Evaluation Certifiante

Pilotage d’un projet d’automatisation du SAV à partir de tweets clients : planification, ressources et contraintes

Bloc 3 - C3.1

Réalisé par :

SGHIOURI Mohammed
SOBGUI Ivan Joel
BOTI Armel Cyrille
BEN LOL Oumar
DIVENGI KIBANGUDI BUNKEMBO Nagui
ELOUMOU MBOUDOU Pascal Aurele

Date : 19 Novembre 2025

Table des matières

Liste des tableaux

Liste des figures

Introduction

Dans le cadre d'une mission confiée à une entreprise de Data Consulting par Free, il s'agit de développer une solution d'automatisation du SAV basée sur le traitement de 5 000 tweets via LLM. Ce projet présente plusieurs défis de pilotage : un délai contraint de 3 semaines, des données hétérogènes, et une dépendance critique aux API LLM (accès, latence, coûts).

Face à ces contraintes, une stratégie de pilotage structurée est nécessaire pour respecter les délais, mobiliser les ressources appropriées et anticiper les points critiques. Ce travail se concentre sur la planification, sans aborder la gestion de crise (traitée dans d'autres travaux).

1. Méthodologie de gestion de projet

1.1. Choix de la méthodologie Agile adaptée

L'analyse des caractéristiques du projet révèle plusieurs éléments déterminants pour le choix de la méthodologie de gestion : un délai très contraint de trois semaines, des incertitudes techniques liées au traitement de données brutes et à l'utilisation d'API externes, la nécessité de livrer une démonstration fonctionnelle rapidement, et un besoin de flexibilité pour s'adapter aux contraintes émergentes liées aux LLM.

Face à ces caractéristiques, le choix s'oriente vers une méthodologie Agile de type **Scrum adapté**, combinée à des éléments de **Kanban** pour la gestion des flux de traitement des données. Cette approche hybride permet de bénéficier de la structure et de la cadence offertes par Scrum, tout en conservant la flexibilité nécessaire pour gérer les tâches de traitement de données qui peuvent être parallélisées et nécessitent un suivi en flux continu.

Le choix de Scrum adapté se justifie par plusieurs éléments : la nécessité de livrer rapidement des incréments fonctionnels, la présence d'incertitudes techniques nécessitant des ajustements fréquents, et le besoin de communication intensive entre les membres de l'équipe pour coordonner les travaux sur les données, les API LLM et le développement du dashboard.

1.2. Justification du choix méthodologique

Justification en quatre dimensions :

1.2.1. Adaptation au délai contraint

Scrum permet de prioriser les fonctionnalités essentielles et de livrer rapidement. La structuration en sprints courts de 1 semaine garantit le traitement prioritaire des éléments critiques.

1.2.2. Gestion des incertitudes techniques

Une approche itérative est nécessaire face aux incertitudes des données brutes, les API LLM. Les caractéristiques des tweets (langage informel, emojis) et les contraintes API (quotas, latence) peuvent nécessiter des ajustements en cours de projet.

1.2.3. Flexibilité face aux dépendances externes

La dépendance aux API tierces nécessite une capacité d'adaptation rapide. La méthode Agile est adaptée car elle permet de réagir rapidement aux changements sans remettre en cause la planification.

1.2.4. Coordination des compétences multiples

La coordination de compétences variées telles que Data Engineering, backend/frontend, LLM, UX tout en travaillant en parallèle avec les rituels Scrum (daily stand-up, sprint planning, sprint review) deviennent plus simple.

1.3. Organisation des itérations et rôles clés

L'organisation du projet repose sur une structure en trois sprints d'une semaine chacun, précédés d'une phase de préparation initiale et suivis d'une phase de finalisation et de soutenance. Cette structure permet de garantir une progression régulière vers l'objectif de démonstration fonctionnelle tout en conservant une marge de manœuvre pour les ajustements nécessaires.

1.3.1. Structure des sprints

Chaque sprint dure une semaine et suit le cycle classique Scrum : sprint planning en début de semaine, daily stand-up quotidiens, et sprint review en fin de semaine. Cette cadence permet de maintenir un rythme de travail soutenu tout en conservant la flexibilité nécessaire pour s'adapter aux contraintes émergentes.

1.3.2. Rôles et responsabilités

La répartition des rôles suit la structure Scrum classique, adaptée aux spécificités techniques du projet :

Product Owner (PO) : Responsable de la définition et de la priorisation des fonctionnalités, en lien avec les attentes du client Free. Le PO valide les choix techniques concernant les LLM et garantit que la solution répond aux besoins opérationnels identifiés.

Scrum Master : Assure la facilitation des rituels Scrum, la résolution des blocages, et la coordination entre les équipes. Dans le contexte de ce projet, le Scrum Master joue également un rôle crucial dans la gestion des dépendances externes aux API LLM et dans l'anticipation des risques techniques.

Data Engineer : Responsable de la préparation, du nettoyage et du préprocessing des données brutes issues du fichier free tweet export.csv. Cette fonction est critique car la qualité du traitement des données conditionne directement la performance des modèles LLM.

Développeur Backend : En charge du développement de l'infrastructure de traitement des données, de l'intégration des API LLM, et de la mise en place des pipelines de traitement. Ce rôle nécessite une expertise technique approfondie sur les API REST, la gestion des quotas et des limites de taux, et l'optimisation des appels API.

Développeur Frontend : Responsable du développement du dashboard de visualisation et d'analyse des résultats. Ce rôle nécessite des compétences en développement web moderne (Next.js, React, TypeScript) et en visualisation de données (Recharts).

UX Designer : En charge de la conception de l'interface utilisateur du dashboard, en veillant à ce que les visualisations soient intuitives et répondent aux besoins des utilisateurs finaux (agents du service client, managers, analystes).

Data Scientist / ML Engineer : Responsable de la configuration et de l'optimisation des appels aux LLM, de la définition des prompts efficaces pour la classification et l'analyse de sentiment, et de l'évaluation de la qualité des résultats.

2. Planification du projet

2.1. Découpage en phases opérationnelles

Le projet est structuré en cinq phases principales, chacune correspondant à un objectif opérationnel précis et contribuant à la livraison de la démonstration fonctionnelle dans le délai imparti.

2.1.1. Phase 1 : Préparation et exploration des données (Jours 1-3)

La phase initiale constitue le fondement du projet et vise à comprendre la nature des données à traiter, à identifier les défis techniques spécifiques, et à préparer l'environnement de développement. Les activités principales incluent :

- **Exploration et analyse exploratoire des données** : Analyse statistique du fichier free tweet export.csv pour identifier les caractéristiques des données (volume, distribution, qualité, présence de bruit)
- **Définition de la stratégie de nettoyage** : Identification des techniques de préprocessing nécessaires (normalisation, gestion des emojis, traitement des mentions et hashtags)
- **Mise en place de l'environnement de développement** : Configuration des outils, des dépôts Git, et de l'infrastructure de base
- **Sélection et configuration initiale des API LLM** : Choix du ou des fournisseurs de LLM (Gemini Flash, Mistral Small, GPT-4o-mini, GPT-4o), obtention des clés API, et tests préliminaires de connectivité

2.1.2. Phase 2 : Traitement NLP via LLM (Jours 4-10)

- **Développement du pipeline de préprocessing** : Implémentation des fonctions de nettoyage et de normalisation des tweets
- **Intégration des API LLM** : Développement des modules d'appel aux API, gestion des quotas et des limites de taux, implémentation de mécanismes de retry et de gestion d'erreurs
- **Définition et optimisation des prompts** : Création et affinement des prompts pour la classification des tweets (type de demande, urgence, sentiment) et l'extraction d'informations structurées
- **Traitement par lots des 5 000 tweets** : Exécution du traitement avec gestion de la progression et des erreurs, ajout de trois colonnes supplémentaires au dataset (sentiment, priority, main_topic), et stockage des résultats dans une base de données structurée
- **Validation et évaluation de la qualité** : Analyse de la qualité des résultats obtenus, identification des cas limites et des erreurs de classification

2.1.3. Phase 3 : Développement du dashboard (Jours 11-15)

- **Conception UX du dashboard** : Définition de la structure et des composants visuels en collaboration avec les utilisateurs finaux
- **Système d'authentification** : Implémentation de l'authentification JWT avec gestion des sessions et contrôle d'accès basé sur les rôles (administrateurs et analystes de données)
- **Gestion des utilisateurs** : Développement des fonctionnalités de gestion des utilisateurs et des rôles pour permettre l'administration de l'application
- **Développement frontend** : Implémentation des composants de visualisation (graphiques, tableaux, filtres) avec Recharts
- **Développement backend API** : Création des endpoints API pour l'accès aux données traitées, aux métriques calculées, et à la gestion des utilisateurs
- **Intégration frontend-backend** : Connexion de l'interface aux données et tests d'intégration
- **Intégration RabbitMQ** : Connexion du frontend à RabbitMQ pour recevoir les mises à jour en temps réel des tâches
- **Optimisation des performances** : Amélioration des temps de chargement et de la réactivité de l'interface

2.1.4. Phase 4 : Tests techniques et validation (Jours 16-18)

Pour garantir la qualité de la solution avant la démonstration des test ont été efectués :

- **Tests unitaires et d'intégration** : Vérification du bon fonctionnement de chaque composant et de leur intégration
- **Tests de performance** : Validation des temps de traitement et de la capacité à gérer le volume de données
- **Tests de régression** : Vérification que les modifications n'ont pas introduit de régressions

- **Validation fonctionnelle** : Vérification que la solution répond aux besoins fonctionnels identifiés
- **Préparation de la démonstration** : Préparation des scénarios de démonstration et des supports de présentation

2.1.5. Phase 5 : Finalisation et soutenance (Jours 19-21)

- **Corrections des bugs critiques** : Résolution des problèmes identifiés lors des tests
- **Documentation technique et utilisateur** : Rédaction de la documentation nécessaire pour l'utilisation et la maintenance de la solution
- **Préparation de la soutenance** : Préparation de la présentation, des démonstrations, et des réponses aux questions potentielles
- **Livraison de la démonstration fonctionnelle** : Mise à disposition de la solution pour la démonstration au client

2.2. Estimation des durées et jalons critiques

Estimation avec marges de sécurité pour les activités à risque élevé (API LLM, données brutes).

Tableau 2 : Planification détaillée par phase

Phase	Durée	Jalon critique	Risque principal
Phase 1 : Préparation des données	3 jours	Validation stratégie nettoyage	Complexité inattendue
Phase 2 : Traitement NLP via LLM	7 jours	Traitement complet 5 000 tweets	Dépendance API, quotas, coûts
Phase 3 : Développement dashboard	5 jours	Dashboard fonctionnel	Intégration frontend-backend
Phase 4 : Tests et validation	3 jours	Validation fonctionnelle	Bugs critiques
Phase 5 : Finalisation et soutenance	3 jours	Livraison démonstration	Préparation présentation

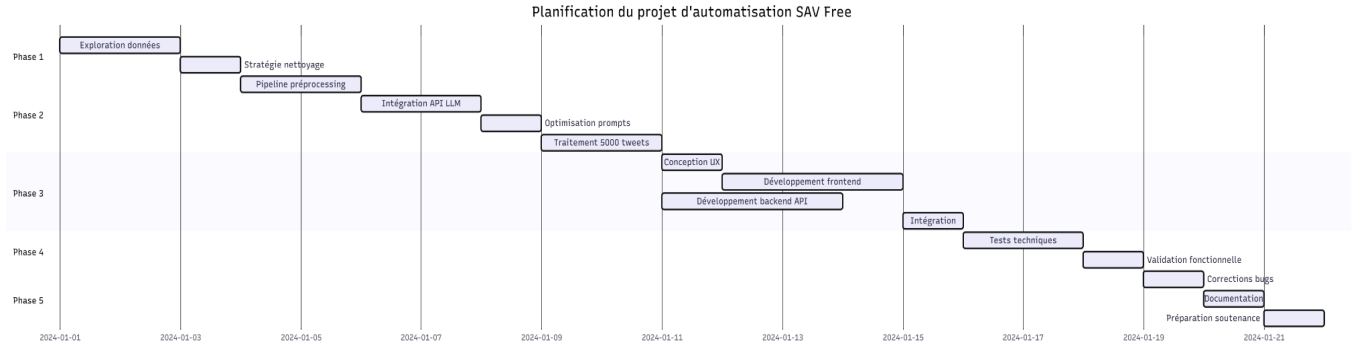
Total : 21 jours (3 semaines)

Chaque jalon fait l'objet d'une validation formelle avant passage à la phase suivante.

2.3. Schéma de planification temporelle

Le schéma de planification suivant présente visuellement la répartition temporelle des phases et des activités principales :

Figure 1 : Diagramme de Gantt simplifié du projet



A ce stade, on arrive à iidentifier les chevauchements possibles entre les phases (par exemple, le développement frontend et backend peuvent être partiellement parallélisés) et de visualiser les dépendances critiques, notamment entre le traitement des données et le développement du dashboard.

3. Mobilisation des ressources

3.1. Ressources humaines et compétences requises

La réussite du projet dépend de la mobilisation d’une équipe pluridisciplinaire possédant les compétences techniques et méthodologiques nécessaires pour mener à bien les différentes phases. L’analyse des besoins révèle la nécessité de combiner des expertises complémentaires dans les domaines du traitement de données, du développement logiciel, de l’intégration d’API, et de la visualisation.

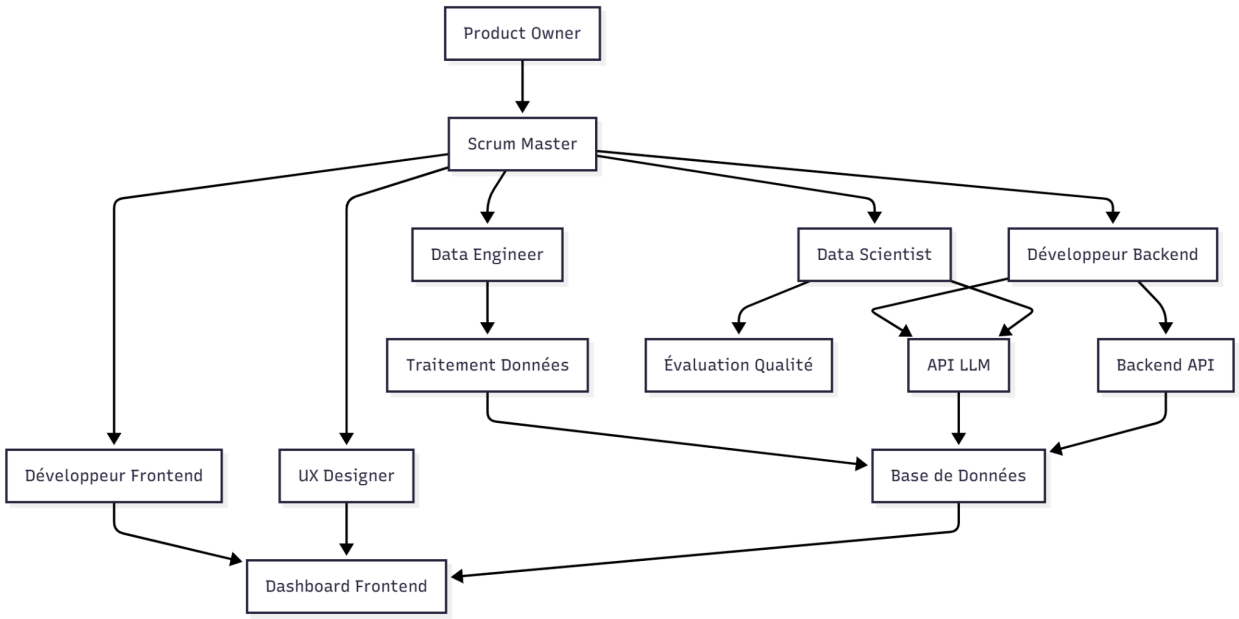
Tableau 1 : Répartition des rôles et responsabilités

Rôle	Charge estimée	Compétences clés	Responsabilités principales
Product Owner	20%	Gestion de projet, communication client	Priorisation, validation fonctionnelle, interface client
Scrum Master	30%	Méthodologie Agile, facilitation	Animation rituels, résolution blocages, coordination
Data Engineer	80%	Python, pandas, traitement données textuelles	Préparation et nettoyage des données, pipelines ETL
Développeur Backend	100%	Express.js, TypeScript, API REST, gestion asynchrone	Intégration API LLM, développement API, optimisation
Développeur Frontend	80%	Next.js, React, TypeScript, Recharts	Développement dashboard, composants visuels
UX Designer	40%	Design interface, expérience utilisateur	Conception interface, wireframes, prototypage
Data Scientist	60%	NLP, LLM, optimisation prompts	Configuration LLM, évaluation qualité, fine-tuning

Total équipe : 4,1 ETP (équivalents temps plein)

Certains rôles peuvent être combinés si les compétences le permettent (ex. : Data Engineer + Data Scientist).

Figure 2 : Architecture de l’équipe projet



3.2. Contraintes d'accès aux API LLM

L'utilisation des API LLM tierces introduit des contraintes significatives qui doivent être intégrées dès la phase de planification pour éviter des blocages en cours de projet. Ces contraintes concernent principalement l'accès aux API, les quotas d'utilisation, les coûts associés, et les risques réglementaires.

3.2.1. Accès et authentification

Obtention de clés API auprès des fournisseurs (Google Gemini, Mistral AI, OpenAI). Délais possibles pour comptes professionnels ou volumes importants. Prévoir cette étape dès le début et valider les clés avant le démarrage du traitement.

3.2.2. Quotas et limites de taux

Les API LLM imposent des limites de taux (rate limiting) qui peuvent affecter la vitesse de traitement des 5 000 tweets. Ces limites varient selon le fournisseur et le type de compte :

- **Google Gemini 1.5 Flash** : Limite variable selon le type de compte (gratuit avec limitations, payant avec quotas plus élevés)
- **Mistral AI (Small)** : Limite de 50 requêtes par minute pour les comptes de base
- **OpenAI GPT-4o-mini / GPT-4o** : Limite typique de 40-60 requêtes par minute selon le modèle et le type de compte

Pour traiter 5 000 tweets avec une limite de 40 requêtes par minute, le temps minimal de traitement serait d'environ 125 minutes (2 heures), sans compter les temps de traitement des requêtes elles-mêmes. En pratique, avec des temps de réponse moyens de 2-3 secondes par

requête, le traitement complet peut nécessiter 4-6 heures, ce qui doit être intégré dans la planification.

3.2.3. Estimation des coûts

Les coûts associés à l'utilisation des API LLM constituent une contrainte budgétaire importante à anticiper. L'estimation des coûts dépend du modèle choisi, de la longueur des prompts, et du volume de tokens traités.

Tableau 3 : Estimation des coûts API LLM

Priorité	Fournisseur	Modèle	Coût (Input/Output)	Cas d'usage
1 (Défaut)	Google	Gemini 1.5 Flash	Très faible / Gratuit (limité)	Classification rapide, Sentiment, Tri de masse
2	Mistral AI	Mistral Small	Faible	Traitement souverain (Europe), tâches intermédiaires
3	OpenAI	GPT-4o-mini	Moyen	Analyse complexe nécessitant du raisonnement
4	OpenAI	GPT-4o	Élevé	Uniquement pour les cas très ambigus ou VIP

*Note : Les prix sont dynamiques et surveillés en temps réel par le module d'administration. La stratégie adoptée est "Modèle le moins cher suffisant" pour optimiser les coûts.

Estimation à valider avec le client. La marge de sécurité recommandée est de 20-30%.

3.2.4. Stratégie de gestion des quotas et coûts

Pour optimiser l'utilisation des API et minimiser les coûts, plusieurs stratégies peuvent être mises en place :

- **Traitement par lots avec gestion de la file d'attente** : Implémentation d'un système de file d'attente qui respecte les limites de taux et gère automatiquement les retries en cas d'erreur
- **Mise en cache des résultats** : Stockage des résultats de traitement pour éviter les appels redondants en cas de retraitement
- **Optimisation des prompts** : Réduction de la longueur des prompts tout en conservant leur efficacité pour minimiser le nombre de tokens consommés

- **Choix du modèle adapté** : Utilisation de modèles moins coûteux (Gemini Flash pour le tri de masse, Mistral Small pour les tâches intermédiaires) plutôt que GPT-4o, réservant les modèles plus puissants (GPT-4o-mini, GPT-4o) aux cas complexes ou ambigus

3.3. Infrastructure technique et environnement de développement

L'infrastructure technique nécessaire au projet doit être définie en tenant compte des contraintes de performance, de scalabilité, et de coût, tout en garantissant un environnement de développement efficace pour l'équipe.

3.3.1. Environnement de développement local vs cloud

Le choix entre un environnement de développement local et un environnement cloud dépend de plusieurs facteurs : la complexité de l'infrastructure, les besoins de collaboration, et les contraintes budgétaires.

Pour ce projet, nous abordons une approche hybride :

- **Développement local / On-premise** : Pour les phases de développement et de test initiales, permettant une itération rapide et une réduction des coûts
- **Environnement cloud (optionnel)** : Pour les phases de traitement intensif des données et les tests de performance, permettant de bénéficier de ressources scalables

3.3.2. Stack technique recommandée

- **Langage de programmation** : Python 3.10+ pour le traitement de données et l'intégration des API LLM (microservice), TypeScript/JavaScript pour le backend et frontend
- **Bibliothèques de traitement de données** : pandas, numpy pour la manipulation des données, et des bibliothèques spécialisées pour le NLP (spaCy, NLTK) pour le préprocessing
- **Framework web backend** : Express.js 5 avec TypeScript pour le développement de l'API REST, offrant une intégration simple avec les API LLM et RabbitMQ
- **Framework frontend** : Next.js 16 avec React 19 et TypeScript pour le développement du dashboard, avec la bibliothèque Recharts pour les visualisations interactives
- **Base de données** : MongoDB pour le stockage des métadonnées (utilisateurs, fichiers, tâches, logs, paramètres), offrant flexibilité et scalabilité
- **Message broker** : RabbitMQ pour la communication événementielle asynchrone entre les services
- **Orchestration de tâches** : Celery avec Python pour le traitement asynchrone des datasets
- **Outils de déploiement** : Docker et Docker Compose pour la containerisation et l'orchestration multi-services

3.3.3. Contraintes de performance et de scalabilité

Le traitement de 5 000 tweets nécessite une attention particulière aux performances, notamment pour garantir que le traitement puisse être effectué dans les délais impartis. Les principaux goulots d'étranglement identifiés sont :

- **Latence des API LLM** : Les temps de réponse des API peuvent varier de 1 à 5 secondes selon le modèle et la complexité de la requête
- **Limites de taux** : Les limites de taux imposées par les fournisseurs peuvent ralentir le traitement si elles ne sont pas gérées efficacement
- **Traitement séquentiel vs parallèle** : Le traitement peut être partiellement parallélisé en utilisant des appels asynchrones, mais les limites de taux doivent être respectées

Pour optimiser les performances, il est recommandé d'implémenter un système de traitement asynchrone avec gestion de la concurrence et respect des limites de taux, permettant de maximiser le débit tout en respectant les contraintes des API.

3.3.4. Architecture microservices

La solution adopte une architecture microservices pour garantir la scalabilité, la maintenabilité et la résilience du système. Cette architecture se compose de trois couches principales :

Couche Frontend : - **Frontend Next.js/React** : Interface utilisateur web moderne avec authentification JWT et contrôle d'accès basé sur les rôles (admin/analyste) - **RabbitMQ Client** : Connexion pour recevoir les mises à jour en temps réel des tâches de traitement

Couche Backend : - **Backend Express.js** : API REST pour la gestion des utilisateurs, fichiers, tâches, et paramètres - **MongoDB** : Base de données partagée pour le stockage des métadonnées (utilisateurs, fichiers, tâches, logs, paramètres) - **Stockage partagé** : Système de fichiers partagé pour les datasets (local/AWS/Azure)

Couche Microservices : - **Microservice Python/Celery** : Traitement asynchrone des datasets avec workers Celery - **RabbitMQ** : Message broker pour la communication événementielle entre les services - **LLM Service** : Intégration avec Ollama (local) ou API externes (OpenAI-compatible)

Communication entre services : - Le backend publie des événements `proceed_task` sur RabbitMQ pour déclencher le traitement - Le microservice écoute ces événements et traite les datasets de manière asynchrone - Le microservice publie des mises à jour de statut sur RabbitMQ (progression, erreurs, complétion) - Le frontend s'abonne aux événements RabbitMQ pour afficher les mises à jour en temps réel - MongoDB et le stockage sont partagés entre le backend et le microservice pour garantir la cohérence des données

Cette architecture permet une scalabilité horizontale (ajout de workers Celery), une résilience (redémarrage indépendant des services), et une maintenabilité (séparation des responsabilités).

3.3.5. Schéma de base de données MongoDB

La solution utilise **MongoDB** comme base de données principale pour le stockage des métadonnées et des informations de gestion. Le choix de MongoDB s'explique par plusieurs avantages adaptés aux besoins du projet :

- **Flexibilité du schéma** : Permet d'adapter facilement la structure des données aux évolutions du projet sans migrations complexes
- **Scalabilité horizontale** : Facilite l'ajout de nœuds pour gérer l'augmentation du volume de données
- **Intégration native avec Node.js/TypeScript** : Bibliothèques officielles performantes (Mongoose) pour le backend Express.js
- **Support des documents imbriqués** : Structure de données adaptée aux métadonnées complexes (paramètres IA, configurations)
- **Performance pour les lectures fréquentes** : Optimisé pour les opérations de consultation des métadonnées (fichiers, tâches, utilisateurs)

Le schéma de base de données est organisé en **six collections principales** dans la base de données dallosh_analysis :

Tableau 5 : Schéma de base de données MongoDB - Collections principales
{#tableau-5–schéma-de-base-de-données-mongodb—collections-principales}

Collection	Rôle	Champs principaux
users	Stocke les informations des comptes utilisateurs et l'authentification	uid, data.email, data.password (hashé bcryptjs), data.roleId (ref. roles), createdAt, updatedAt
roles	Définit les rôles et leurs permissions pour le contrôle d'accès basé sur les rôles (RBAC)	uid, data.name, data.description, data.permissions[], createdAt, updatedAt
files	Enregistre les métadonnées des fichiers CSV uploadés par les utilisateurs	uid, data.filename, data.size, data.file_path, data.extension, data.type, createdAt, updatedAt
tasks	Suit l'état et la progression des tâches de traitement des datasets par le microservice	uid, data.file_id (ref. files), data.file_path, data.status, data.file_cleaned, data.file_analysed, createdAt, updatedAt
logs	Enregistre les activités du système pour l'audit, le débogage et le suivi des opérations	uid, data.method (HTTP), data.path, data.response, data.requested_by, createdAt
settings	Contient les paramètres de configuration de l'application, notamment les préférences IA et de stockage	uid, data.general, data.ai.preferences, data.ai.local (Ollama), data.ai.external (Gemini, Mistral, OpenAI), data.storage, createdAt, updatedAt

4. Outils de suivi et de reporting

4.1. Outils de gestion des tâches et de collaboration

Le choix des outils de gestion des tâches et de collaboration est déterminant pour garantir une coordination efficace de l'équipe et un suivi précis de l'avancement du projet. Dans le contexte d'un projet Agile avec un délai contraint, ces outils doivent permettre une visibilité en temps réel sur l'état des tâches et faciliter la communication entre les membres de l'équipe.

4.1.1. Outil de gestion des tâches : Jira

Choix de **Jira** pour : intégration native Scrum, visibilité temps réel, traçabilité, intégrations (GitHub, Slack). Configuration : projet Scrum avec 3 sprints, épiques par phase, user stories, tâches techniques, tableaux Kanban.

4.1.2. Outils de communication

Slack : communication asynchrone par canaux, intégrations (Jira, GitHub), archivage. **Zoom/Teams** : visioconférence pour rituels Scrum.

4.2. Documentation et versioning

La documentation et le versioning du code sont essentiels pour garantir la traçabilité du projet, faciliter la maintenance, et permettre la transmission des connaissances au client.

4.2.1. Documentation : Jira

Choix de **Jira** pour : flexibilité modulaire, collaboration temps réel, intégrations (GitHub), accessibilité. Structure : documentation technique, utilisateur, notes de réunion, ADR.

4.2.2. Versioning : GitHub

GitHub est utilisé pour le versioning du code, offrant :

- **Gestion des branches** : Utilisation d'une stratégie Git Flow avec des branches pour les fonctionnalités, les corrections de bugs, et les releases
- **Pull requests** : Processus de revue de code permettant de valider les modifications avant intégration
- **Issues** : Suivi des bugs et des améliorations, intégré avec Jira pour une traçabilité complète
- **Actions CI/CD** : Automatisation des tests et des déploiements pour garantir la qualité du code

La structure du dépôt GitHub proposée inclut :

- **Branche principale** (main) : Code stable et testé
- **Branches de fonctionnalités** (feature/*) : Développement des nouvelles fonctionnalités
- **Branches de correction** (fix/*) : Corrections de bugs
- **Branches de release** (release/*) : Préparation des versions de livraison

4.3. Modes de reporting et communication

Le suivi de l'avancement du projet nécessite la mise en place de mécanismes de reporting réguliers permettant de communiquer l'état du projet aux différentes parties prenantes (équipe projet, client, management).

4.3.1. Daily Stand-up (15 minutes quotidiennes)

Les daily stand-up sont organisés chaque jour à heure fixe (par exemple, 9h00) et permettent à chaque membre de l'équipe de partager :

Tableau 6 : Points abordés lors du Daily Stand-up
{#tableau-6–points-abordés-lors-du-daily-stand-up}

Point	Description
Ce qui a été fait hier	Accomplissements de la veille
Ce qui sera fait aujourd'hui	Objectifs du jour
Les blocages éventuels	Problèmes rencontrés nécessitant une assistance

Les réunions sont brèves (15 minutes maximum) et visent à synchroniser l'équipe sans entrer dans les détails techniques, qui font l'objet de discussions séparées si nécessaire.

4.3.2. Sprint Review (1 heure en fin de sprint)

À la fin de chaque sprint, une sprint review est organisée

Tableau 7 : Objectifs de la Sprint Review {#tableau-7–objectifs-de-la-sprint-review}

Point	Description
Présenter les fonctionnalités développées	Démonstration des incréments livrés
Recueillir les retours	Feedback du Product Owner et des parties prenantes
Ajuster les priorités	Révision du backlog pour le sprint suivant

4.3.3. Sprint Retrospective (30 minutes en fin de sprint)

Tableau 8 : Points abordés lors de la Sprint Retrospective
{#tableau-8–points-abordés-lors-de-la-sprint-retrospective}

Point	Description
Identifier les points positifs	Ce qui a bien fonctionné pendant le sprint
Identifier les points d'amélioration	Ce qui pourrait être amélioré
Définir des actions correctives	Mesures concrètes à mettre en place pour le sprint suivant

4.3.4. Reporting hebdomadaire au client

Un rapport hebdomadaire est transmis au client Free chaque vendredi, incluant :

Tableau 9 : Contenu du rapport hebdomadaire au client
{#tableau-9–contenu-du-rapport-hebdomadaire-au-client}

Point	Description
État d'avancement	Pourcentage de complétion, jalons atteints
Risques identifiés	Problèmes potentiels et mesures d'atténuation
Décisions à valider	Choix techniques ou fonctionnels nécessitant une validation client
Planification de la semaine suivante	Objectifs et activités prévues

Ce reporting aura pour objectif de maintenir une transparence avec le client et d'anticiper les problèmes avant qu'ils ne deviennent critiques.

5. Contraintes techniques anticipées

5.1. Contraintes liées aux modèles LLM

L'utilisation des API LLM tierces introduit plusieurs contraintes techniques qui doivent être anticipées et intégrées dans la stratégie de pilotage pour éviter des blocages en cours de projet.

5.1.1. Quotas et limites de taux

Comme évoqué précédemment, les limites de taux imposées par les fournisseurs de LLM peuvent constituer un goulot d'étranglement significatif. Pour traiter 5 000 tweets dans les délais impartis, il est essentiel de :

- **Implémenter un système de file d'attente intelligent** : Gestion automatique des limites de taux avec mise en file d'attente des requêtes excédentaires
- **Parallélisation contrôlée** : Utilisation de requêtes asynchrones tout en respectant les limites de taux
- **Monitoring en temps réel** : Suivi du nombre de requêtes effectuées et du temps restant avant la réinitialisation des quotas

5.1.2. Latence et temps de réponse

Les temps de réponse des API LLM peuvent varier considérablement selon le modèle, la complexité de la requête, et la charge des serveurs du fournisseur. Cette variabilité peut affecter la planification du traitement. Pour gérer cette contrainte :

- **Estimation conservative** : Planification basée sur des temps de réponse moyens à élevés (3-5 secondes par requête) plutôt que sur les temps optimaux
- **Gestion des timeouts** : Implémentation de mécanismes de timeout et de retry pour gérer les requêtes qui prennent trop de temps
- **Traitement asynchrone** : Utilisation de requêtes asynchrones pour ne pas bloquer le traitement pendant l'attente des réponses

5.1.3. Coûts et optimisation budgétaire

Les coûts associés à l'utilisation des API LLM peuvent rapidement devenir significatifs, notamment si des modèles premium sont utilisés. Pour optimiser les coûts :

- **Choix du modèle adapté** : Utilisation de modèles moins coûteux pour les tâches simples (classification basique) et réservation des modèles premium pour les cas complexes
- **Optimisation des prompts** : Réduction de la longueur des prompts tout en conservant leur efficacité pour minimiser le nombre de tokens consommés
- **Mise en cache** : Stockage des résultats pour éviter les appels redondants en cas de retraitement ou de tests

5.1.4. Risques de disponibilité et de panne

Les API tierces peuvent être sujettes à des pannes ou des interruptions de service, ce qui peut bloquer complètement le traitement. Pour anticiper ce risque :

- **Stratégie de fallback** : Cascade de repli en cas de panne : Niveau 1 (API externes Gemini/Mistral/OpenAI) → Niveau 2 (Serveurs GPU internes avec Mistral Open Source via Ollama) → Niveau 3 (Mode dégradé avec modèles NLP classiques BERT/RoBERTa)
- **Monitoring de la disponibilité** : Mise en place d'un monitoring pour détecter rapidement les problèmes de disponibilité
- **Traitement par lots avec sauvegarde** : Sauvegarde régulière de l'état du traitement pour permettre une reprise en cas d'interruption

5.2. Défis du traitement des données brutes

Le traitement des 5 000 tweets bruts présente des défis techniques spécifiques liés à la nature hétérogène et non structurée des données textuelles issues des réseaux sociaux.

5.2.1. Hétérogénéité linguistique et variabilité

Les tweets présentent une grande variabilité linguistique (langage formel et informel, abréviations, emojis, fautes de frappe) qui peut affecter la qualité du traitement par les LLM. Pour gérer cette contrainte :

- **Préprocessing robuste** : Implémentation de techniques de nettoyage adaptées aux spécificités des tweets (normalisation des emojis, gestion des abréviations, correction des fautes de frappe courantes)
- **Validation de la qualité** : Mise en place de mécanismes de validation pour identifier les tweets qui nécessitent un traitement spécial ou une intervention manuelle
- **Tests sur échantillon représentatif** : Validation du pipeline de traitement sur un échantillon représentatif avant le traitement complet pour identifier les problèmes potentiels

5.2.2. Présence de bruit et de données non pertinentes

Une partie des tweets peut ne pas être directement pertinente pour le service client (retweets promotionnels, messages généraux, spams). Pour gérer cette contrainte :

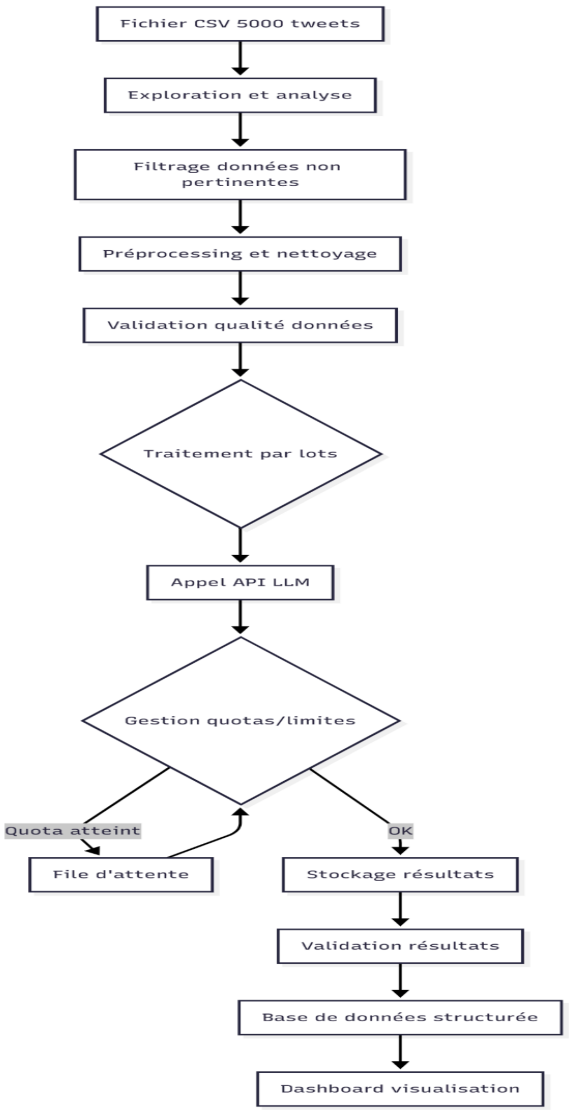
- **Filtrage préalable** : Implémentation de règles de filtrage pour exclure les tweets non pertinents avant le traitement par les LLM
- **Classification préliminaire** : Utilisation d'une classification simple (règles basées ou modèle léger) pour identifier les tweets à traiter en priorité
- **Validation manuelle optionnelle** : Possibilité de validation manuelle pour les cas limites ou douteux

5.2.3. Volume et performance

Le traitement de 5 000 tweets nécessite une attention particulière aux performances pour garantir que le traitement puisse être effectué dans les délais impartis. Pour optimiser les performances :

- **Traitement par lots optimisé** : Organisation du traitement en lots de taille optimale pour équilibrer la charge et respecter les limites de taux
- **Parallélisation intelligente** : Utilisation de la parallélisation pour accélérer le traitement tout en respectant les contraintes des API
- **Monitoring des performances** : Suivi en temps réel de la vitesse de traitement et identification des goulots d'étranglement

Figure 3 : Flux de traitement des données



5.3. Gestion des dépendances externes

La dépendance aux API tierces constitue un risque majeur pour le projet, nécessitant une stratégie de gestion proactive pour anticiper et gérer les problèmes potentiels.

5.3.1. Identification des points de dépendance

Les principaux points de dépendance identifiés sont :

- **Accès aux API** : Dépendance aux clés API et à leur validité
- **Disponibilité des services** : Dépendance à la disponibilité des serveurs des fournisseurs
- **Évolution des API** : Risque de modifications des API qui pourraient affecter le fonctionnement de la solution
- **Réglementation** : Risques liés aux évolutions réglementaires concernant l'utilisation des LLM ou le traitement des données personnelles

5.3.2. Stratégies d'atténuation

Pour atténuer les risques liés aux dépendances externes, plusieurs stratégies sont mises en place :

- **Diversification des fournisseurs** : Préparation d'une alternative avec un autre fournisseur de LLM pour permettre une migration rapide en cas de problème
- **Abstraction de l'intégration** : Implémentation d'une couche d'abstraction permettant de changer de fournisseur sans modifier l'ensemble du code
- **Monitoring proactif** : Mise en place d'un monitoring pour détecter rapidement les problèmes de disponibilité ou de performance
- **Documentation des dépendances** : Documentation claire des dépendances et des procédures de migration en cas de problème

Tableau 4 : Matrice des risques techniques

Risque	Probabilité	Impact	Criticité	Mesures d'atténuation
Indisponibilité API LLM	Moyenne	Élevé	Critique	Fournisseur alternatif, monitoring
Dépassement quotas	Élevée	Moyen	Élevée	Gestion file d'attente, estimation conservative
Coûts API supérieurs	Moyenne	Moyen	Moyenne	Optimisation prompts, choix modèles adaptés
Qualité données insuffisante	Faible	Élevé	Moyenne	Validation échantillon, préprocessing robuste
Latence API excessive	Moyenne	Moyen	Moyenne	Traitement asynchrone, estimation conservative
Évolution réglementaire	Faible	Élevé	Moyenne	Veille réglementaire, conformité RGPD

6. Conclusion

La stratégie de pilotage suivante s'adapte aux contraintes temporelles de 3 semaines, techniques avec les API LLM tierces et opérationnelles du projet d'automatisation du SAV de Free.

Le choix d'une méthodologie Agile (Scrum + Kanban) permet de structurer le travail en sprints courts et de s'adapter aux contraintes émergentes. La planification en cinq phases opérationnelles (préparation, traitement NLP, développement dashboard, tests, finalisation) offre une vision réaliste avec marges de sécurité pour les activités à risque élevé.

La mobilisation d'une équipe pluridisciplinaire de 4,1 ETP et l'utilisation d'outils de suivi comme Jira, GitHub et Slack garantissent une coordination efficace. L'anticipation des contraintes techniques causées par les API LLM, les données brutes et les dépendances externes; et la matrice des risques permettent de prioriser les actions et d'éviter les blocages.

La stratégie définie offre les fondations nécessaires pour mener à bien le projet dans les délais impartis. La réussite dépendra de la rigueur de mise en œuvre, de la qualité de la communication au sein de l'équipe, et de la capacité à anticiper et gérer les risques identifiés.

7. Annexes

Annexe 1 : Documentation API LLM

- OpenAI API Rate Limits : *https://platform.openai.com/docs/guides/rate-limits*
- Anthropic Claude API : *https://docs.anthropic.com/claude/reference*

Annexe 2 : Méthodologies Agile

- Scrum Guide 2020 : *https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-French.pdf*
- Kanban Method : *https://kanban.university/kanban-guide/*