICATION

USERS

CONNECTIONS

SNAPSHOTS

SETTINGS

Consumers

The Consumer object represents a consumer - or a user - of an API. You can either rely on Kong as the primary datastore, or you can map the consumer list with your database to keep consistency between Kong and your existing primary database.

+ CREATE CONSUMER

search... Results: 25

USERNAME	CUSTOM_ID	TAGS	CREATED	
user-3	-	-	Aug 31 2025 @22:01	DELETE
user-2	-	-	Aug 31 2025 @17:55	DELETE
user-1	-	-	Aug 31 2025 @17:20	DELETE

KONGA 0.14.9 GitHub Issues Support the project

Connected to api.realdataent.it.com

Figure 71: Liste des consommateurs adéquats

KONGA

DASHBOARD

API GATEWAY

INFO

SERVICES

ROUTES

CONSUMERS

PLUGINS

UPSTREAMS

CERTIFICATES

APPLICATION

USERS

CONNECTIONS

SNAPSHOTS

SETTINGS

Plugins

A Plugin entity represents a plugin configuration that will be executed during the HTTP request/response workflow, and it's how you can add functionalities to APIs that run behind Kong, like Authentication or Rate Limiting for example.

+ ADD GLOBAL PLUGINS

search... Results: 25

NAME	SCOPE	APPLY TO	CONSUMER	CREATED	
rate-limiting	global	All Entrypoints	9eb70e96-27bf-4ea6-8e97-72a90b415174	Aug 31, 2025	DELETE
rate-limiting	global	All Entrypoints	83a41773-7688-41be-8b71-a6e4a759b918	Aug 31, 2025	DELETE
rate-limiting	global	All Entrypoints	cc7a1bdc-d60c-439f-9ce9-38996658aebd	Aug 31, 2025	DELETE

KONGA 0.14.9 GitHub Issues Support the project

Connected to api.realdataent.it.com

Figure 72: les plugins utilisés

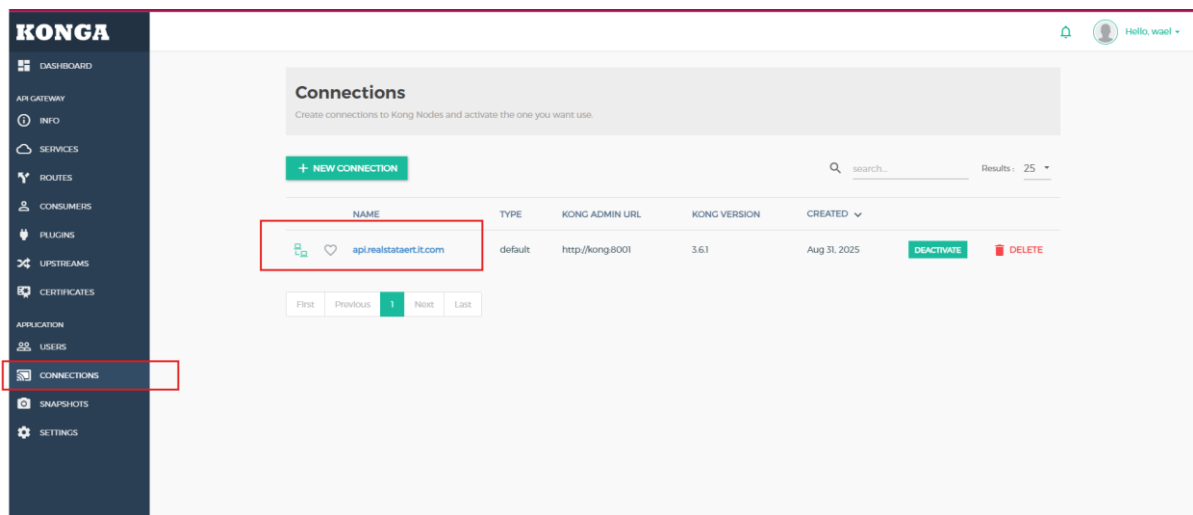


Figure 73: Connection Kong avec Konga

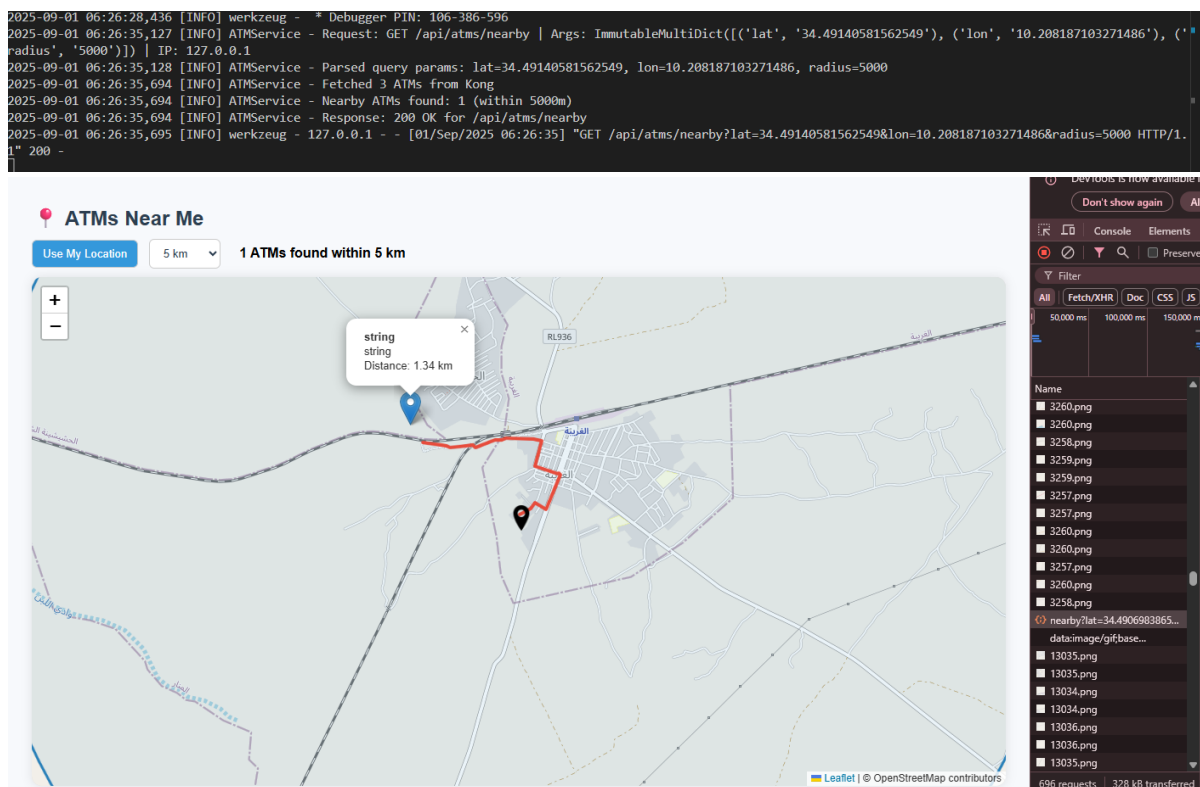


Figure 74: Consommation de l'Api ATM a partir de kong

3.7.6. Burndown Chart

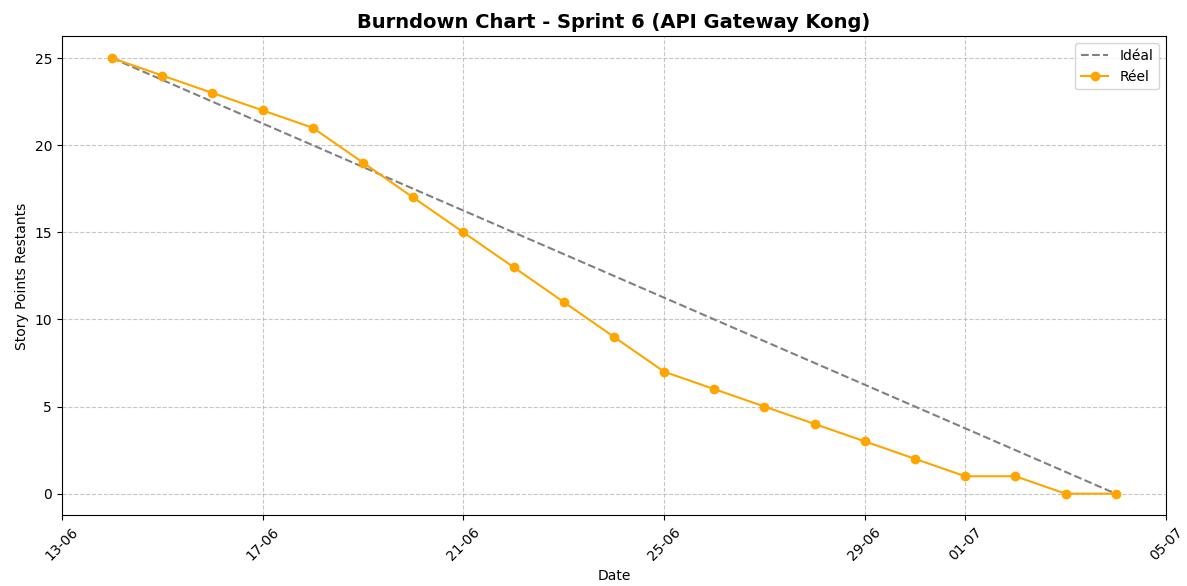


Figure 75: Burndown chart, Sprint 6

3.8. Sprint 7: Microservice Authentication (JWT, OAuth, SSL)

3.8.1. Introduction :

Le Sprint 8 a été consacré à la mise en place du **micro-service d'authentification** permettant de sécuriser l'accès à l'application Open Banking.
L'objectif principal était de développer un système robuste basé sur **JWT** (JSON Web Token), en assurant la confidentialité des échanges via **SSL**.
Ce sprint a posé les bases de la gestion centralisée des utilisateurs, des rôles et de la distribution des clés API associées aux différents plans business.

3.8.2. Backlog du sprint

Rôle	User Story	Tâche	Estimation
Utilisateur	En tant qu'utilisateur, je veux créer un compte pour accéder à la plateforme	Implémentation de l'endpoint register (backend + frontend)	5 pts
Utilisateur	En tant qu'utilisateur, je veux me connecter avec email/mot de passe.	Implémentation de l'endpoint login (backend + frontend)	5 pts
Utilisateur	En tant qu'utilisateur connecté, je veux visualiser mes informations.	Endpoint /user protégé avec JWT	4 pts
Utilisateur	En tant qu'utilisateur, je veux renouveler ma clé API quand elle expire.	Endpoint /api-key/renew	3 pts
Utilisateur	En tant qu'utilisateur, je veux souscrire à un plan business (BASIC, PRO, SUPER, ELITE).	Endpoint /assign-plan et intégration avec Kong Gateway	5 pts
Administrateur	En tant qu'admin, je veux sécuriser les échanges avec SSL/TLS.	Installation et configuration de Certbot (Let's Encrypt)	3 pts

Tableau 9: Backlog Sprint 7

3.8.3. Conception du sprint

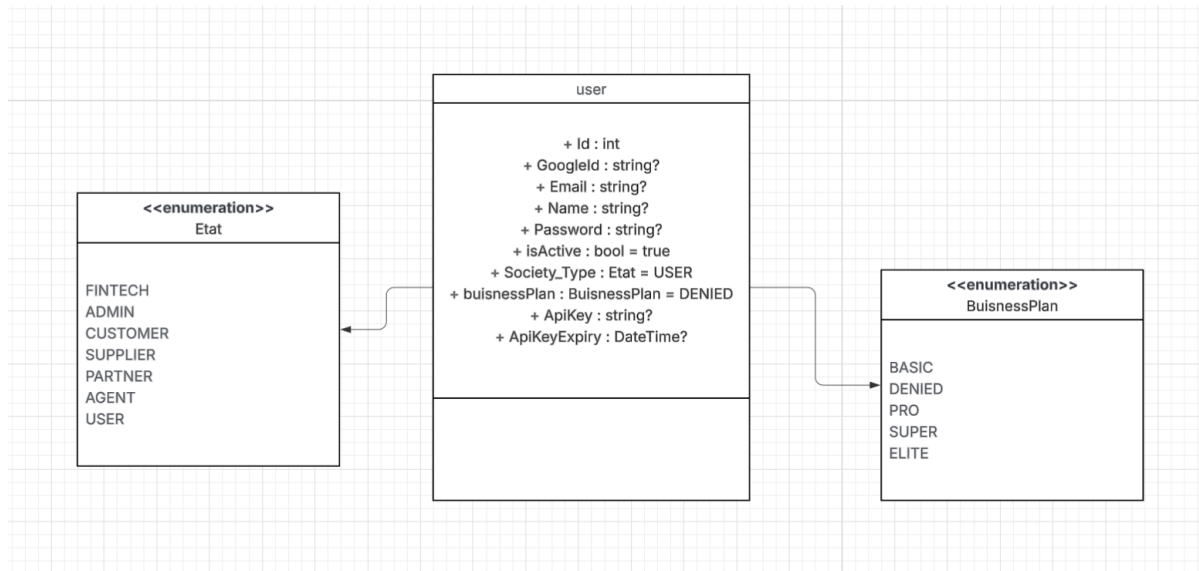


Figure 76 :Diagramme de classe (Authenticator Service)

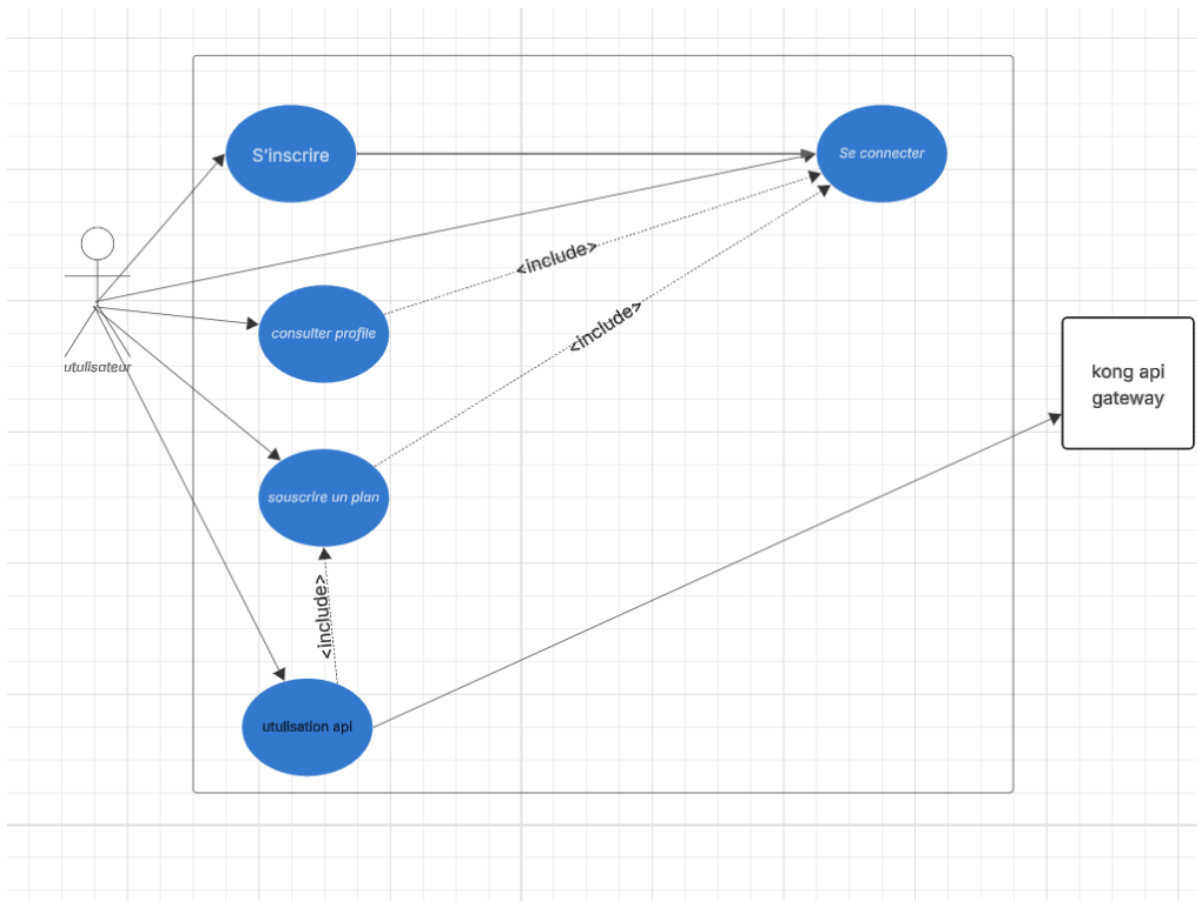


Figure 77:Diagramme de cas d'utilisation (Authenticator Service)

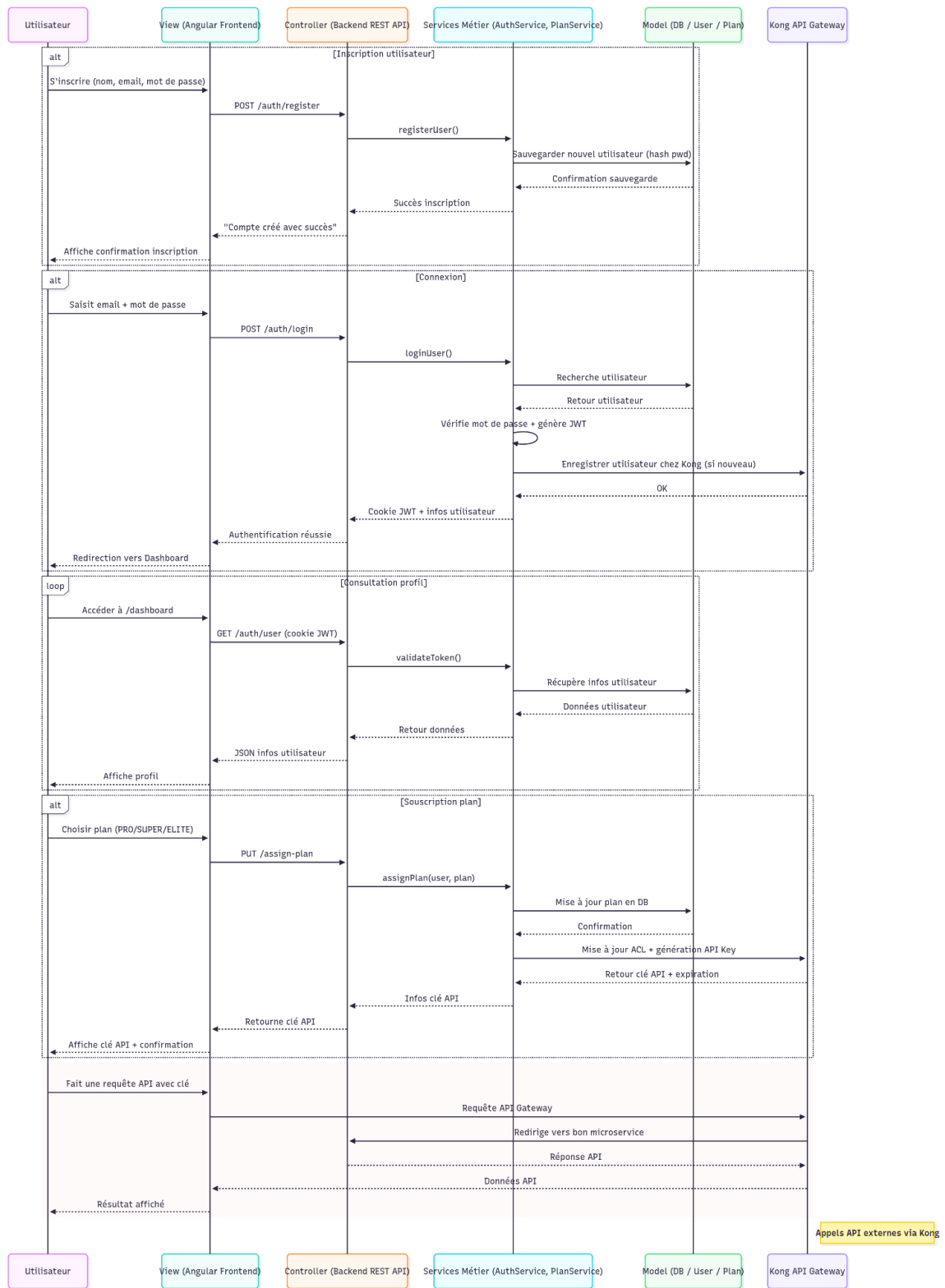


Figure 78: Diagramme de séquence (Authenticator Service)

3.8.4. Revu du Sprint

À l'issue du Sprint 8, les fonctionnalités principales du microservice d'authentification ont été implémentées et validées.

Les utilisateurs peuvent désormais **s'inscrire, se connecter et gérer leurs informations personnelles** via une API sécurisée. Les tokens JWT sont stockés dans des cookies sécurisés (HttpOnly, Secure, SameSite=None), garantissant une protection contre les attaques XSS.

L'intégration avec **Kong API Gateway** a permis d'automatiser la gestion des **clés API et ACL** selon le plan business choisi (BASIC, PRO, SUPER, ELITE). De plus, l'authentification via **Google OAuth2** a été mise en place, permettant un login alternatif simplifié.

Côté frontend, un module Angular a été développé pour gérer l'inscription, la connexion et la souscription aux plans business. L'expérience utilisateur a été renforcée avec la détection automatique de session via le cookie JWT.

Malgré quelques difficultés rencontrées lors de la synchronisation avec Kong (gestion des duplications de clés et erreurs 409), le système a pu être stabilisé avec des contrôles de robustesse.

Ce sprint marque donc une avancée importante dans la sécurisation et la professionnalisation de la plateforme Open Banking.

3.8.5. Démonstration

3.8.5.1. Démonstration Api

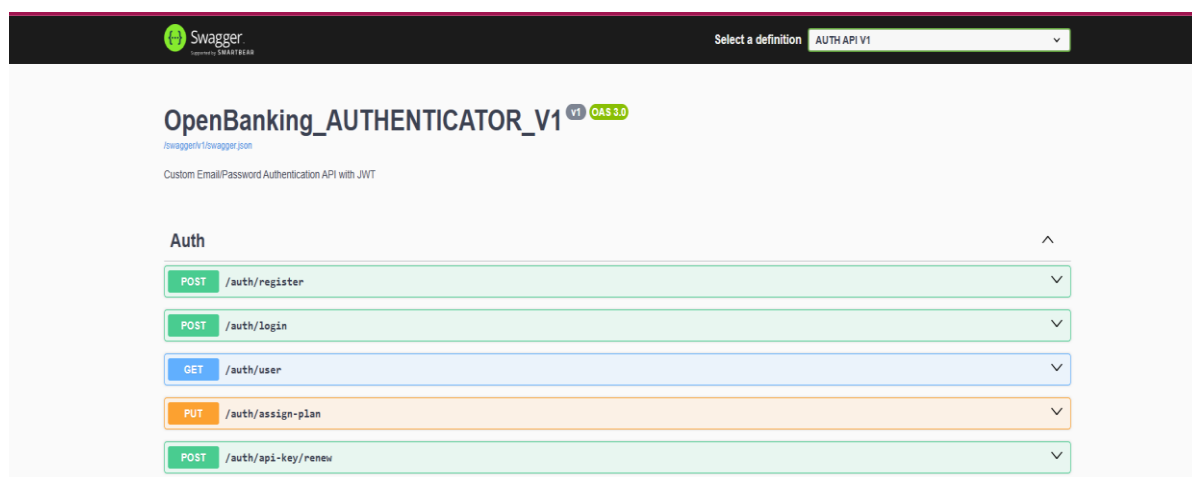


Figure 79: Documentation Api

3.8.5.2. Intégration Authentication service avec kong

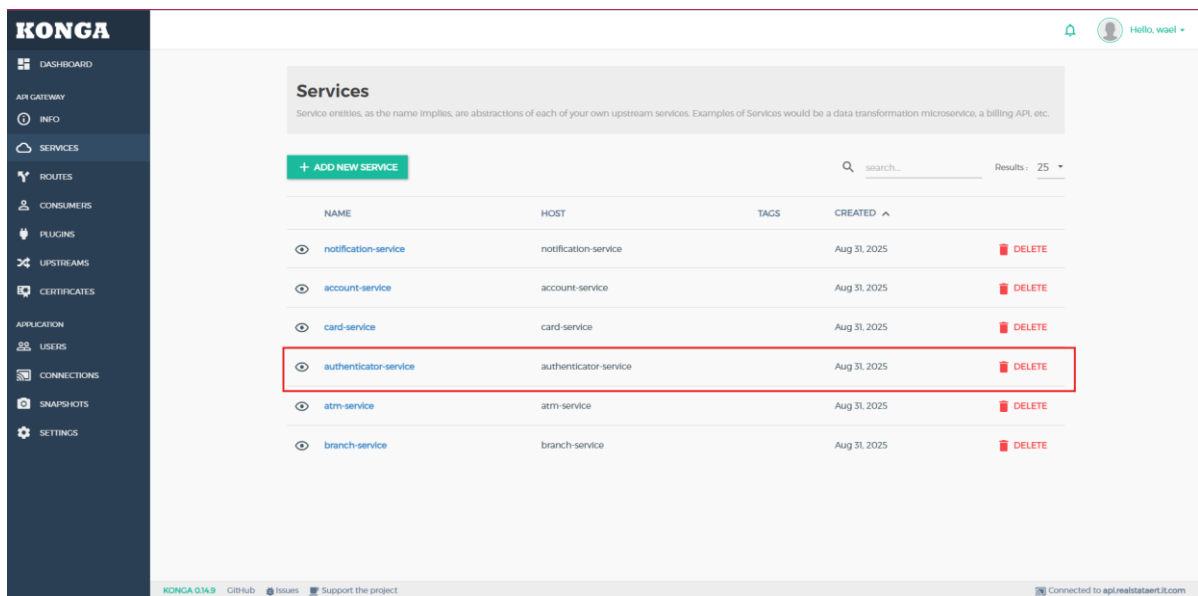


Figure 80: Intégration Authentication service avec kong

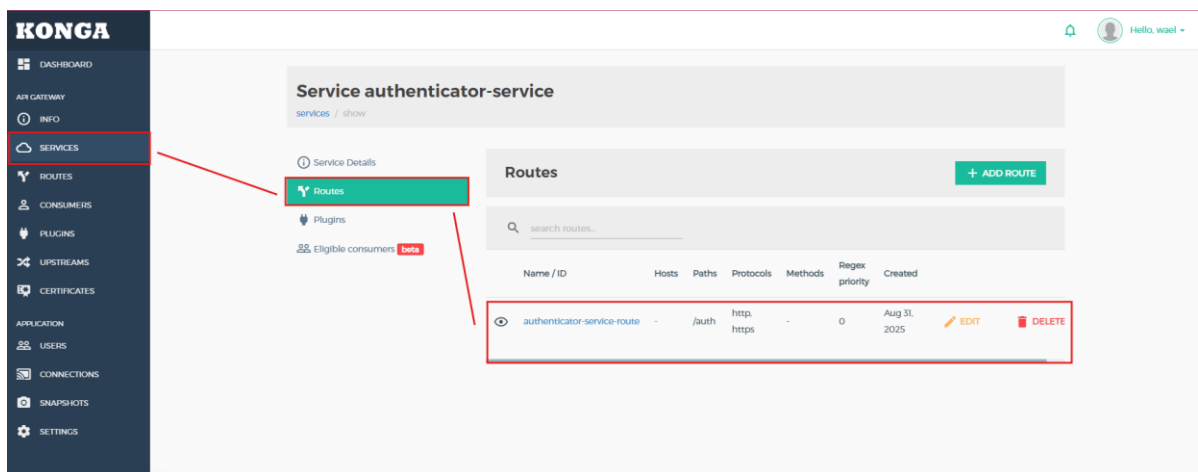


Figure 81 : Déclaration de la route à accéder ce service a partir de kong

3.8.5.3. Démonstration en vue client

The screenshot shows a web browser window with the URL `realdataert.it.com/authentication`. The page has a purple-to-pink gradient background. In the center is a white card titled "Create Account" with the subtitle "Join us and start your journey". The card contains the following fields: "Name" (with "John Doe" entered), "Email Address" (with "john@example.com" entered), and "Password" (with "Enter your password" as a placeholder). Below these fields is a checkbox labeled "I agree to the terms". At the bottom of the card is a blue "Create Account" button and a link that says "Already have an account? Sign in".

Figure 82: Interface Création Compte pour les Fintechs

The screenshot shows a web browser window with the URL `realdataert.it.com/authentication`. The page has a purple-to-pink gradient background. In the center is a white card titled "Welcome Back" with the subtitle "Sign in to your account". The card contains the following fields: "Email Address" (with "fintech@gmail.com" entered) and "Password" (with "*****" as a placeholder). Below these fields is a blue "Sign In" button. At the bottom of the card is a link that says "Don't have an account? Create one".

Figure 83: Interface login pour les Fintechs

The screenshot shows a web application dashboard for "STB BANK". On the left is a dark sidebar with a "Logout" button at the bottom and a menu with "Dashboard", "API Keys", "Settings", "Usage", and "Billing". The main content area is titled "Welcome, fintech" and displays a grid of user information cards. The cards contain the following data:

ID	GOOGLE ID	EMAIL	STATUS	SOCIETY TYPE	BUSINESS PLAN
2	---	fintech@gmail.com	Active	USER	DENIED
API KEY	API KEY EXPIRY				
Not assigned	---				

Figure 84: Dashboard Gestion Clé pour les Fintechs

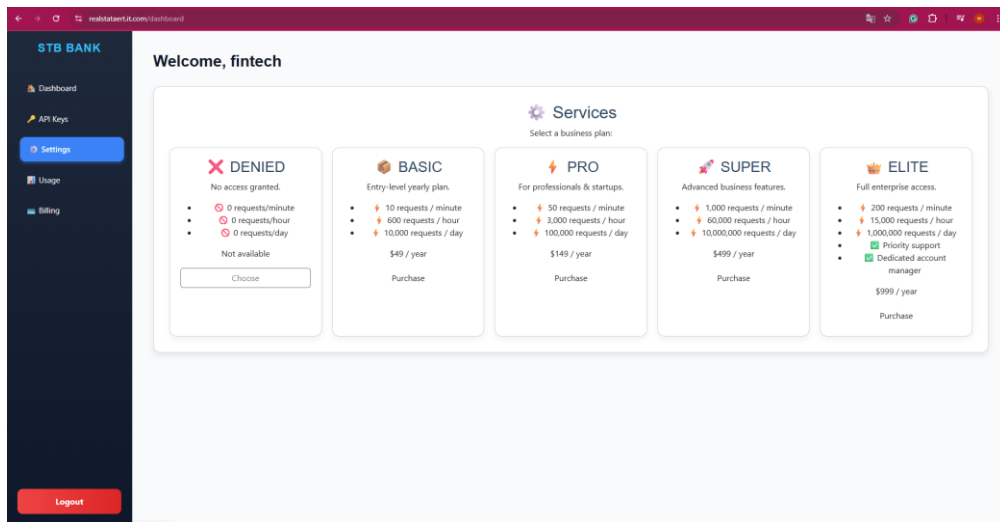


Figure 85: Offres des services

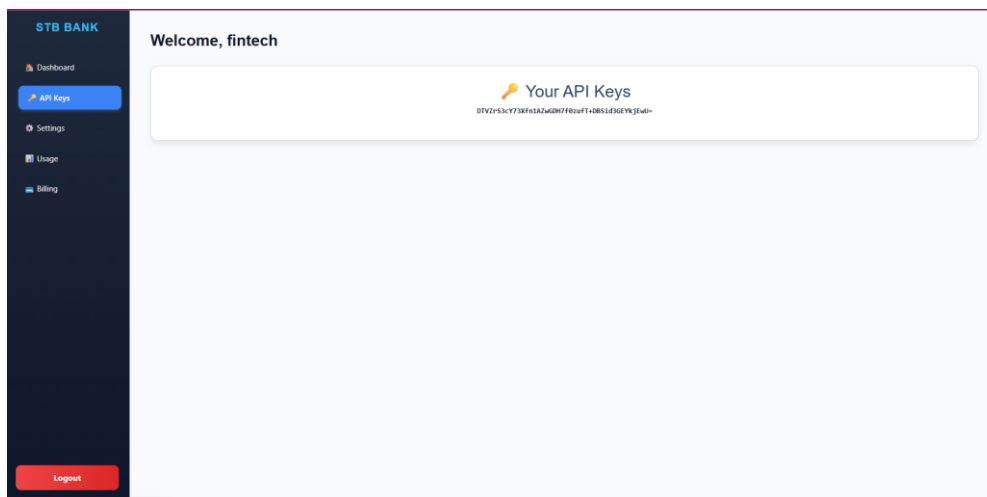


Figure 86: Interface clé

3.8.5.4. Consommation documentation

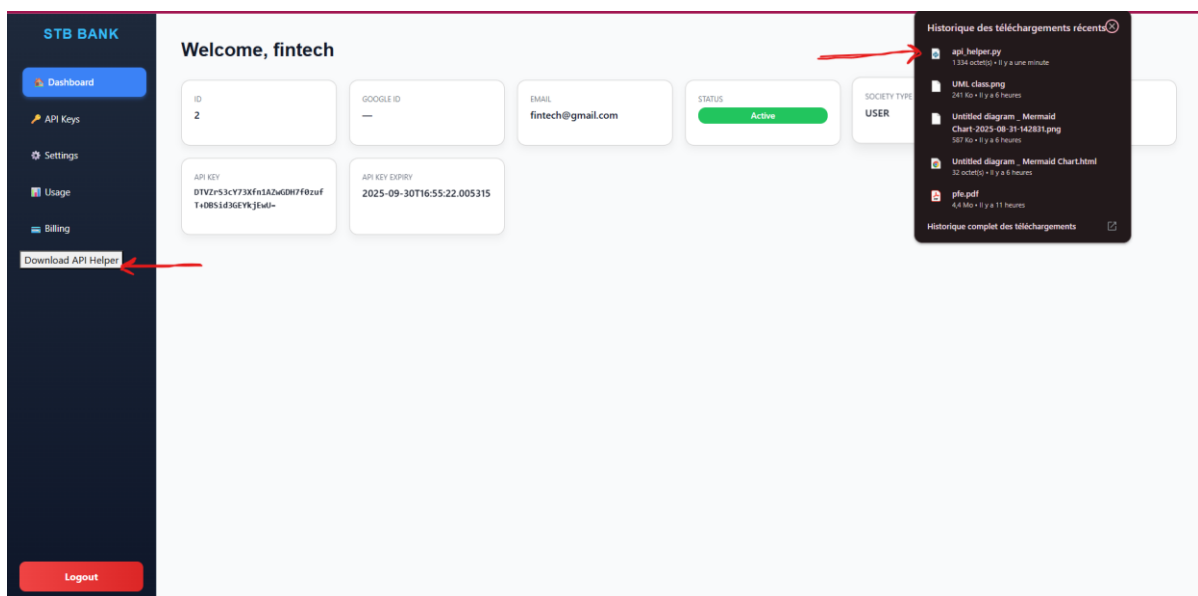


Figure 87: Téléchargement de documentation

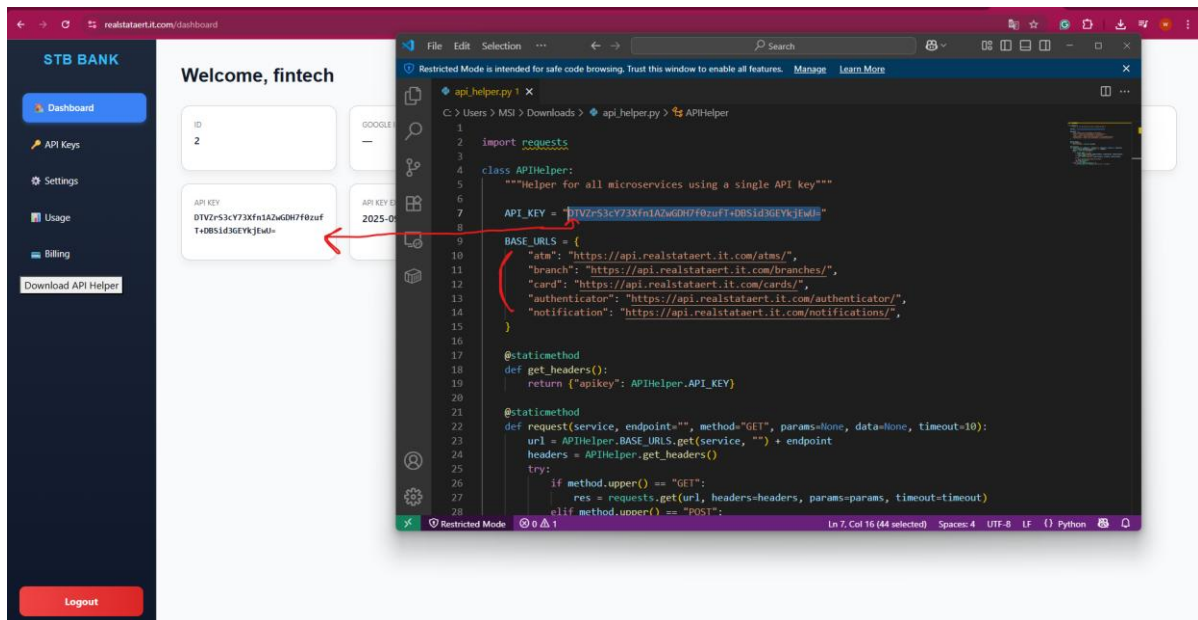


Figure 88: consommation du api_helper dans des autres projets

3.8.5.5. Intégration helper dans des autres projets

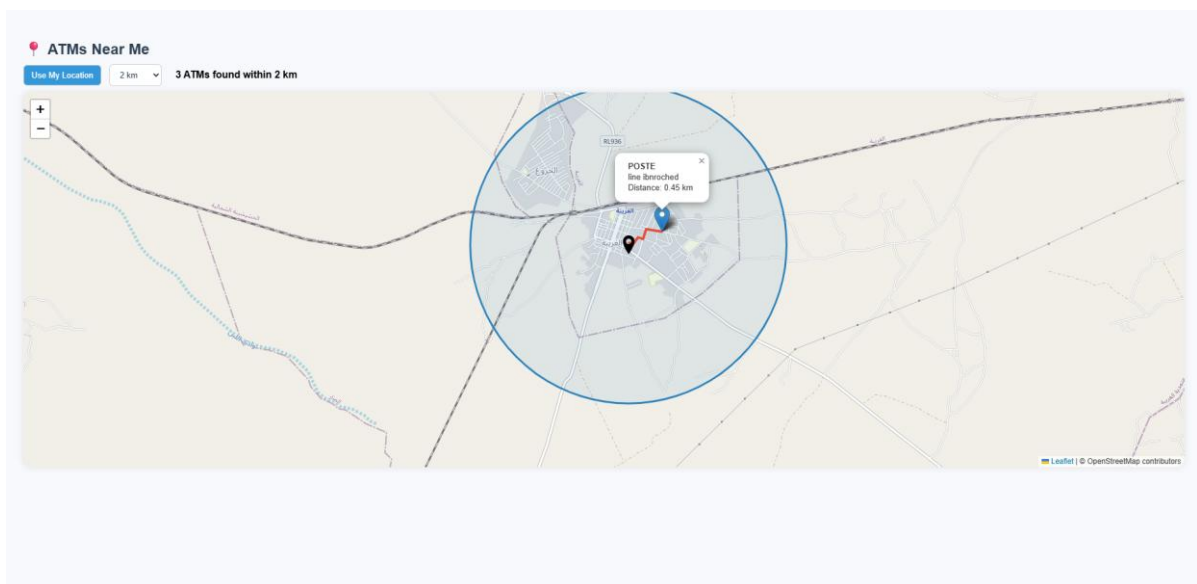


Figure 89: application Fintech pour Visualisations des distributeurs actifs avec localisation

La figure ci-dessus illustre une application Fintech développée par un client **sur notre plateforme Banking as a Service (Baas)**. Cette application consomme nos API distribuées, notamment l'API ATM, pour récupérer et afficher :

- ❖ **Les distributeurs actifs** et leur état en temps réel (disponible, en maintenance ou hors service).
- ❖ **La localisation géographique** permet de visualiser les distributeurs les plus proches de l'utilisateur sur une carte interactive.
- ❖ **L'activité des transactions et l'utilisation des services** reflètent les interactions réelles avec le système.

En intégrant nos API, l'application Fintech démontre que notre plateforme BaaS :

- ❖ **Supporte un usage réel** par des clients externes développant leurs propres applications.
- ❖ **Gère efficacement des microservices distribués**, avec une communication orchestrée via RabbitMQ et le Kong API Gateway.
- ❖ **Fournit des services scalables, fiables et observables**, validés par la consommation en temps réel des API ATM, comptes et agences.

En résumé, cette application **prouve que notre système BaaS fonctionne en conditions réelles**, permettant aux clients de construire des applications bancaires pleinement opérationnelles tout en exploitant nos microservices et nos données en temps réel.

3.8.6. Burndown Chart

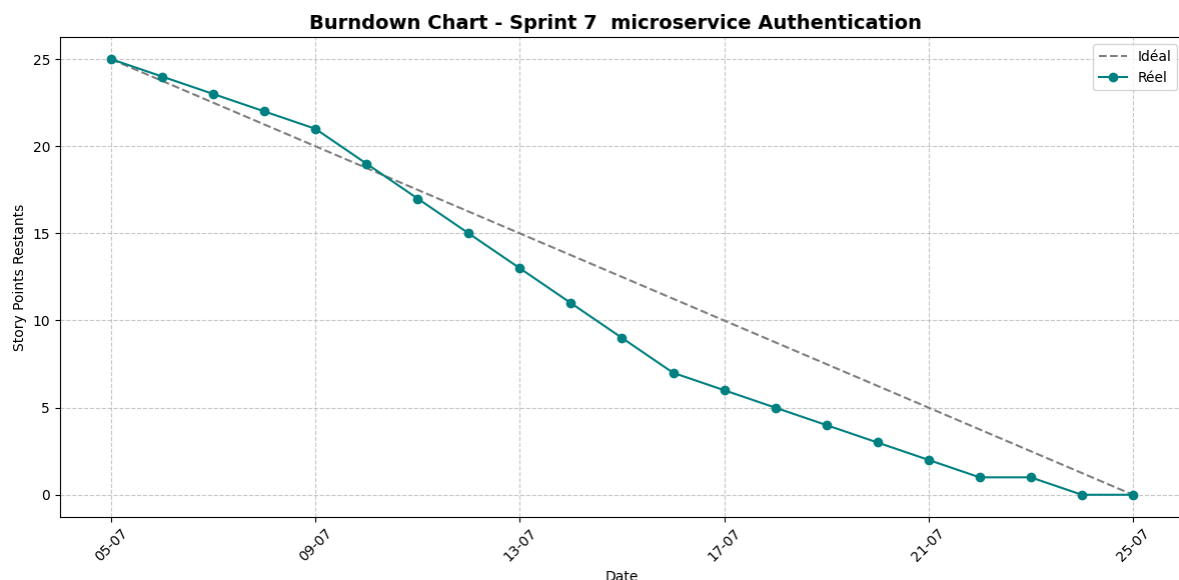


Figure 90: Burndown Chart, Sprint 7

3.9. Sprint 8 : Déploiement & Monitoring (Docker, OVH, UFW,SSL)

3.9.1. Introduction :

Le Sprint 8 avait pour objectif le déploiement de l'application Open Banking dans un environnement de production simulé. L'accent a été mis sur la mise en place d'une infrastructure sécurisée et scalable à travers l'utilisation de Docker, Kubernetes et l'hébergement sur un VPS OVH.

Ce sprint constitue une étape cruciale, car il marque la transition de la phase de développement et de tests locaux vers un déploiement sur serveur réel, accessible au public via un nom de domaine sécurisé par SSL.

3.9.2. Backlog du sprint

Rôle	User Story	Tâche	Estimation
DevOps	En tant qu'administrateur système, je souhaite disposer d'un VPS performant pour héberger l'application afin de garantir la disponibilité et la sécurité.	Choisir le VPS OVH adapté, effectuer le paiement et mise en service	5 pts
DevOps	En tant qu'administrateur, je souhaite sécuriser le serveur pour réduire les risques d'attaques	Modifier le port SSH, configurer Fail2Ban, activer UFW	4 pts
DevOps	En tant qu'utilisateur, je souhaite accéder à l'application via un nom de domaine personnalisé sécurisé par SSL.	Configurer le DNS, générer certificat SSL avec Certbot	3 pts
DevOps	En tant qu'équipe technique, nous souhaitons protéger et accélérer le site	Intégrer Cloudflare pour la sécurité et le CDN	3 pts
DevOps	En tant que développeur, je souhaite déployer l'application sous forme de conteneurs pour simplifier la maintenance et le scaling	Déploiement des microservices Docker, configuration des ports	5 pts
DevOps	En tant qu'administrateur, je souhaite surveiller l'état des services et des métriques	Configurer Prometheus, Grafana, Kibana, RabbitMQ Management	5 pts
Frontend	En tant qu'utilisateur, je souhaite accéder à l'interface web de manière fluide.	Vérifier l'accès aux dashboards Angular, Account Dashboard et Konga	3 pts
QA	En tant que testeur, je souhaite	Test DNS, ping, https, accès ports et	3 pts

	m'assurer de la disponibilité et de la sécurité du serveur.	conteneurs	
--	---	------------	--

Tableau 10: Backlog Sprint 8

3.9.3. Revu du Sprint

Au terme du Sprint 8, tous les buts prévus pour la mise en œuvre de l'application Open Banking sur un environnement de production simulé ont été réalisés avec succès. Le sprint a facilité la sécurisation et l'accessibilité de l'application grâce à un VPS OVH, tout en assurant une infrastructure modulable et surveillée.

En termes de sécurité, le serveur a été renforcé en changeant le port par défaut SSH, en configurant Fail2Ban afin d'éviter les tentatives d'intrusion et en appliquant le pare-feu UFW pour limiter l'accès seulement aux services indispensables. L'intégration de Cloudflare a offert une protection supplémentaire contre les attaques DDoS tout en optimisant les performances via le CDN.

Pour l'accessibilité, le domaine realstataert.it.com a été configuré via Namecheap, et les tests de résolution DNS et de connectivité ont confirmé le bon routage vers notre VPS OVH. La génération d'un certificat SSL/TLS avec Certbot a permis de sécuriser les échanges entre les utilisateurs et le serveur, assurant la confidentialité et l'intégrité des communications. L'application est désormais accessible en HTTPS avec un cadenas de sécurité valide.

Le déploiement des microservices via Docker a été mené à bien, chaque composant (frontend, backend, bases de données, services de notifications, API Gateway, tableaux de bord) étant fonctionnel et accessible sur les ports dédiés. Cette pratique de la conteneurisation rend les mises à jour, l'entretien et l'évolutivité future plus aisés.

Pour finir, grâce à l'intégration de Prometheus, Grafana, Kibana et RabbitMQ Management pour la supervision et le suivi, il est désormais possible de surveiller en temps réel l'état et les performances des services. Les tests de fin ont attesté de la solidité, de la sûreté et de la disponibilité de l'infrastructure mise en place.

Malgré quelques ajustements nécessaires sur les configurations initiales (ports, accès dashboard et SSL), le sprint a posé une **base solide pour la mise en production**, garantissant une plateforme sécurisée, performante et scalable pour l'Open Banking.

3.9.4. Démonstration

3.9.4.1.1. Choix du VPS(Serveur virtuel Privé)

Pour héberger et tester notre application Open Banking, nous avons choisi un **VPS (Serveur Virtuel Privé)**. Un VPS offre les avantages d'un serveur dédié avec un coût réduit et la flexibilité du cloud.

Nouveau VPS-1 À partir de 12,920 DT HT/mois soit 15,246 DT TTC/mois Frais d'installation: Offert Configurer ✓ 4 vCores ✓ 8 Go RAM ✓ 75 Go SSD NVMe ✓ Sauvegarde automatisée 1 jour ✓ Trafic illimité ✓ 400 Mbit/s bande passante publique	Nouveau VPS-2 À partir de 20,100 DT HT/mois soit 23,718 DT TTC/mois Frais d'installation: Offert Configurer ✓ 6 vCores ✓ 12 Go RAM ✓ 100 Go SSD NVMe ✓ Sauvegarde automatisée 1 jour ✓ Trafic illimité ✓ 1 Gbit/s bande passante publique	Nouveau VPS-3 À partir de 40,200 DT HT/mois soit 47,436 DT TTC/mois Frais d'installation: Offert Configurer ✓ 8 vCores ✓ 24 Go RAM ✓ 200 Go SSD NVMe ✓ Sauvegarde automatisée 1 jour ✓ Trafic illimité ✓ 1,5 Gbit/s bande passante publique	Nouveau VPS-4 À partir de 71,700 DT HT/mois soit 84,606 DT TTC/mois Frais d'installation: Offert Configurer ✓ 12 vCores ✓ 48 Go RAM ✓ 300 Go SSD NVMe ✓ Sauvegarde automatisée 1 jour ✓ Trafic illimité ✓ 2 Gbit/s bande passante publique	Nouveau VPS-5 À partir de 106 DT HT/mois soit 125,080 DT TTC/mois Frais d'installation: Offert Configurer ✓ 16 vCores ✓ 64 Go RAM ✓ 350 Go SSD NVMe ✓ Sauvegarde automatisée 1 jour ✓ Trafic illimité ✓ 2,5 Gbit/s bande passante publique	Nouveau VPS-6 À partir de 140,400 DT HT/mois soit 165,672 DT TTC/mois Frais d'installation: Offert Configurer ✓ 24 vCores ✓ 96 Go RAM ✓ 400 Go SSD NVMe ✓ Sauvegarde automatisée 1 jour ✓ Trafic illimité ✓ 3 Gbit/s bande passante publique
--	--	--	---	---	---

Figure 91: gamme des Vps d' OVH

Après comparaison des offres disponibles chez OVH Cloud, nous avons opté pour le **VPS-2** avec la configuration suivante :

- ❖ **CPU**: 6 vCores
- ❖ **RAM** : 12 Go
- ❖ **Stockage**: 100 Go SSD NVMe
- ❖ **Sauvegarde automatisée** : 1 jour
- ❖ **Bande passante publique** : 1 Gbit/s, trafic illimité
- ❖ **Prix** : 20,100 DT TTC/mois, frais d'installation offerts
- ❖ **Système d'exploitation** : Debian 12

Cette configuration assure une performance optimale pour le déploiement et le test de l'ensemble de notre application, tout en garantissant la sécurité et la fiabilité.

Choisissez le nombre de VPS et leur localisation

Vous pouvez héberger votre VPS dans un ou plusieurs de nos datacenters au sein de notre réseau mondial.

Europe Amérique du Nord Asie

☒ Europe (Allemagne)

☐ Europe (France - Gravelines)

☐ Europe (France - Strasbourg)

☐ Europe (Royaume-Uni)

☐ Europe (Pologne - Varsovie)

Choisissez votre image

Distribution uniquement Distribution avec application

☐ Plesk Debian ☐ cPanel ☒ Docker ☐ n8n

Web Admin 1 account Debian 12 Debian 12

Plesk (Debian 12) cPanel (AlmaLinux 9)

27,000 din HT/mois 37,000 din HT/mois **Gratuit** **Gratuit**

VPS

VPS-2 23,600 din

Processeur 6 vCore

Mémoire 12 Go RAM

Stockage 100 Go SSD NVMe

Bande passante publique 1 Gbit/s illimitée

Engagement **Aucun engagement** 23,600 din

Localisation du datacenter 1x Europe (Allemagne)

Image Debian 12 - Docker **Gratuit**

Option Option Automated Backup Standard - VPS-2 0,000 din

Total 23,600 din

1 produit HT/mois

[Continuer la commande](#)

Figure 92: choix du VPS

3.9.4.1.2. Paiement et mise en service

Le VPS a été configuré et payé via le portail OVH Cloud. Les frais d'installation ont été offerts, ce qui nous a permis de démarrer immédiatement le serveur.

✓ Votre commande # 23... a été traitée !

Merci d'avoir choisi OVHcloud !

[Imprimer le résumé de la commande](#)

[Suivre ma commande](#)

[Configurer](#)

[Historique d'achat](#)

[Affichez ou modifiez votre compte](#)

Figure 93 : Achat du VPS

3.9.4.1.3. Connexion et configuration initiale du VPS

★ Connexion SSH

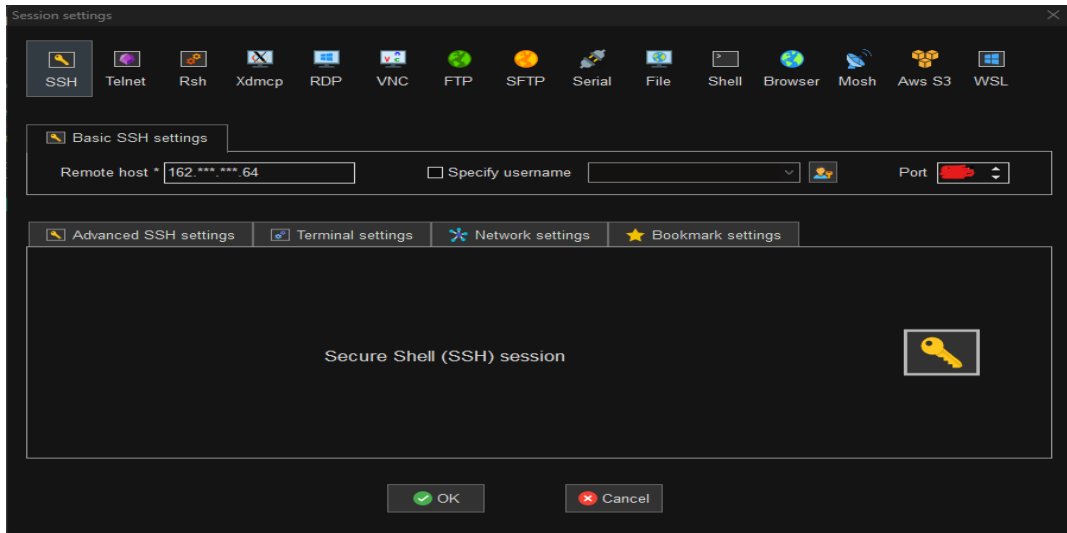


Figure 94: Connexion via le portail mobaxterm

★ Mise à jour du système

Après connexion, le système a été mis à jour pour assurer les dernières corrections de sécurité :

```
debian@vps-162-***.***.64:~$ sudo apt update && sudo apt upgrade -y
Get:1 file:/etc/apt/mirrors/debian.list Mirrorlist [30 B]
Get:2 file:/etc/apt/mirrors/debian-security.list Mirrorlist [39 B]
Hit:3 https://deb.debian.org/debian bookworm InRelease
Hit:7 https://download.docker.com/linux/debian bookworm InRelease
Hit:4 https://deb.debian.org/debian bookworm-updates InRelease
Hit:5 https://deb.debian.org/debian bookworm-backports InRelease
Hit:6 https://deb.debian.org/debian-security bookworm-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
11 packages can be upgraded. Run 'apt list --upgradable' to see them.
W: https://download.docker.com/linux/debian/dists/bookworm/InRelease: Key is st
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
```

Figure 95 : Mise à jour Du Système

★ Vérification de Docker

Docker a été installé et son statut vérifié :

```

debian@vps:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; preset: enabled)
   Active: active (running) since Thu 2025-08-28 16:18:42 UTC; 4min 0s ago
 TriggeredBy: ● docker.socket
    Docs: https://docs.docker.com
   Main PID: 64833 (dockerd)
      Tasks: 12
     Memory: 29.5M
        CPU: 396ms
    CGroup: /system.slice/docker.service
            └─64833 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

```

Figure 96 : Vérification Du Docker

Cette étape permet de garantir que la plateforme est prête pour le déploiement des microservices.

3.9.4.1.4. Sécurisation du VPS

★ Modification du port SSH

Pour réduire les risques d'attaques automatiques, le port SSH par défaut (22) a été modifié dans le fichier **/etc/ssh/sshd config** :

```
# This is the sshd server system-wide configuration file.  See
# sshd_config(5) for more information.

# This sshd was compiled with PATH=/usr/local/bin:/usr/bin:/bin:/usr/games

# The strategy used for options in the default sshd_config shipped with
# OpenSSH is to specify options with their default value where
# possible, but leave them commented.  Uncommented options override the
# default value.

Include /etc/ssh/sshd_config.d/*.conf

Port 4499
#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::
```

Figure 97 : Modification du port SSH

Ensuite, le service SSH a été redémarré :

```

debian@vps-:~$ sudo nano /etc/ssh/sshd_config
debian@vps-:~$ sudo systemctl restart ssh
debian@vps-:~$

```

Figure 98: redémarrage du service ssh

★ Configuration Fail2Ban

Fail2Ban est un outil qui protège contre les tentatives d'intrusion répétées. La configuration pour SSH a été ajoutée dans **/etc/fail2ban/jail.local** :

```
#
# JAILS
#
#
# SSH servers
#

[sshd]
enabled = true
port = 22
filter = sshd
logpath = /var/log/auth.log
backend = systemd
maxretry = 5
findtime = 600
bantime = 3600
```

Figure 99: Configuration fail2ban

- ❖ **maxretry** : nombre de tentatives avant ban (5)
- ❖ **findtime** : intervalle de recherche (600 secondes)
- ❖ **bantime** : durée du ban (3600 secondes)

Fail2Ban a été relancé :

```

debian@vps-11111111:~$ sudo systemctl status fail2ban
● fail2ban.service - Fail2Ban Service
   Loaded: loaded (/lib/systemd/system/fail2ban.service; enabled; preset: enabled)
   Active: active (running) since Thu 2025-08-28 14:43:12 UTC; 37s ago
     Docs: man:fail2ban(1)
  Main PID: 30101 (fail2ban-server)
    Tasks: 5 (limit: 14000)
   Memory: 14.1M
      CPU: 147ms
   CGroup: /system.slice/fail2ban.service
           └─30101 /usr/bin/python3 /usr/bin/fail2ban-server -xf start

Aug 28 14:43:12 vps-11111111 systemd[1]: Started fail2ban.service - Fail2Ban Service.
Aug 28 14:43:12 vps-11111111 fail2ban-server[30101]: 2025-08-28 14:43:12,754 fail2ban.configreader [30101]: WARNING 'allowip6' not defined in 'Definition'. Using default one: 'auto'
Aug 28 14:43:12 vps-11111111 fail2ban-server[30101]: Server ready

```

Figure 100: statut.fail2ban

```
debian@vps-XXXXXXXXXX:~$ sudo fail2ban-client status  
Status  
|- Number of jail:      1  
|_ Jail list:    sshd  
  
debian@vps-XXXXXXXXXX:~$ sudo fail2ban-client status sshd  
Status for the jail: sshd  
|- Filter  
|   |- Currently failed: 0  
|   |- Total failed:     0  
|   '- File list:        /var/log/auth.log  
- Actions  
|   |- Currently banned: 0  
|   |- Total banned:     0  
|   '- Banned IP list:
```

Figure 101: statut du jail

★ Activation du pare-feu UFW

Services accessibles depuis l'extérieur (clients, dashboards, monitoring)

- ❖ Frontend Angular : 8083 → accès web
- ❖ Dashboard Account: 18888 et 18889
- ❖ Kibana : 5601 → monitoring ES
- ❖ Prometheus : 9090 → monitoring metrics
- ❖ Grafana: 3000 → dashboards
- ❖ Kong (API Gateway) : 8000, 8443 → proxies clients ; 8001, 8444 → admin
- ❖ Konga : 1337 → dashboard admin de Kong
- ❖ RabbitMQ Management UI: 15672 (optionnel, monitoring)

```
debian@vps-...:~$ sudo ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
debian@vps-...:~$ sudo ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), deny (routed)
New profiles: skip

To Action From
--
22/tcp ALLOW IN Anywhere
2375/tcp ALLOW IN Anywhere
8083/tcp ALLOW IN Anywhere
18888/tcp ALLOW IN Anywhere
18889/tcp ALLOW IN Anywhere
5601/tcp ALLOW IN Anywhere
3000/tcp ALLOW IN Anywhere
9090/tcp ALLOW IN Anywhere
8000/tcp ALLOW IN Anywhere
8443/tcp ALLOW IN Anywhere
8001/tcp ALLOW IN Anywhere
8444/tcp ALLOW IN Anywhere
1337/tcp ALLOW IN Anywhere
15672/tcp ALLOW IN Anywhere
4199/tcp (v6) ALLOW IN Anywhere (v6)
2375/tcp (v6) ALLOW IN Anywhere (v6)
8083/tcp (v6) ALLOW IN Anywhere (v6)
18888/tcp (v6) ALLOW IN Anywhere (v6)
18889/tcp (v6) ALLOW IN Anywhere (v6)
5601/tcp (v6) ALLOW IN Anywhere (v6)
3000/tcp (v6) ALLOW IN Anywhere (v6)
9090/tcp (v6) ALLOW IN Anywhere (v6)
8000/tcp (v6) ALLOW IN Anywhere (v6)
8443/tcp (v6) ALLOW IN Anywhere (v6)
8001/tcp (v6) ALLOW IN Anywhere (v6)
8444/tcp (v6) ALLOW IN Anywhere (v6)
1337/tcp (v6) ALLOW IN Anywhere (v6)
15672/tcp (v6) ALLOW IN Anywhere (v6)
```

Figure 102: liste des ports à accédé

3.9.4.1.5. DNS configuration

Afin de rendre le système accessible publiquement via un nom de domaine, nous avons configuré le DNS sur **Namecheap** en associant notre serveur hébergé chez OVH (IP publique 162.**.**.64) à un domaine personnalisé : realstataert.it.com.

☐	A Record	@	162.***.64	Automatic	
☐	A Record	www	162.***.64	Automatic	
☐	CNAME Record	www	realstataert.it.com.	Automatic	
☐	A Record	api	162.***.64	Automatic	
☐	A Record	auth	162.***.64	Automatic	

Figure 103: Configuration des records DNS

une fois que les enregistrements terminés on va tester la résolution avec nslookup et ping

```
C:\Users\MSI>nslookup realstataert.it.com
Serveur : dns.google
Address: 8.8.8.8

Réponse ne faisant pas autorité :
Nom : realstataert.it.com
Address: 162.***.64
```

Figure 104 : Résolution DNS (nslookup)

```
C:\Users\MSI>ping api.realstataert.it.com

Envoi d'une requête 'ping' sur api.realstataert.it.com [162.***.64] avec 32 octets de données :
Réponse de 162.***.64 : octets=32 temps=52 ms TTL=48
Réponse de 162.***.64 : octets=32 temps=52 ms TTL=48
Réponse de 162.***.64 : octets=32 temps=52 ms TTL=48
Réponse de 162.***.64 : octets=32 temps=53 ms TTL=48

Statistiques Ping pour 162.***.64:
    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
    Durée approximative des boucles en millisecondes :
        Minimum = 52ms, Maximum = 53ms, Moyenne = 52ms

C:\Users\MSI>ping auth.realstataert.it.com

Envoi d'une requête 'ping' sur auth.realstataert.it.com [162.***.64] avec 32 octets de données :
Réponse de 162.***.64 : octets=32 temps=52 ms TTL=48
Réponse de 162.***.64 : octets=32 temps=63 ms TTL=48
Réponse de 162.***.64 : octets=32 temps=52 ms TTL=48
Réponse de 162.***.64 : octets=32 temps=60 ms TTL=48

Statistiques Ping pour 162.***.64:
    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
    Durée approximative des boucles en millisecondes :
        Minimum = 52ms, Maximum = 63ms, Moyenne = 56ms
```

Figure 105 : Test de connectivité (ping)

Ces résultats attestent que l'adresse `realstataert.it.com` est bien dirigée vers notre serveur OVH, ce qui permet d'accéder au projet via une URL facile à mémoriser.

3.9.4.1.6. Configuration certificat SSL

Afin de sécuriser les échanges entre le client et le serveur, nous avons intégré un certificat SSL/TLS. Pour cela, l'outil **Certbot** (Let's Encrypt) a été utilisé afin d'émettre un certificat gratuit et renouvelable automatiquement.

Les étapes réalisées :

```
debian@vps-3020b30b:~/projet$ sudo apt update
sudo apt install nginx certbot python3-certbot-nginx -y
Get:1 file:/etc/apt/mirrors/debian.list Mirrorlist [30 B]
Get:2 file:/etc/apt/mirrors/debian-security.list Mirrorlist [39 B]
Hit:7 https://download.docker.com/linux/debian bookworm InRelease
Hit:3 https://deb.debian.org/debian bookworm InRelease
Hit:4 https://deb.debian.org/debian bookworm-updates InRelease
Hit:5 https://deb.debian.org/debian bookworm-backports InRelease
Hit:6 https://deb.debian.org/debian-security bookworm-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Figure 106: Installation de Certbot et du plugin Nginx

```
debian@vps-3020b30b:~/projet$ sudo apt update
sudo apt install certbot python3-certbot-nginx -y

# Obtain SSL for each domain/subdomain
sudo certbot --nginx -d realstataert.it.com
sudo certbot --nginx -d auth.realstataert.it.com
sudo certbot --nginx -d api.realstataert.it.com
Get:1 file:/etc/apt/mirrors/debian.list Mirrorlist [30 B]
```

Figure 107 : Génération du certificat pour le domaine

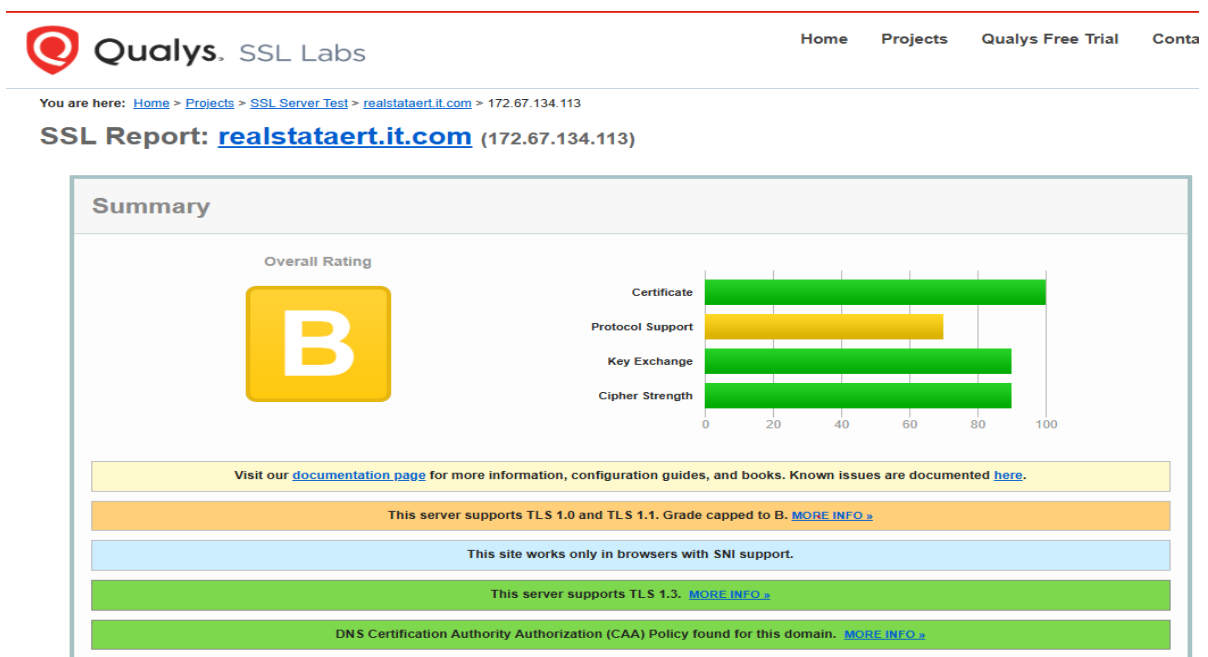


Figure 108 : Test du SSL

L'application est désormais accessible via **https://realstataert.it.com** avec un cadenas de sécurité valide, garantissant la confidentialité et l'intégrité des communications.

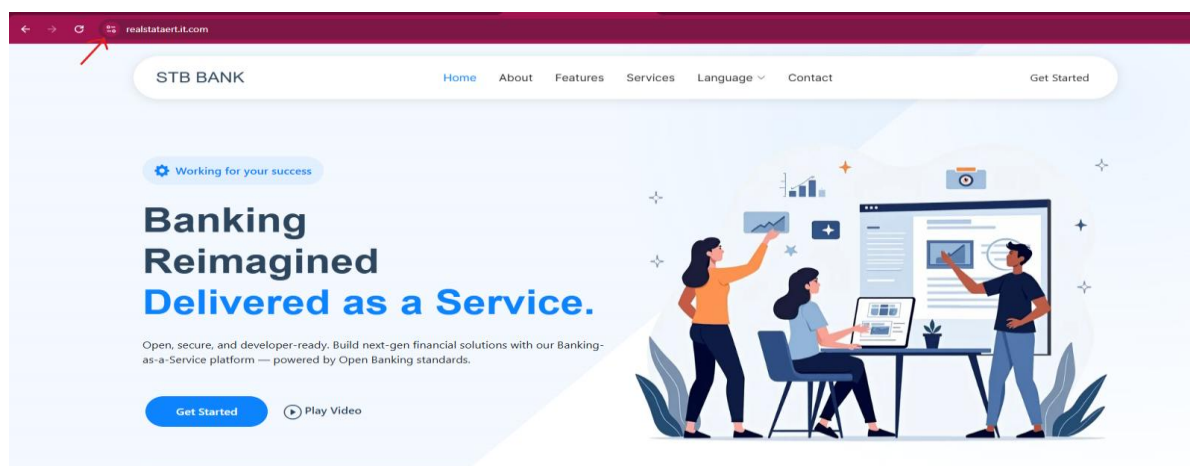


Figure 109 : <https://realstataert.it.com>

3.9.4.1.7. Intégration CloudFlare

Afin de renforcer la sécurité et d'optimiser les performances du site, nous avons intégré **Cloudflare** comme couche de protection DNS et de proxy. parmi ces avantages :

Protection DDoS : filtrage automatique du trafic malveillant.

CDN : mise en cache du contenu statique et réduction de la latence.

Gestion DNS avancée : propagation rapide des enregistrements DNS.

SSL flexible : compatibilité avec le certificat Let's Encrypt déjà installé.

☐	Type ①	▲ Name ①	Content ①	Proxy status ①	TTL ①	Actions
☐	A	api	162.159.166.64	Proxied	Auto	Edit ▶
☐	A	auth	162.159.166.64	Proxied	Auto	Edit ▶
☐	A	realstataert.it.com	162.159.166.64	Proxied	Auto	Edit ▶
☐	CNAME	www	realstataert.it.com	Proxied	Auto	Edit ▶

Figure 110 : : Configuration des enregistrements DNS

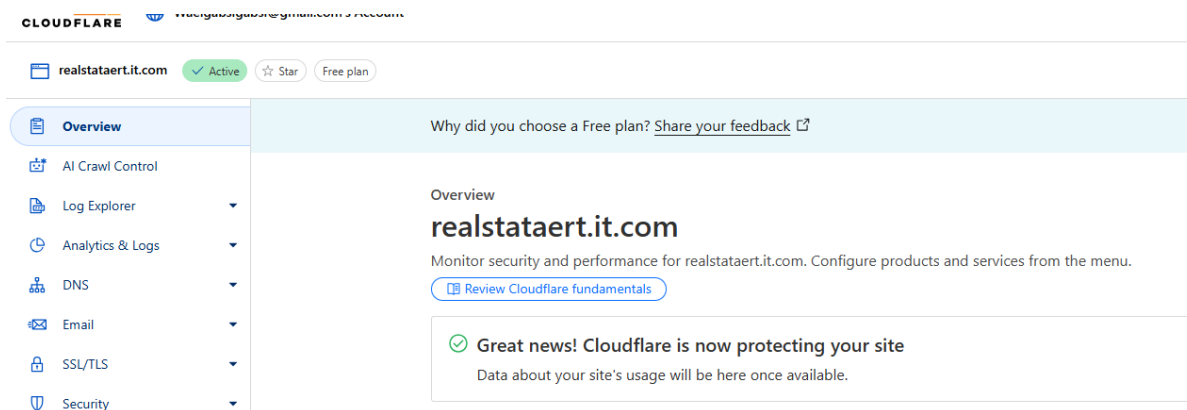


Figure 111 : : Application est sécurisée

3.9.4.1.8. Déploiement des conteneurs Docker

L'application a été déployée sous forme de multiples conteneurs Docker, chacun représentant un micro service ou un composant d'infrastructure. La commande **docker ps** (voir figure ci-dessous) montre l'ensemble des conteneurs actifs, leurs images, leurs statuts ainsi que les ports exposés.

CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS	PORTS
025e58d48964	banking_solution_pfe_2025-frontend	banking_solution_pfe_2025-frontend-1	"nginx -g 'daemon off;'"	2 hours ago	Up 2 hours (healthy)	0.0.0.0:8090->80/tcp, [::]:8090->80/tcp
01db11a20f59	banking_solution_pfe_2025-authenticator-service	authenticator-service	"dotnet OpenBanking..."	3 hours ago	Up 3 hours	0.0.0.0:8090->8090/tcp, [::]:8090->8090/tcp, 8091/tcp
1be40ae19ca5	pantsel/konga	konga	"./app/start.sh"	4 hours ago	Up 4 hours	0.0.0.0:1337->1337/tcp, [::]:1337->1337/tcp
6db92933dfc2	kong:3.6	kong	"/docker-entrypoint..."	4 hours ago	Up 4 hours (healthy)	0.0.0.0:8000-8001->8000-8001/tcp, [::]:8000-8001->8000-8001/tcp, 0.0.0.0:8443-8444->8443-8444/tcp, [::]:8443-8444->8443-8444/tcp
adabab8864ca	grafana/grafana	grafana	"./run.sh"	4 hours ago	Up 4 hours	0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp
6d15c3002129	banking_solution_pfe_2025-card-service	card-service	"dotnet OpenBanking..."	4 hours ago	Up 4 hours	0.0.0.0:8086-8087->8086-8087/tcp, [::]:8086-8087->8086-8087/tcp
a01e428370ce	docker.elastic.co/kibana/kibana:8.13.0	kibana	"./bin/tini -- /usr/L..."	4 hours ago	Up 4 hours	0.0.0.0:5601->5601/tcp, [::]:5601->5601/tcp
2f7e1ab0e55e	banking_solution_pfe_2025-notification-service	notification-service	"dotnet OpenBanking..."	4 hours ago	Up 4 hours	0.0.0.0:9094-9095->9094-9095/tcp, [::]:9094-9095->9094-9095/tcp
b7822450c8dc	banking_solution_pfe_2025-branch-service	branch-service	"dotnet OpenBanking..."	4 hours ago	Up 4 hours	0.0.0.0:8084-8085->8084-8085/tcp, [::]:8084-8085->8084-8085/tcp
c786866dc609	banking_solution_pfe_2025-atm-service	atm-service	"dotnet OpenBanking..."	4 hours ago	Up 4 hours	0.0.0.0:8082-8083->8082-8083/tcp, [::]:8082-8083->8082-8083/tcp
6a1e1213215a	redis:latest	redis	"docker-entrypoint.s..."	4 hours ago	Up 4 hours	0.0.0.0:6379->6379/tcp, [::]:6379->6379/tcp
1605b26d8a8f	mysql:5.7	mysql	"docker-entrypoint.s..."	4 hours ago	Up 4 hours (healthy)	0.0.0.0:3306->3306/tcp, [::]:3306->3306/tcp, 33060/tcp
0b57e34c4c31	postgres:11	postgres	"docker-entrypoint.s..."	4 hours ago	Up 4 hours (healthy)	0.0.0.0:5440->5442/tcp, [::]:5440->5442/tcp
c774b8cd8be1	postgres:15-alpine	postgres	"docker-entrypoint.s..."	4 hours ago	Up 4 hours (healthy)	0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp
e93b10898c95	mcr.microsoft.com/dotnet/aspire-dashboards:latest	aspire-dashboards	"dotnet /app/Aspire..."	4 hours ago	Up 4 hours	0.0.0.0:18888-18889->18888-18889/tcp, [::]:18888-18889->18888-18889/tcp
43da9edef105	prom/prometheus	prometheus	"./bin/prometheus --c..."	4 hours ago	Up 4 hours	0.0.0.0:9090->9090/tcp, [::]:9090->9090/tcp
478e8999c72b	postgres:13	postgres	"docker-entrypoint.s..."	4 hours ago	Up 4 hours (healthy)	0.0.0.0:5433->5432/tcp, [::]:5433->5432/tcp
8f6e81125d9	rabbitmq:3-management	rabbitmq	"docker-entrypoint.s..."	4 hours ago	Up 4 hours (healthy)	4369/tcp, 5671/tcp, 0.0.0.0:5672->5672/tcp, [::]:5672->5672/tcp, 15691-15692/tcp, 25672/tcp
8c122cfd91e	docker.elastic.co/elasticsearch/elasticsearch:8.13.0	elasticsearch	"./bin/tini -- /usr/L..."	4 hours ago	Up 4 hours	0.0.0.0:9200->9200/tcp, [::]:9200->9200/tcp, 9300/tcp
62c411b799e	mongo:latest	mongo	"docker-entrypoint.s..."	4 hours ago	Up 4 hours (healthy)	0.0.0.0:27017->27017/tcp, [::]:27017->27017/tcp
8aaffdcfc5cf	mysql:5.7	mysql	"docker-entrypoint.s..."	4 hours ago	Up 4 hours (healthy)	33060/tcp, 0.0.0.0:3307->3306/tcp, [::]:3307->3306/tcp

Figure 112 : Conteneurs Docker

Service / Conteneur	Image	Ports exposés	role
Frontend	banking_solution_pfe_2025-frontend	8090, 8091	Interface utilisateur
Authenticator Service	banking_solution_pfe_2025-authenticator-service	1337	Authentification
Konga (Kong Dashboard)	pantsel/kong	1337	Dashboard API Gateway
Kong (API Gateway)	kong:3.6	8000-8001, 8443-	Proxy & gestion API

		8444	
Grafana	grafana/grafana	3000	Dashboards de supervision
Card Service	banking_solution_pfe_2025-card-service	8086-8087	Gestion des cartes
Kibana	docker. Elastic.co/kibana:8.13.0	5601	Visualisation des logs
Notification Service	banking_solution_pfe_2025-notification-service	9094-9095	Envoi de notifications
Branch Service	banking_solution_pfe_2025-branch-service	8084-8085	Gestion des agences
ATM Service	banking_solution_pfe_2025-atm-service	8082-8083	Gestion des distributeurs
Redis	redis: latest	6379	Cache & sessions
MySQL	mysql:5.7	3306, 3307	Base de données SQL
PostgreSQL	postgres:11 / 13 / 15-alpine	5432-5433	Base de données relationnelle
Dashboard Account	mcr.microsoft.com/dotnet/aspire-dashboard	18888-18889	Dashboard applicatif
Prometheus	prom/prometheus	9090	Collecte de métriques
RabbitMQ Management +	rabbitmq:3-management	15672, 5672, 4369, 25672	File de messages + monitoring
Elasticsearch	docker. Elastic.co/elasticsearch:8.13.0	9200, 9300, 27017	Indexation et recherche
MongoDB	mongo:latest	27017	Base de données NoSQL

Tableau 11: Détails du Conteneurs

3.9.5. Visualisation du Système

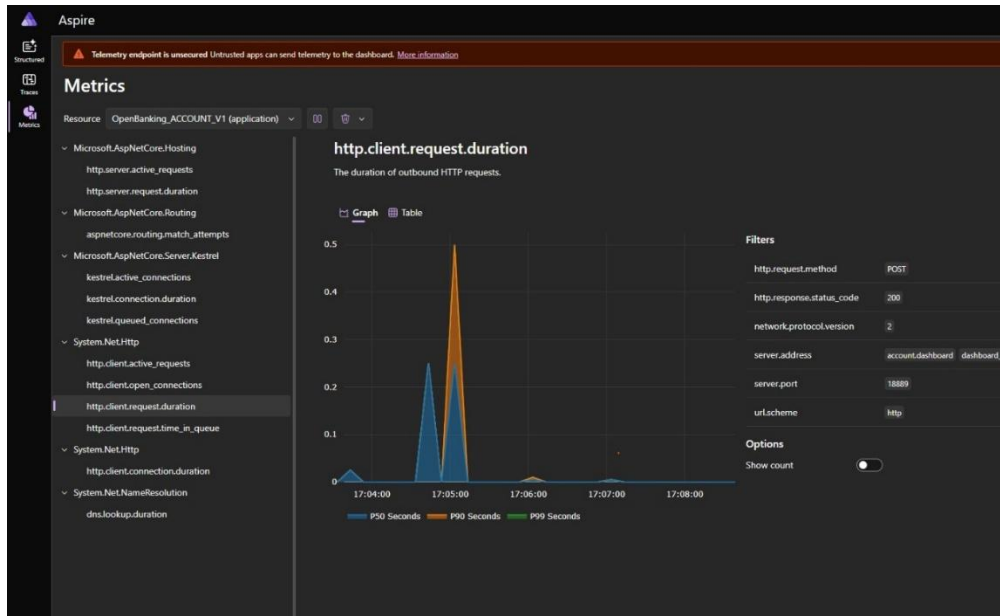


Figure 113: Aspire Dashboard

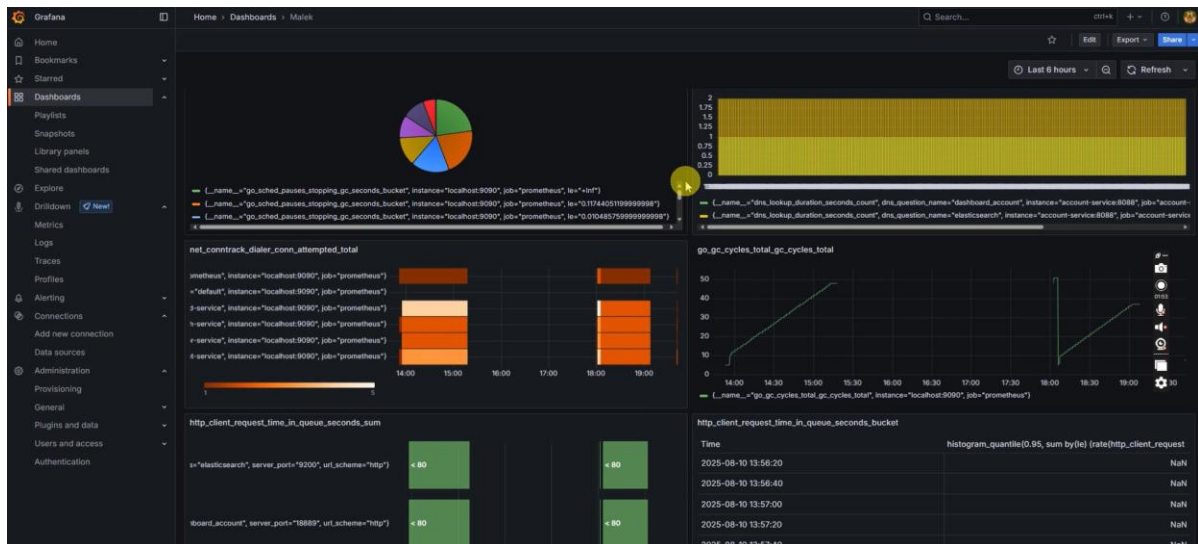
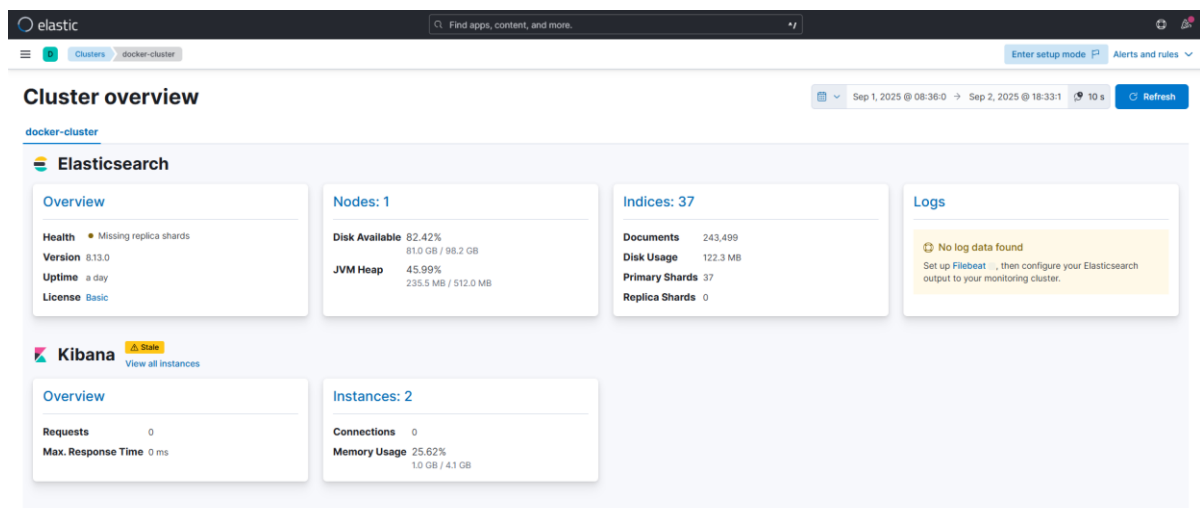
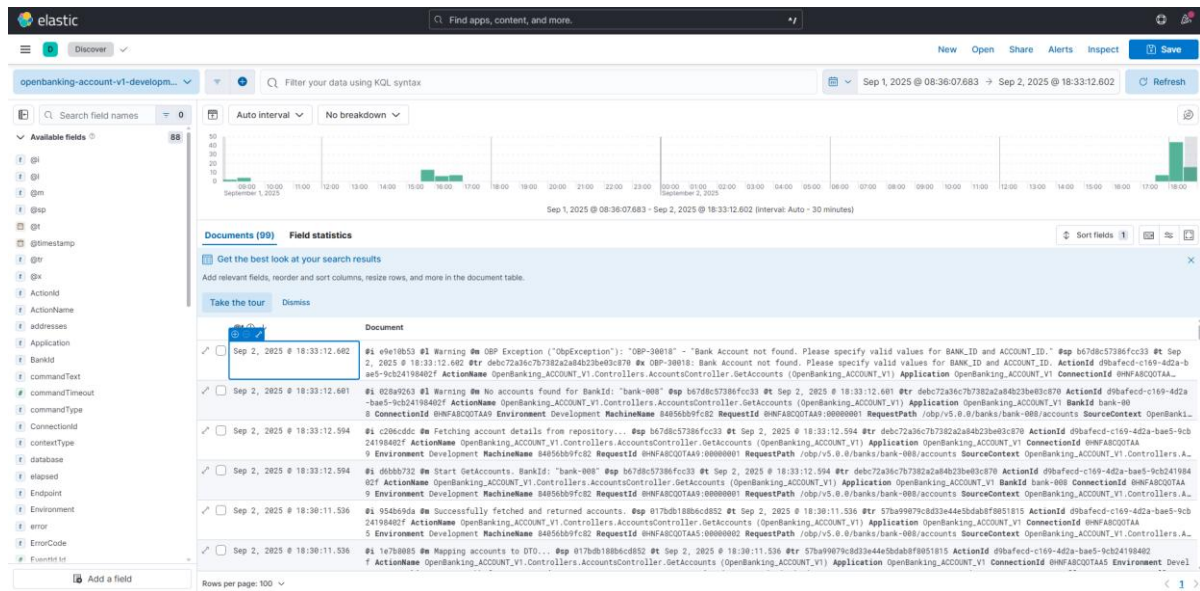


Figure 114 : Grafana Dashboard



3.9.6. Burndown Chart

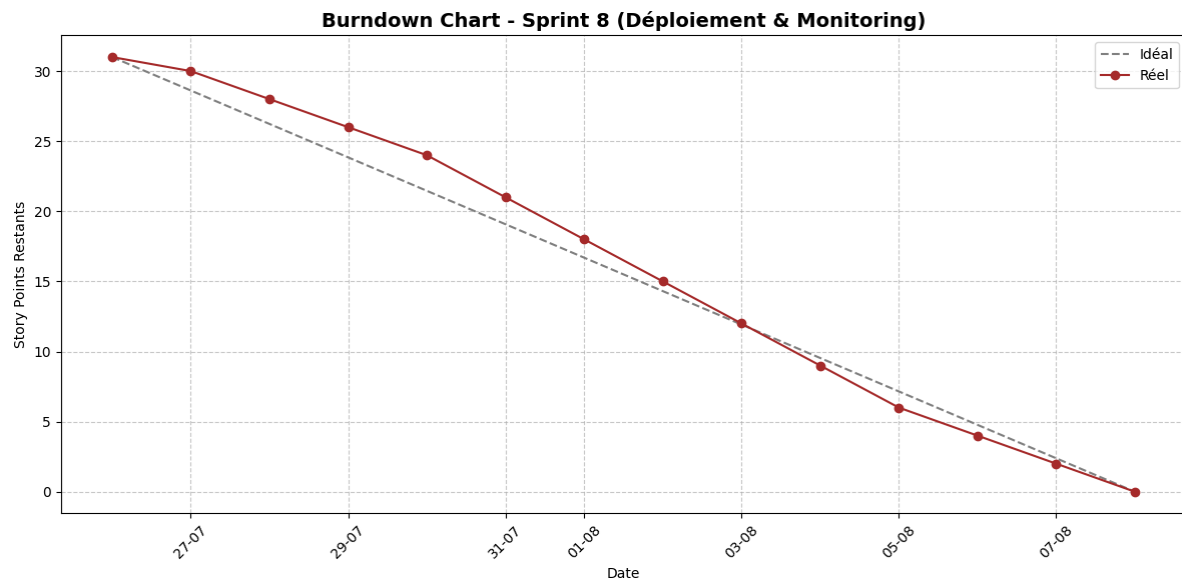


Figure 117 : Burndown Chart Sprint 8

Chapitre 4 : Conclusion & Perspectives

4. Chapitre 4 : Conclusion & Perspective

Ce projet de fin d'études a permis de concevoir et de mettre en œuvre une plateforme bancaire innovante basée sur une architecture en microservices, déployée dans un environnement conteneurisé et orchestrée par Kubernetes. L'approche retenue a démontré son efficacité en termes de modularité, de scalabilité et de sécurité, répondant ainsi aux enjeux actuels de la transformation digitale du secteur bancaire.

La méthodologie Agile SCRUM a favorisé une progression itérative et incrémentale, assurant un suivi rigoureux des besoins et une validation progressive des livrables. Chaque sprint a contribué à enrichir le système, depuis l'analyse initiale jusqu'au déploiement final sur un environnement de production sécurisé. Les technologies adoptées, telles que RabbitMQ, gRPC, Kong API Gateway, Docker/Kubernetes, ainsi que les outils de supervision (Prometheus, Grafana, Kibana, Elasticsearch), ont permis d'aboutir à une solution robuste et conforme aux standards Open Banking.

Sur le plan pratique, le projet a montré qu'il est possible de proposer une solution **Banking-as-a-Service (BaaS)** adaptée à la fois aux institutions financières traditionnelles et aux fintechs, favorisant l'interopérabilité, la transparence et la rapidité d'intégration. L'expérience acquise tout au long de ce travail constitue un apport significatif, tant sur le plan technique que méthodologique.

Perspectives

Malgré les résultats atteints, plusieurs pistes d'amélioration et d'évolution peuvent être envisagées :

- ❖ **Élargissement fonctionnel** : ajouter de nouveaux microservices tels que la gestion des prêts, des investissements ou des assurances, afin d'offrir une plateforme plus complète.
- ❖ **Optimisation de la performance** : intégrer des mécanismes avancés de scaling automatique et de tolérance aux pannes pour renforcer la résilience du système.
- ❖ **Sécurité renforcée** : mettre en place une solution de gestion des identités et des accès (IAM) plus avancée, avec support du MFA (Multi-Factor Authentication).
- ❖ **Expérience utilisateur** : développer des interfaces mobiles (Android/iOS) pour compléter le frontend Angular et toucher un plus large public.
- ❖ **Ouverture internationale** : adapter la plateforme aux réglementations européennes (PSD2) et internationales, afin de favoriser son adoption au-delà du marché tunisien.
- ❖ **Intégration IA et Big Data** : exploiter des modèles d'intelligence artificielle pour analyser les comportements des clients, proposer des services personnalisés et renforcer la détection des fraudes.

Ainsi, ce projet constitue une première étape vers une solution bancaire moderne, évolutive et orientée vers l'avenir, capable d'accompagner la transition numérique du secteur financier.

Références Bibliographiques

1. Wikipédia. « Société tunisienne de banque». Article en ligne. Disponible sur: https://fr.wikipedia.org/wiki/Soci%C3%A9t%C3%A9_tunisienne_de_banque. Consulté le 7 avril 2025.
2. DOCS. « PRESENTATION RÉSUMÉE DE LA SOCIÉTÉ TUNISIENNE DE BANQUE ». pdf en ligne. Disponible sur : chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.cmf.tn/sites/default/files/pdfs/emetteurs/informations/references/doc_reference_stb_2021.pdf. Consulté le 7 avril 2025.
3. Communication financière de la BIAT : Projets et réalisations de l'année 2024 <https://www.biat.com.tn/biat-la-une/press-media/communication-financiere-de-la-biat-projets-et-realizations-de-lannee-2024>
4. Assemblée Générale de la BIAT : Bilan 2024 Positif, Transformation Digitale et Engagement Sociétal <https://www.webmanagercenter.com/2025/04/25/545054/assemblee-generale-de-la-biat-bilan-2024-positif-transformation-digitale-et-engagement-societal>
5. BNP Paribas and BPCE join forces in payment processing <https://www.reuters.com/business/finance/bnp-paribas-bpce-join-forces-payment-processing-2025-02-13>
6. HFDJSHGFJ@213JFKS
7. BNP Paribas to hire 60 salespeople in global markets drive <https://www.fnlondon.com/articles/bnp-paribas-to-hire-60-salespeople-in-global-markets-drive-509462cb>
8. Meilleure stratégie de digitalisation <https://www.biat.com.tn/communication-financiere/recompenses-bancaires>

9. BIAT (Banque Internationale Arabe de Tunisie): Best Digitalisation Strategy North Africa 2024 <https://cfi.co/awards/africa/2024/biat-banque-internationale-arabe-de-tunisie-best-digitalisation-strategy-north-africa-2024>
10. BNP Paribas, Mistral AI ink partnership deal <https://www.reuters.com/markets/deals/bnp-paribas-mistral-ai-ink-partnership-deal-2024-07-10>
11. BIAT : projets et réalisations de l'année 2024 <https://www.businessnews.com.tn/biat--projets-et-realizations-de-lannee-2024%2C544%2C147218%2C3>
12. Hello bank! https://en.wikipedia.org/wiki/Hello_bank%21
13. Definition JWT <https://auth0.com/fr/learn/json-web-tokens>



ECOLE SUPÉRIEURE PRIVÉE D'INGÉNIERIE ET DE TECHNOLOGIES

www.esprit.tn - E-mail : contact@esprit.tn

Siège Social : 18 rue de l'Usine - Charguia II - 2035 - Tél. : +216 71 941 541 - Fax. : +216 71 941 889

Annexe : Z.I. Chotrana II - B.P. 160 - 2083 - Pôle Technologique - El Ghazala - Tél. : +216 70 685 685 - Fax. : +216 70 685 454