



Institut Supérieur de Formation Continue d'Etterbeek  
**I.S.F.C.E.**

# Conception et développement d'une solution d'automatisation intelligente des courriels



*Valentin VANRUMBEKE*

Travail de Fin d'Étude présenté en vue de l'obtention  
du diplôme d'informatique orienté développement d'applications

Année académique 2024-2025

# Remerciements

Ma profonde gratitude est tout particulièrement adressée à Monsieur Wafflard, dont l'accompagnement et la pédagogie ont suscité en moi la passion pour la programmation tout au long de ces trois années à l'ISFCE. La clarté de ses cours et le suivi attentif dont j'ai bénéficié durant ce TFE ont été grandement appréciés. Son soutien et la confiance qu'il m'a accordés ont été déterminants dans la rédaction et le développement de ce travail. C'est avec le plus grand respect et la plus sincère reconnaissance que j'admets que ses enseignements ont conforté mon choix de faire de l'informatique ma voie.

Mes remerciements s'adressent également à l'ensemble des autres professeurs rencontrés au fil de ce cursus. À Monsieur VO, dont l'expertise et l'exigence ont transformé des cours d'abord intimidants en rendez-vous que j'attendais avec impatience pour perfectionner mes compétences : il m'a appris, dès la première année, que pour devenir un bon programmeur, il ne faut pas procrastiner mais programmer, inlassablement. À Monsieur Huguier, dont la passion contagieuse pour le réseau informatique a éveillé ma curiosité et mon intérêt pour une discipline qui, au départ, ne m'attirait guère.

Enfin, tous ceux qui œuvrent au sein de l'école, que j'aie pu côtoyer de près ou de loin, sont ici remerciés : c'est grâce à cet établissement qu'un nouveau départ m'a été offert après plusieurs années difficiles et sombres, et j'en serai éternellement reconnaissant.

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Objectifs du TFE . . . . .	5
1.2	Méthodologie . . . . .	6
1.3	Structure du document . . . . .	6
<b>2</b>	<b>Contexte</b>	<b>7</b>
2.1	Description de Lexlau . . . . .	7
2.2	Évolution de la demande . . . . .	8
2.3	Processus Lexlau . . . . .	9
<b>3</b>	<b>Description du sujet</b>	<b>10</b>
3.1	Qu'est-ce que MailFlow ? . . . . .	10
3.2	A qui sera destiné MailFlow ? . . . . .	12
<b>4</b>	<b>Analyse de l'existant</b>	<b>13</b>
4.1	Démarche d'analyse de l'existant . . . . .	13
4.2	Analyse comparée des solutions . . . . .	13
4.2.1	Gmelius . . . . .	13
4.2.2	SaneBox . . . . .	14
4.3	MailFlow face à l'existant . . . . .	15
<b>5</b>	<b>Exigences et besoins</b>	<b>17</b>
5.1	Diagramme de cas d'utilisation . . . . .	17
5.2	Exigences et besoins techniques, de sécurité et de performance . . . . .	19
5.2.1	Besoins techniques . . . . .	19
5.2.2	Besoins de sécurité . . . . .	19
5.2.3	Besoins de performance . . . . .	20
<b>6</b>	<b>Analyse</b>	<b>21</b>
6.1	Diagramme de composants . . . . .	21
6.2	Diagramme de classes . . . . .	23
6.3	Diagramme d'entités relationnelles . . . . .	25
6.4	Diagrammes de séquences . . . . .	27
6.4.1	Cas d'utilisation : <i>Éditer un flow</i> . . . . .	27
6.4.2	Cas d'utilisation : <i>Exécuter une tâche</i> . . . . .	29
6.4.3	Cas d'utilisation : <i>Analyser un mail</i> . . . . .	32
6.5	Diagramme d'activité . . . . .	35

---

<b>7 Conception</b>	<b>36</b>
7.1 APIs utilisées . . . . .	36
7.1.1 Unipile . . . . .	36
7.1.2 OpenAI . . . . .	37
7.2 Technologies front-end . . . . .	38
7.3 Technologies back-end . . . . .	39
7.4 Pourquoi ces choix ? . . . . .	40
<b>8 développement</b>	<b>41</b>
8.1 Architecture du moteur d'exécution . . . . .	41
8.1.1 Vue d'ensemble du processus d'exécution . . . . .	41
8.1.2 Boucle principale d'exécution . . . . .	44
8.1.3 Exécution des différents types de nœuds . . . . .	45
8.1.4 Finalisation de l'exécution . . . . .	48
8.2 Résumé du fonctionnement du moteur d'exécution . . . . .	48
8.3 IHM . . . . .	49
8.3.1 Zones fonctionnelles de l'interface . . . . .	50
8.4 Tests unitaires . . . . .	53
<b>9 Aspects financiers</b>	<b>54</b>
9.1 Coûts de développement . . . . .	54
<b>10 Conclusion</b>	<b>55</b>
10.1 Difficultés rencontrées . . . . .	55
10.2 Bilan . . . . .	56
10.3 Améliorations futures . . . . .	57
<b>Bibliographie</b>	<b>59</b>
<b>Glossaire</b>	<b>60</b>

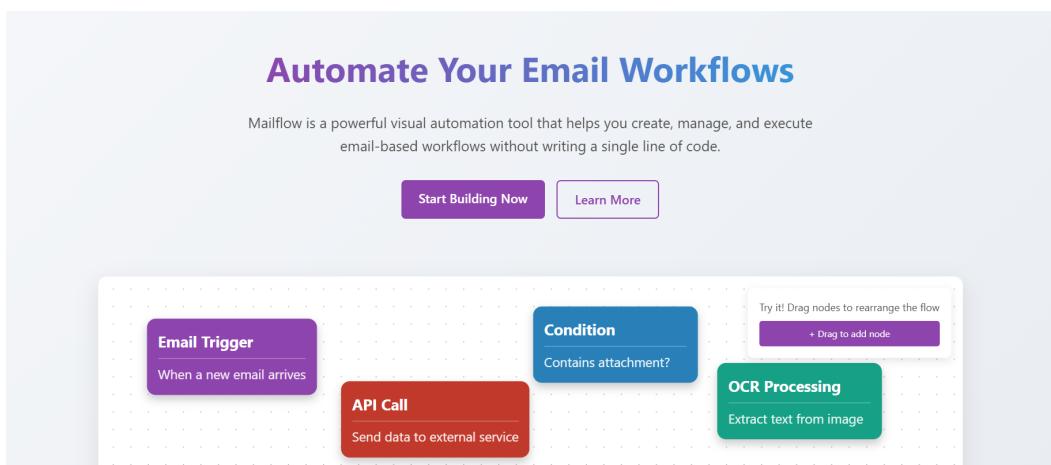
# 1 Introduction

À l'heure où les cabinets d'avocats traitent quotidiennement un volume croissant de courriels — demandes de suivi de dossier, pièces à fournir, convocations — l'absence de secrétariat dédié conduit souvent à des retards de traitement, une moindre satisfaction client et un risque d'erreurs administratif. Or, l'automatisation de ces tâches répétitives apparaît comme un levier clé pour améliorer la réactivité et la qualité de service des professionnels du droit.

## 1.1 Objectifs du TFE

Ce Travail de Fin d'Études a pour objectif de **concevoir**, **développer** et **valider** une solution d'automatisation intelligente des courriels, baptisée **MailFlow**, spécifiquement adaptée aux besoins d'un cabinet d'avocats. Plus précisément, il vise à :

- Analyser les processus de traitement des emails au sein du cabinet Lexlau et identifier les points de blocage.
- Concevoir une architecture modulaire, reposant sur un éditeur visuel de flux et un moteur d'exécution contextuel.
- Implémenter les fonctionnalités clés (routage conditionnel, extraction d'informations, envoi automatisé).
- Valider la solution au travers de tests fonctionnels et de retours d'expérience terrain, en mesurant ses performances et sa robustesse.



**Figure 1.1** – Screenshot de l'home page de MailFlow

## 1.2 Méthodologie

Pour atteindre ces objectifs, ce travail s'articule en quatre étapes :

- **Analyse des besoins** : entretiens avec les utilisateurs du cabinet, étude comparative de solutions existantes.
- **Conception** : spécifications fonctionnelles et techniques, modélisation UML et choix technologiques.
- **Implémentation** : développement de l'interface graphique, du moteur de traitement et intégration des API externes.
- **Validation** : mise en place de scénarios de test, profilage de performance et évaluation de la satisfaction utilisateurs.

## 1.3 Structure du document

Le corps de ce TFE commence par poser le décor du cabinet Lexlau et l'origine du besoin d'une automatisation des courriels, puis il plonge dans la présentation de MailFlow (l'application que j'ai développée pour ce tfe), ses objectifs et son mode de fonctionnement, avant de confronter notre proposition aux solutions existantes pour en dégager les points forts. Ensuite, je vous présenterai les besoins métier, techniques, de sécurité et de performance, puis ces exigences en une analyse précise à l'aide de diagrammes (composants, classes, séquences, activités). Fort de cette analyse, nous pourrons aborder les choix architecturaux (API, frameworks, bases de données) et décrire le développement du moteur d'exécution, de l'interface visuelle et des tests unitaires. Enfin, nous clôturons par une évaluation des coûts et des gains, des recherches de pistes d'évolution pour pérenniser la solution.

## 2 Contexte

### 2.1 Description de Lexlau

Le cabinet Lexlau, implanté à Ixelles, est spécialisé dans les procédures de visas pour ressortissants étrangers, avec une clientèle majoritairement originaire d'Afrique. Il est composé de trois membres : deux juristes et Maître Charles Epée, avocat. Durant la période de rentrée universitaire (de septembre à fin décembre), un afflux massif de dossiers est enregistré, car le cabinet pratique un contentieux de masse afin d'assurer sa rentabilité par le volume de dossiers traités.



**Figure 2.1** – Lexlau cabinet d'avocats  
[1]

Récemment créé, Lexlau a pour ambition d'informatiser et d'automatiser un maximum de tâches administratives afin de consacrer davantage de temps à l'accompagnement des clients. Aucun informaticien n'étant employé en interne, un prestataire externe est mandaté par Maître Epée pour épauler les stagiaires — sur qui repose une part importante du développement du CRM — et pour automatiser les principaux processus métiers, notamment la génération de documents juridiques à partir de modèles et la gestion des courriels.

À mon arrivée pour effectuer mon stage au sein du cabinet, la mission de poursuivre le développement du CRM m'a été confiée, ce dernier n'étant pas encore finalisé. Plusieurs groupes de stagiaires y avaient déjà contribué, et il m'a été demandé de prendre le relais de leurs travaux.

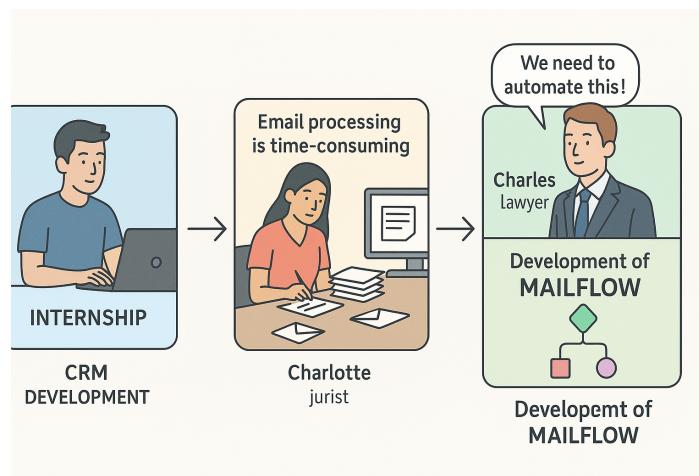
Au cours de ce travail sur le CRM, les besoins métiers du cabinet ont pu être identifiés et une familiarisation avec le code front-end et back-end a été acquise. Toutefois, au fil des discussions, la demande initiale de Maître Epée a évolué.

## 2.2 Évolution de la demande

L'idée de ce TFE est née peu après le début de mon stage au sein du cabinet d'avocats Lexlau, situé à Ixelles. J'avais initialement été affecté au développement du **CRM** du cabinet, déjà en partie conçu par une précédente équipe de stagiaires. Lors d'échanges plus approfondis avec mon maître de stage, il a été constaté que le traitement manuel des courriels représentait une perte de temps considérable, alors qu'un volume astronomique de messages est reçu chaque jour. Entre les demandes de suivi de dossier formulées par les clients, les documents juridiques relatifs aux affaires jugées, les convocations aux prochaines plaidoiries et les sollicitations de rendez-vous, l'afflux de courriels entraîne rapidement des retards et suscite un mécontentement général. De surcroît, aucune secrétaire n'étant employée, c'est la juriste Charlotte qui se charge chaque matin de leur traitement : une situation que Charles, avocat chez Lexlau, souhaite vivement résoudre.

C'est au cours de ces discussions que l'idée de développer un outil destiné à automatiser ce processus a été retenue : un outil accessible même à des utilisateurs dépourvus de compétences informatiques, et suffisamment générique pour pouvoir être déployé dans d'autres cabinets d'avocats, offrant ainsi un potentiel de valorisation financière.

Ainsi, à l'issue de ces échanges et de l'analyse des processus métier, le développement de MailFlow — objet de ce TFE — a été lancé afin de répondre précisément aux besoins du client.



**Figure 2.2** – illustration évolution de la demande (image générée par IA)

## 2.3 Processus Lexlau

Lorsque j'ai reçu la mission de développer une application d'automatisation des e-mails, on m'a également remis un document recensant tous les types de courriers que le cabinet doit traiter au quotidien. Ce référentiel m'a offert une vision transverse et claire des flux métier à modéliser dans MailFlow pour que l'outil soit réellement exploitable par rapport à l'API existante.

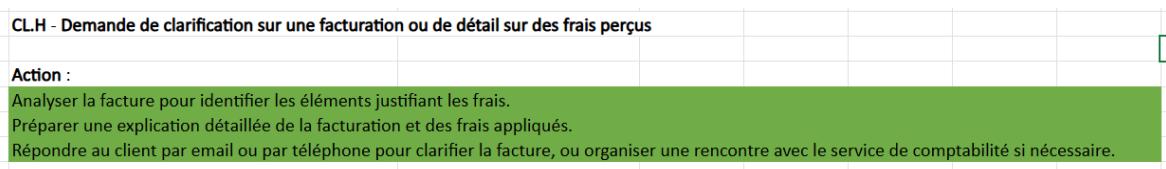


Figure 2.3 – processus analyse facture

Par exemple, en analysant le cas d'un mail de facture impayée, j'ai pu identifier quatre nœuds clés dès la conception du flow :

- **Récupération** de la pièce jointe : appeler l'API Unipile pour télécharger le document lié à la facture.
- **OCR** : lancer Tesseract.js via le nœud OCR pour convertir l'image en texte.
- **Analyse IA** : transmettre le texte obtenu à un service d'analyse (OpenAI ou futur module interne) pour extraire et structurer les informations métier.
- **Envoi de mail** : formater la réponse et poster un nouvel email via l'API Unipile pour répondre automatiquement.

Cette étude de cas m'a permis de structurer dès le départ une chaîne de traitement modulaire et réutilisable : chaque nœud réalise une tâche précise, et la boucle d'exécution parcourt tous les nœuds connectés jusqu'à épuisement, garantissant qu'on enchaîne proprement récupération, transcription, analyse et réponse automatique.

# 3 Description du sujet

## 3.1 Qu'est-ce que MailFlow ?

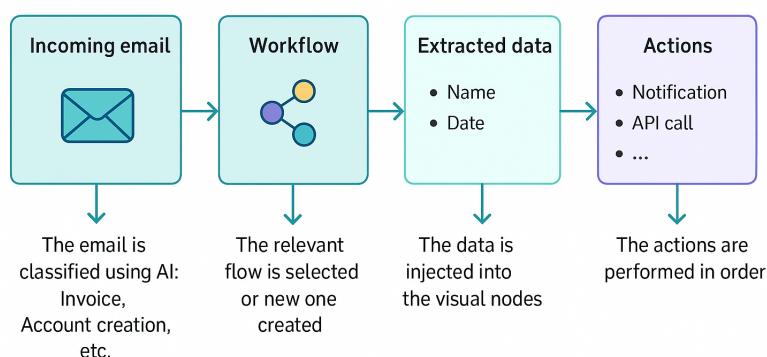
MailFlow est une plateforme d'automatisation de la gestion des courriels, qui transforme chaque message reçu en un enchaînement d'actions configurables visuellement. Concrètement, l'utilisateur construit son workflow en glissant-déposant des nœuds représentant :

- la classification automatique par l'IA (facture fournisseur, demande de création de compte, réclamation, etc.),
- les étapes de traitement (appel à une API, insertion de données en base, envoi de pièces jointes, notification d'un service...),
- et les conditions de passage d'un noeud à l'autre (tests sur le contenu ou les métadonnées du mail).

À l'arrivée d'un courriel, MailFlow :

- Le classe dans la catégorie correspondante grâce à son moteur d'IA.
- Sélectionne le flux associé, ou invite à en créer un nouveau si nécessaire.
- Injecte automatiquement les données extraites (nom, date, montant...) dans les nœuds prévus.
- Exécute chaque action selon l'ordre défini, de la simple notification à l'appel d'API tierce, sans intervention manuelle.

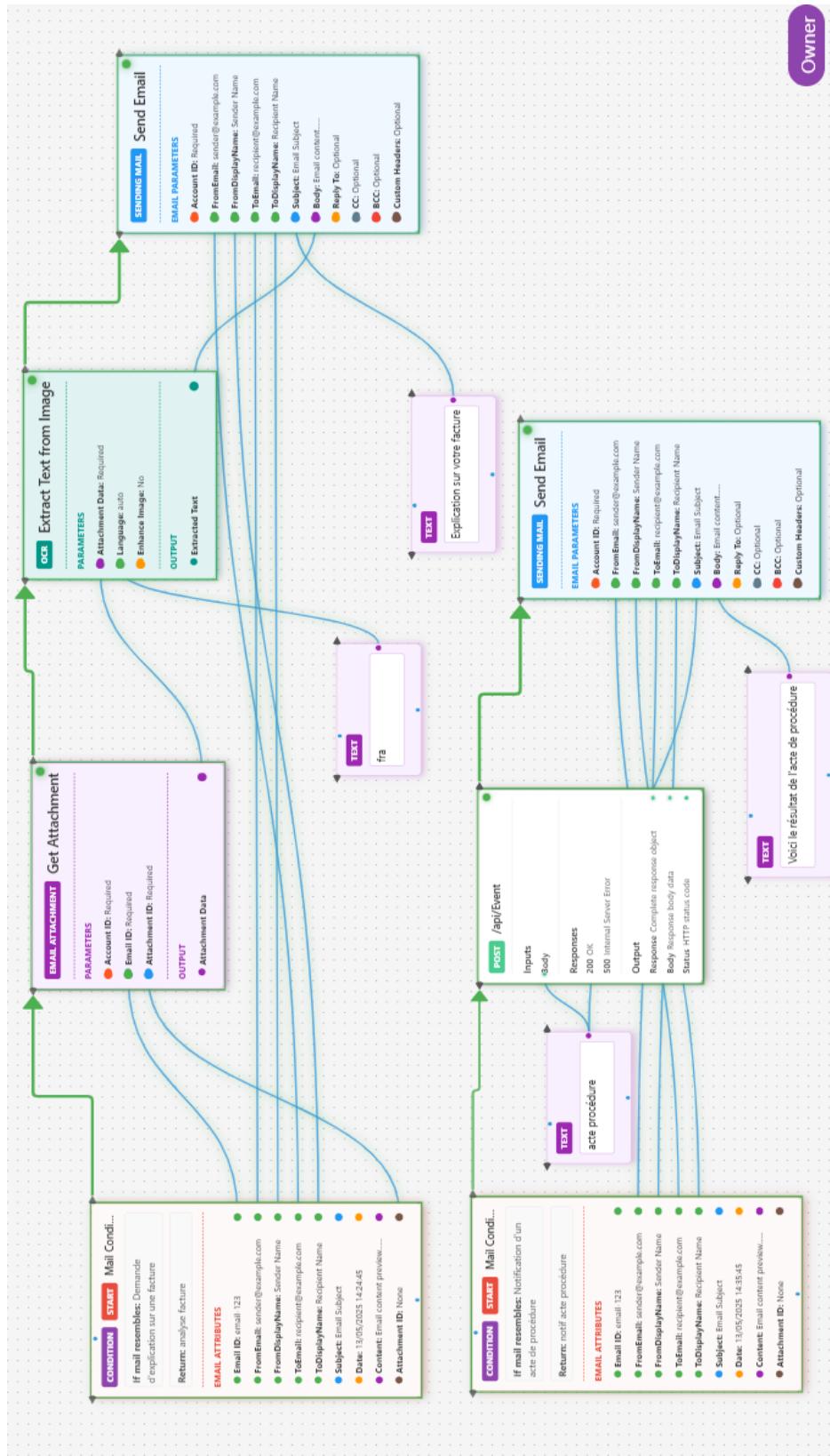
### MailFlow: no-code email automation



**Figure 3.1** – schéma explicatif MailFlow (Image générée par IA)

### 3.1. QU'EST-CE QUE MAILFLOW ?

11



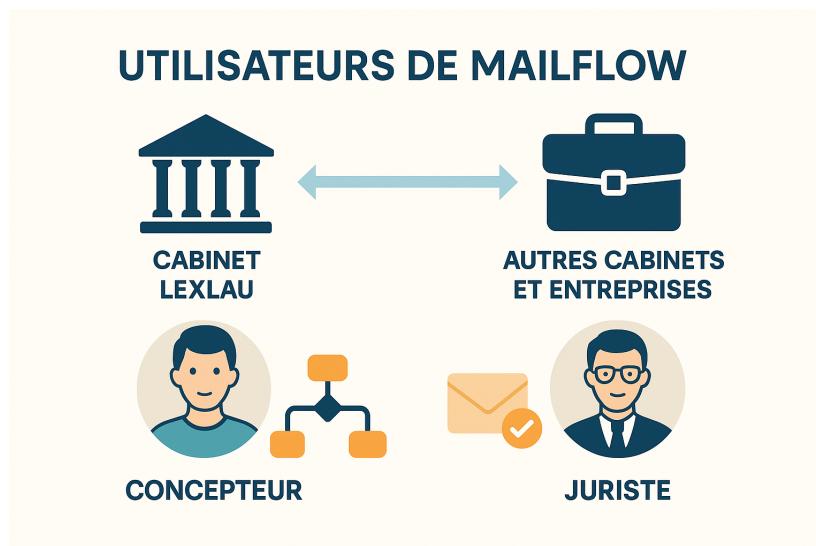
**Figure 3.2** – exemple de flux avec 2 points de départ

### 3.2 A qui sera destiné MailFlow ?

Cet outil a été conçu pour être déployé non seulement au sein du cabinet Lexlau, mais aussi commercialisé auprès d'autres cabinets et entreprises désireux d'automatiser l'intégralité de leur gestion de courriels. Maître Epée envisage en effet de l'exploiter en interne tout en tirant un bénéfice économique lors de sa commercialisation.

Parallèlement, il a été essentiel de déterminer les profils d'utilisateurs au sein du cabinet. C'est dans cette optique que le système de rôles de MailFlow a été établi : Chaque utilisateur ayant un niveau informatique élémentaire pourra structurer les workflows en traduisant les tâches métier appropriées—selon le type de courriel—sous forme de nœuds visuels, tandis que d'autres acteurs, tels que les juristes, n'auront qu'à déclencher l'exécution du processus MailFlow pour que l'ensemble du traitement des mails s'effectue automatiquement.

La section dédiée aux rôles, présentée ultérieurement dans ce TFE, précisera les droits et actions associés à chaque profil. Il convient néanmoins de souligner dès maintenant que tous les membres du cabinet pourront utiliser l'outil et bénéficier de son aspect collaboratif, chacun étant habilité à réaliser les opérations qui relèvent de son périmètre.



**Figure 3.3** – schéma explicatif cibles de MailFlow (image générée par IA)

# 4 Analyse de l'existant

## 4.1 Démarche d'analyse de l'existant

Pour évaluer les solutions de gestion et d'automatisation des courriels aujourd'hui disponibles en ligne, la documentation officielle de chaque outil a été passée en revue afin de recenser leurs fonctionnalités clés, leur mode d'intégration et leurs orientations technologiques. Ainsi, les sites de Gmelius et de SaneBox ont été consultés pour extraire les éléments de comparaison les plus pertinents. Les besoins spécifiques d'un cabinet d'avocats (volume élevé de mails, absence de secrétariat dédié, exigences de traçabilité et de flexibilité) ont ensuite servi de critères de mise en regard, afin d'identifier les atouts et les limites de chaque alternative face à MailFlow.

## 4.2 Analyse comparée des solutions

### 4.2.1 Gmelius



**Figure 4.1 – Gmelius logo**  
[2]

Il a été constaté que Gmelius transforme Gmail en plateforme collaborative grâce à des assistants IA capables de rédiger automatiquement des réponses, de trier les messages et de router les conversations vers les bons collaborateurs. Des règles d'automatisation visuelles permettent de définir des déclencheurs et des conditions (« si X, alors Y ») pour étiqueter, classer ou assigner les emails à un canal ou un label spécifique. Par

ailleurs, la conversion de labels Gmail en tableaux Kanban offre une vue structurée des processus de traitement des messages.

Pour un cabinet d'avocats, Gmelius présente l'avantage d'être intégré nativement à Google Workspace, sans infrastructure supplémentaire à déployer, et de proposer un éventail d'assistants intelligents (réponse, tri, dispatch) déjà éprouvé en entreprise.

En revanche, la solution reste limitée à l'écosystème Gmail et ne couvre pas directement les boîtes Exchange ou les clients de messagerie externes, ce qui constitue un frein pour les structures multi-plateformes.

#### 4.2.2 SaneBox



**Figure 4.2 – Sanebox logo**  
[3]

Il a été observé que SaneBox repose sur des algorithmes de machine learning pour prioriser automatiquement les emails entrants et les répartir dans des dossiers dédiés (SaneLater, SaneBlackHole...) selon leur importance. La fonctionnalité Email Organize permet de traiter en masse les messages déjà classés et d'affiner le filtrage par un apprentissage continu des règles utilisateur. Des rappels automatiques (« SaneReminders ») garantissent qu'aucune relance n'est oubliée, en programmant l'envoi de notifications ou de messages de suivi.

La faible configuration initiale (une simple connexion à la boîte) et la compatibilité universelle (Gmail, Outlook, Yahoo, etc.) sont particulièrement adaptées à des équipes sans ressources informatiques dédiées.

Toutefois, SaneBox ne propose aucune interface de flux visuel type noeuds à relier, ni de logique conditionnelle fine (ifs emboîtés, appels d'API externes), ce qui limite sa capacité à traiter des scénarios métier complexes.

## 4.3 MailFlow face à l'existant

MailFlow vise à combiner la souplesse d'une IA contextuelle et la rigueur d'une logique conditionnelle visuelle, le tout dans une interface modulable par nœuds sans la moindre ligne de code. Là où Gmelius dépend de l'écosystème Google et SaneBox se cantonne à un filtrage « boîte à lettres », MailFlow permettra :

- d'intégrer n'importe quelle API externe via des noeuds dédiés,
- de relier entre eux des déclencheurs et des actions en mode drag-and-drop,
- d'éduquer l'IA sur les types de mails et les réponses attendues, pour une prise de décision autonome,
- d'assurer une collaboration multi-utilisateurs avec droits d'accès granulaires,
- et de piloter l'ensemble sans développement additionnel au sein de toute structure, y compris multi-plateformes.



Figure 4.3 – Publicité MailFlow (image générée par IA)

**Table 4.1** – Comparaison des fonctionnalités

Fonctionnalité	MailFlow	SaneBox	Gmelius
Filtrage automatique	Règles conditionnelles et branchements personnalisés via éditeur de flux (ex. classer, router, exclure)	Filtre les messages peu importants dans SaneLater, SaneNews ou SaneBulk SaneBox	AI Sorting Assistant trie et classe automatiquement vos emails gmelius.com
Mise en veille / Rappel	Snooze et rappels paramétrables directement dans le flow	Snooze – mise en veille temporaire hors boîte de réception SaneBox	Follow-up Reminders – rappels de relance d'emails gmelius.com
Automatisation / Workflows	Éditeur visuel drag-&-drop (API calls, OCR, envoi de mails, extraction PJ...)	Non, focalisé sur le filtrage et les rappels	Automation Rules & Agents – règles d'automatisation avancées gmelius.com
Collaboration / boîtes partagées	Gestion des droits collaboratifs par flow et versioning	Non	Shared Inboxes & Shared Labels – boîtes et libellés partagés gmelius.com
Modèles & séquences	–	–	Email Templates & Sequences – modèles d'emails et campagnes de suivi gmelius.com
Intégrations & API	Appels API externes, <a href="#">OAuth2</a> , <a href="#">Webhook</a> ...	Fonctionne avec tout service mail IMAP/Exchange	Integrations & Channels – Slack, Trello, HubSpot, Zapier, etc. gmelius.com
Traitement pièces jointes & OCR	Extraction automatique, nœud OCR intégré	SaneAttachments – déplacement automatique des PJ vers dossier dédié SaneBox	–

# 5 Exigences et besoins

## 5.1 Diagramme de cas d'utilisation

Recueillir précisément les besoins était crucial pour assurer un développement fluide et une future commercialisation auprès d'autres cabinets. Plusieurs entretiens ont donc été menés avec Maître Epée puis étendus à d'autres avocats et juristes intéressés par l'automatisation des emails. À partir de ces échanges, un diagramme de cas d'utilisation non exhaustif a été élaboré, illustrant les principaux scénarios prévus d'ici à la soutenance.

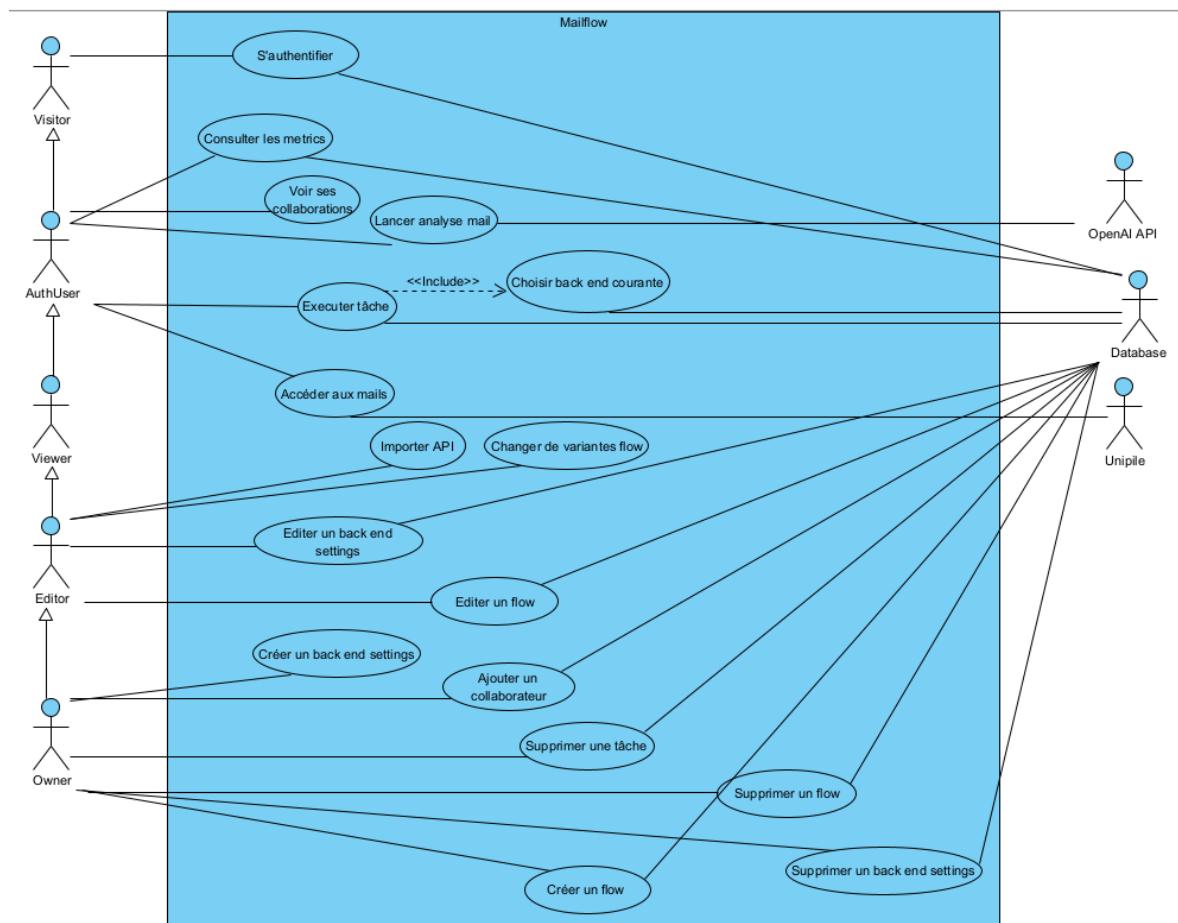


Figure 5.1 – Diagramme d'use case MailFlow

Il est intéressant de relever plusieurs points issus de ce diagramme, élaboré à la suite de multiples entretiens. Tout d'abord, l'accès à l'application est strictement réservé aux utilisateurs authentifiés : les visiteurs, appelés “visitors”, ne peuvent consulter que la page d'accueil.

Trois profils d'utilisateurs, héritant du rôle AuthUser, ont ensuite été définis :

- **Viewer** : ce rôle est conçu pour des acteurs en charge du traitement quotidien des emails (une secrétaire, par exemple). Seules l'analyse des courriels, l'exécution des tâches et la consultation des indicateurs (metrics) leur sont permises ; aucune création ou modification de flux n'est autorisée.
- **Editor** : toutes les prérogatives du Viewer leur sont ouvertes, ainsi que la possibilité de modifier des flux existants et leurs configurations back-end. La création de nouveaux flux reste cependant exclue. Ce rôle s'adresse notamment à des développeurs ou experts qui doivent adapter collectivement un même flux.
- **Owner** : le créateur d'un flux en devient automatiquement le propriétaire (Owner), avec tous les droits de lecture, modification et partage sur ce flux.

Le principe collaboratif de l'application repose sur un onglet dédié, permettant d'accéder aux flux partagés par d'autres utilisateurs. Selon le rôle qui leur a été attribué sur chaque flux, les collaborateurs pourront y exercer les actions permises : un Editor peut ainsi être Owner d'un premier flux et simple éditeur d'un second, et réciproquement. Enfin, chacun est libre de créer ses propres flux depuis l'éditeur visuel et d'y ajouter, à sa convenance, les personnes de son choix.



Figure 5.2 – bd humoristique sur les rôles maiflow (image générée par IA)

## 5.2 Exigences et besoins techniques, de sécurité et de performance

Afin d'assurer une mise en œuvre robuste, évolutive et fluide de MailFlow, trois catégories de besoins ont été définies :

### 5.2.1 Besoins techniques

- **Desktop uniquement** : l'interface n'est pas responsive, les opérations de création et de modification de flux se font sur poste fixe ou tablette au format desktop.
- **Stack front-end** : React + React Flow, gérant la visualisation et l'édition des noeuds via Hooks et Axios pour les appels REST.
- **Stack back-end** : Node.js + Express.js, orchestrant les routes et contrôleurs, et Mongoose pour modéliser les schémas MongoDB (documents Flow, sous-documents versions/noeuds/liaisons).

### 5.2.2 Besoins de sécurité

- **Authentification stateless** : JWT dans cookie HttpOnly, renouvelable, avec CORS strict (origines autorisées uniquement).
- **Contrôle d'accès** : RBAC (Visitor, Viewer, Editor, Owner) appliqué à chaque route via des middlewares hasRole et hasFlowAccess.
- **Validation et assainissement** : fonctions custom et express-validator pour filtrer et nettoyer toutes les entrées (prévenir XSS, injections, ID MongoDB invalides).
- **Chiffrement et hachage** : AES-256-GCM pour les données sensibles au repos, bcryptjs pour les mots de passe, gestion des clés via variables d'environnement.
- **Limitation de débit** : express-rate-limit sur les routes critiques (100 req/15 min/utilisateur).
- **Journalisation et gestion d'erreurs** : morgan en mode “combined”, Winston avec masquage des données sensibles, handlers global notFound et errorHandler.

### 5.2.3 Besoins de performance

- **Performances visuelles** : fluidité du drag-and-drop et du rendu React Flow même avec des centaines de noeuds ; optimisation du DOM virtuel et profilage régulier avec Chrome DevTools.
- **Performances applicatives** : code back-end optimisé (requêtes asynchrones, gestion des contextes d'exécution), architecture scalable (scaling horizontal, load balancing évoqués avec le client), et anticipation des pics de charge (notamment entre septembre et décembre chez Lexlau).
- **Profilage et monitoring** : enregistrement des interactions (DevTools) pour identifier et corriger les goulots d'étranglement, et mise en place future d'outils de supervision (Prometheus/Grafana).

En combinant ces besoins, MailFlow est conçu pour garantir :

- une expérience utilisateur fluide,
- une sécurisation à tous les niveaux (transport, stockage, accès),
- une scalabilité et une maintenabilité adaptées aux volumes élevés et aux exigences métier.

# 6 Analyse

Une part importante du travail a été consacrée à l'analyse et à la conception de diagrammes. Dès le lancement de la deuxième itération de l'application — la première n'ayant pas totalement répondu aux besoins du client — ces diagrammes ont été élaborés pour guider au mieux son développement.

Il convient de souligner que ces schémas reflètent une phase préliminaire de développement ; ils sont susceptibles d'évoluer et seront probablement ajustés avant la soutenance orale.

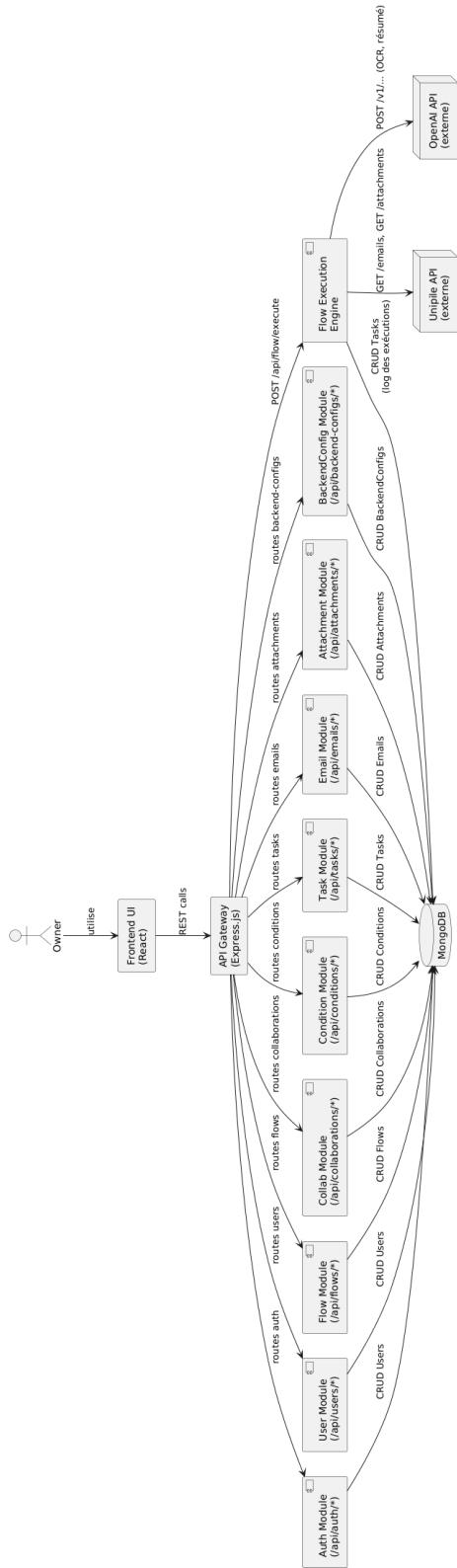
Une brève justification sera fournie à l'issue de chaque diagramme, afin d'expliciter la vision portée sur l'application.

## 6.1 Diagramme de composants

Ce diagramme de composants met en évidence la structure modulaire de l'application et facilite la compréhension de ses différents points d'entrée et flux de données. On y voit l'Owner qui utilise l'UI React, les appels REST qui transitent par la passerelle Express.js vers chacun des modules (auth, users, flows, tâches, emails, pièces jointes, etc.), et chaque module qui réalise ses opérations CRUD sur MongoDB.

Le moteur d'exécution de flows, déclenché via une route POST /api/flow/execute, crée ou met à jour une tâche en base, interroge l'API Unipile pour récupérer les emails et leurs pièces jointes, exécute ensuite la logique métier définie dans le flow, puis journalise l'ensemble des résultats dans MongoDB.

Cette vue d'ensemble permet de repérer rapidement les responsabilités de chaque composant, leurs dépendances internes et externes, et sert de référence pour la maintenance, l'optimisation ou l'extension du système.



**Figure 6.1 – diagramme de composants MailFlow**

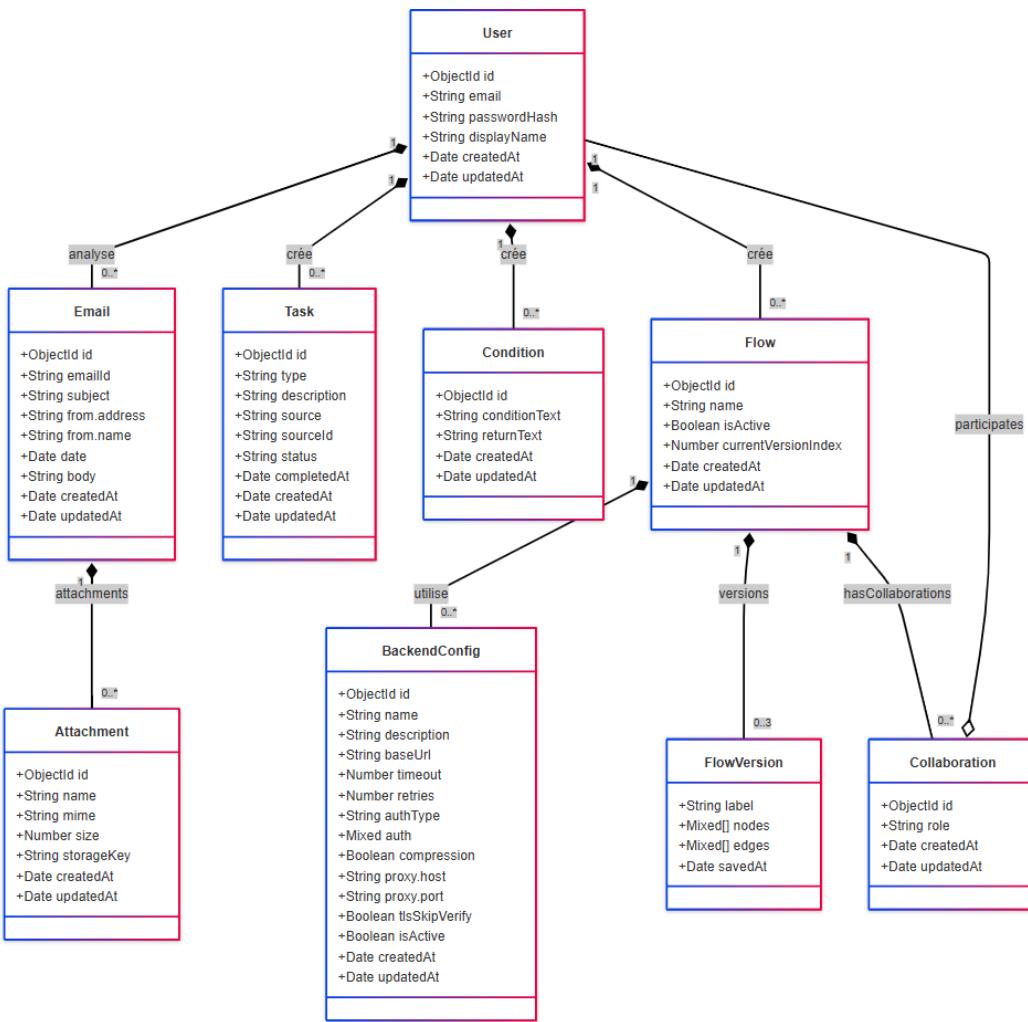
## 6.2 Diagramme de classes

Dans cette version du diagramme de classes, l'utilisateur est défini comme racine de nombreuses compositions, afin d'éviter de conserver en base de données les flows, conditions ou tâches associés à un compte supprimé. Un dilemme s'est toutefois présenté : il semblerait plus cohérent que ces classes soient intégrées par composition au flux (flow) lui-même, ce qui simplifierait la collaboration — par exemple, un utilisateur en rôle Viewer sur le flow d'un tiers pourrait alors accéder directement aux conditions et aux tâches de ce même flow.

Il a également été envisagé de lier chaque condition à la fois à un flow et à son auteur, de manière à préserver l'historique du créateur tout en garantissant qu'une suppression (du flow ou de l'utilisateur) entraîne automatiquement la suppression en cascade de la condition concernée.

À ce jour, aucune décision définitive n'a encore été tranchée, mais la structure générale et les cardinalités resteront inchangées quelle que soit l'option retenue.

- **User** : un compte qui crée et gère des flows, emails, tâches et conditions.
- **Email** : un message stocké (métadonnées + contenu) issu de l'API ou envoyé.
- **Attachment** : une pièce jointe liée à un email, stockée via une clé de stockage.
- **Task** : le suivi d'exécution d'un flow déclenché par un email ou manuel.
- **Condition** : un nœud de démarrage filtrant les emails selon un critère précis.
- **Flow** : la définition graphique d'un processus versionné, associée à sa config backend.
- **BackendConfig** : les paramètres d'URL, d'authent et de timeout pour appeler les API
- **FlowVersion** : un instantané du graph (nœuds + liens) d'un flow à un moment donné.
- **Collaboration** : le rôle (owner/editor/viewer) d'un user sur un flow partagé.

**Figure 6.2** – diagramme de classes MailFlow

## 6.3 Diagramme d'entités relationnelles

Cet ERD met en évidence les champs de premier niveau de chaque collection MongoDB, tels qu'ils sont explicitement déclarés dans les appels à new Schema( ... ). Les sous-documents imbriqués — en particulier les nodes et edges du schéma Flow — n'apparaissent pas comme des colonnes indépendantes : ils sont contenus dans le tableau versions, lui-même stocké au sein de chaque document Flow. Les outils de modélisation relationnelle (comme dbdiagram.io) ne « déplient » en général que les attributs scalaires et les clés étrangères, sans détailler les structures internes de Mongoose.

Cette organisation a été retenue délibérément, car elle correspond étroitement à la logique métier et aux exigences de performance du projet :

Les FlowVersion restent fortement liées à leur Flow parent, garantissant que chaque version de diagramme est gérée en même temps que le Flow lui-même — ce qui assure l'atomicité des mises à jour (une seule opération suffit pour modifier un Flow et toutes ses versions).

Le fait d'embarquer nodes et edges dans un même document, plutôt que de les répartir en collections distinctes, simplifie le code et limite le besoin de jointures coûteuses.

Les opérations de lecture et d'écriture sont optimisées, puisque MongoDB gère plus efficacement les accès à un document unique qu'à plusieurs collections reliées.

Le modèle reflète fidèlement la nature encapsulée d'une version de flux : un ensemble de noeuds et de liaisons n'a pas de sens en dehors du contexte de son Flow.

En conservant cette structure, la base de données reste cohérente, facile à maintenir et parfaitement adaptée à l'écosystème MongoDB, tout en offrant la souplesse nécessaire pour faire évoluer les schémas au fil du développement.

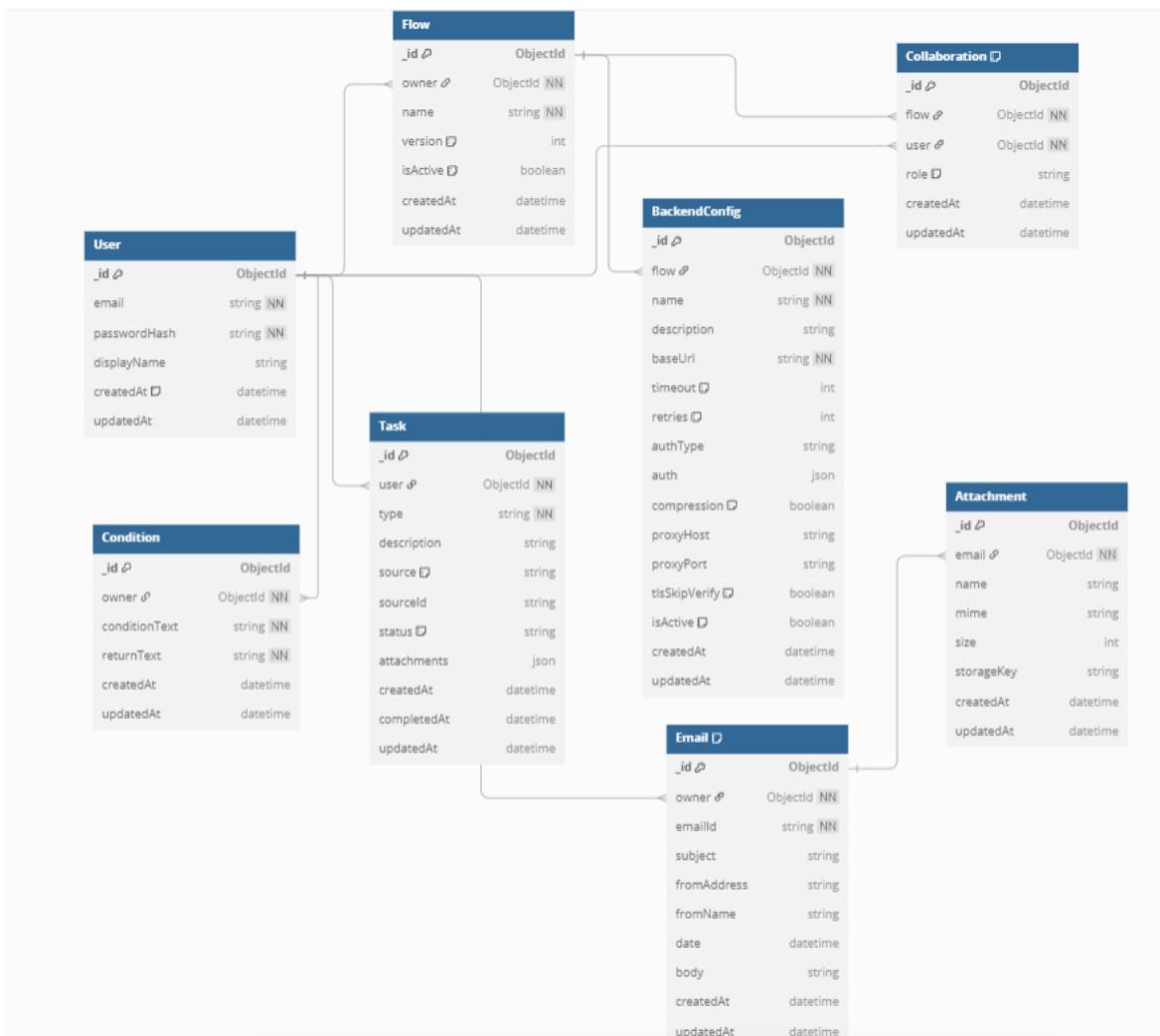


Figure 6.3 – diagramme d’ERD MailFlow

## 6.4 Diagrammes de séquences

Les diagrammes de séquence des cas d'utilisation critiques de l'application sont présentés ci-dessous. Par souci de concision, tous les scénarios ne sont pas illustrés ; seuls ceux jugés prioritaires ont été retenus. Il convient de souligner qu'il s'agit d'une vision en phase de développement préliminaire, et que les schémas seront susceptibles d'évoluer d'ici à la soutenance orale.

### 6.4.1 Cas d'utilisation : *Éditer un flow*

#### *Résumé*

L'utilisateur authentifié (rôle **Owner** ou **Editor**) peut ouvrir un flow existant, modifier sa structure (ajout / suppression de noeuds et de liens) puis sauvegarder la version courante.

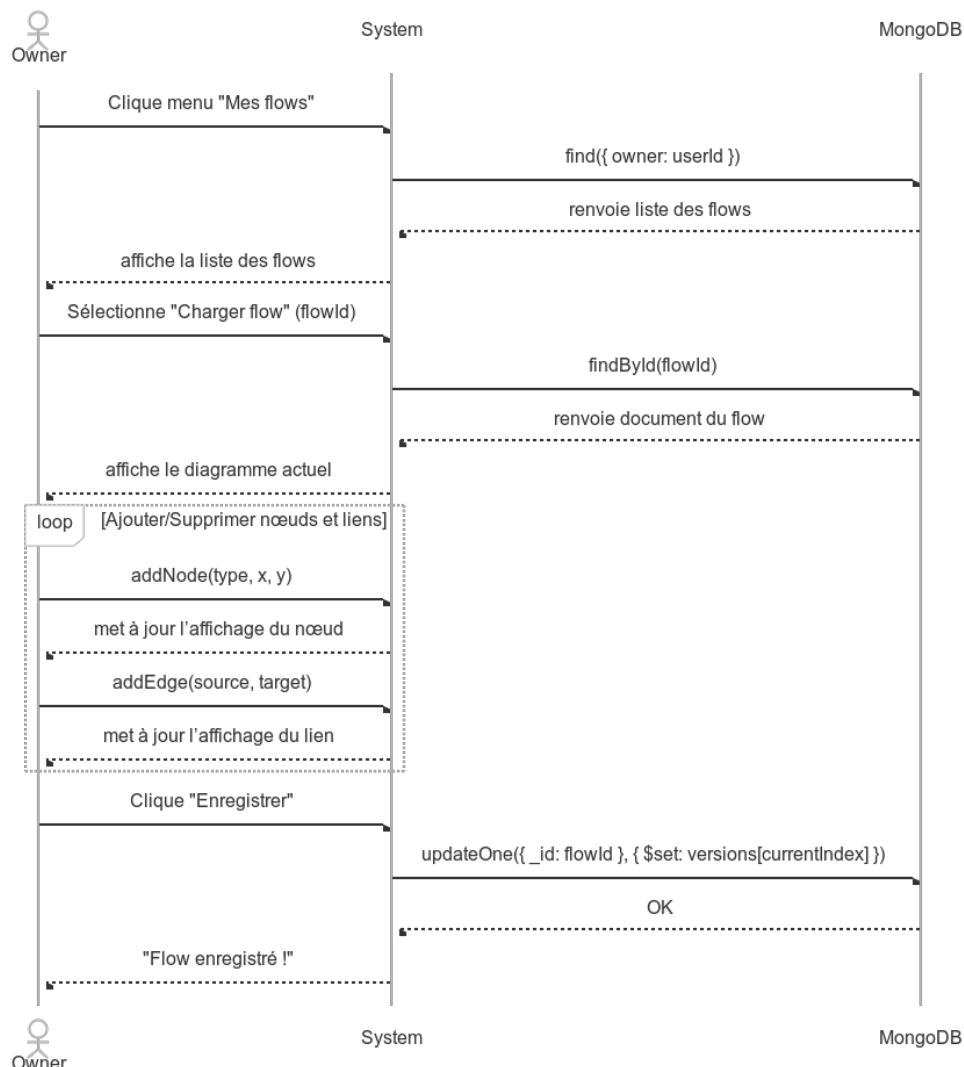
**Table 6.1** – Rôles et descriptions

Rôle	Description
Principal (Owner)	Peut modifier un flow existant
Secondaire (Système)	Composant back-end
Secondaire (MongoDB)	Base de données

#### *Préconditions*

- L'utilisateur est connecté et possède au moins le rôle **Editor** sur le flow.
- Le flow à modifier existe en base (identifiant valide).

#### *Scénario nominal*



**Figure 6.4 – Diagramme du scénario nominal d'édition**

### *Scénarios alternatifs*

#### A1 : Flow introuvable

Si `flowId` n'existe pas, MongoDB renvoie une erreur. Le système affiche « Flow non trouvé » et revient à la liste des flows.

#### A2 : Droits insuffisants

Si l'utilisateur n'a pas le rôle *Editor/Owner*, le système renvoie HTTP 403. Le système affiche « Accès refusé : droits insuffisants » et revient à la liste des flows.

### *Scénario d'erreur*

#### E1 : Erreur technique

En cas de timeout ou d'indisponibilité de MongoDB, le système affiche « Erreur serveur, veuillez réessayer plus tard. » et termine le cas d'utilisation en échec.

## 6.4.2 Cas d'utilisation : *Exécuter une tâche*

### *Résumé*

L'utilisateur authentifié (rôle **Owner**) peut exécuter une tâche existante, en spécifiant éventuellement une configuration particulière pour cette exécution.

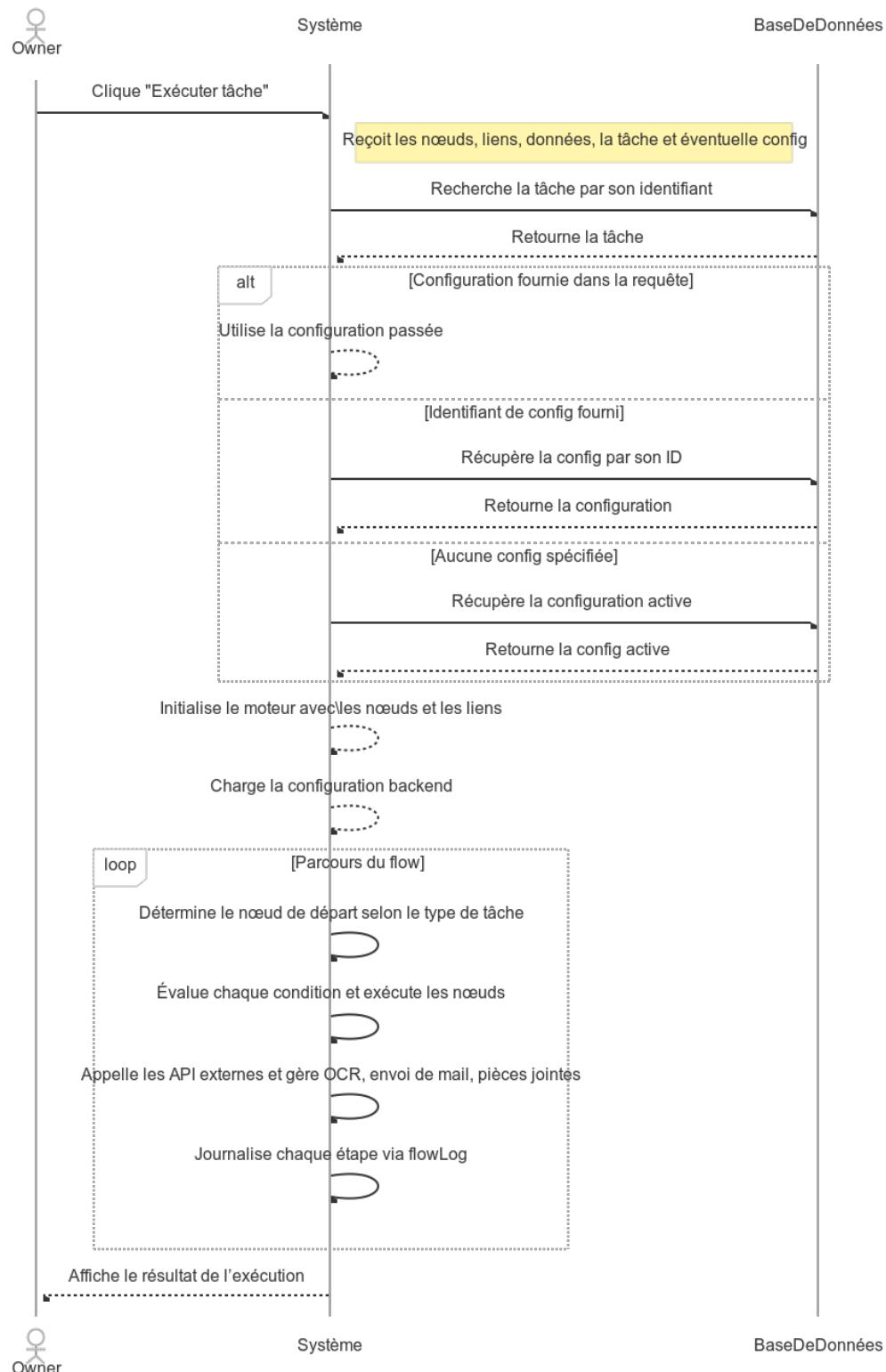
**Table 6.2** – Rôles et descriptions pour l'exécution d'une tâche

Rôle	Description
Principal (Owner)	Peut exécuter une tâche
Secondaire (Système)	Composant back-end qui gère l'exécution
Secondaire (BaseDeDonnées)	Base de données stockant les tâches et configurations

### *Préconditions*

- L'utilisateur est connecté et possède le rôle **Owner**.
- La tâche à exécuter existe en base (identifiant valide).
- Les noeuds et liens nécessaires à l'exécution sont disponibles.

### *Scénario nominal*



**Figure 6.5 – Diagramme du scénario nominal d'exécution d'une tâche**

*Scénarios alternatifs***A1 : Configuration fournie dans la requête**

Si une configuration est fournie dans la requête, le système l'utilise directement.

**A2 : Identifiant de configuration fourni**

Si un identifiant de configuration est fourni, le système récupère cette configuration spécifique.

**A3 : Aucune configuration spécifiée**

Si aucune configuration n'est spécifiée, le système récupère la configuration active par défaut.

*Scénario d'erreur***E1 : Tâche introuvable**

Si l'identifiant de la tâche n'existe pas, le système affiche « Tâche non trouvée » et termine le cas d'utilisation en échec.

**E2 : Configuration invalide**

Si la configuration spécifiée est invalide ou incompatible avec la tâche, le système affiche « Configuration invalide » et termine le cas d'utilisation en échec.

**E3 : Erreur d'exécution**

En cas d'erreur pendant l'exécution (API externe indisponible, erreur de traitement), le système journalise l'erreur et affiche un message approprié à l'utilisateur.

### 6.4.3 Cas d'utilisation : *Analyser un mail*

#### *Résumé*

L'utilisateur authentifié (rôle **Owner**) peut consulter ses emails et sélectionner un message spécifique pour analyse. Le système extrait les informations pertinentes, les analyse via l'IA et crée automatiquement une tâche associée.

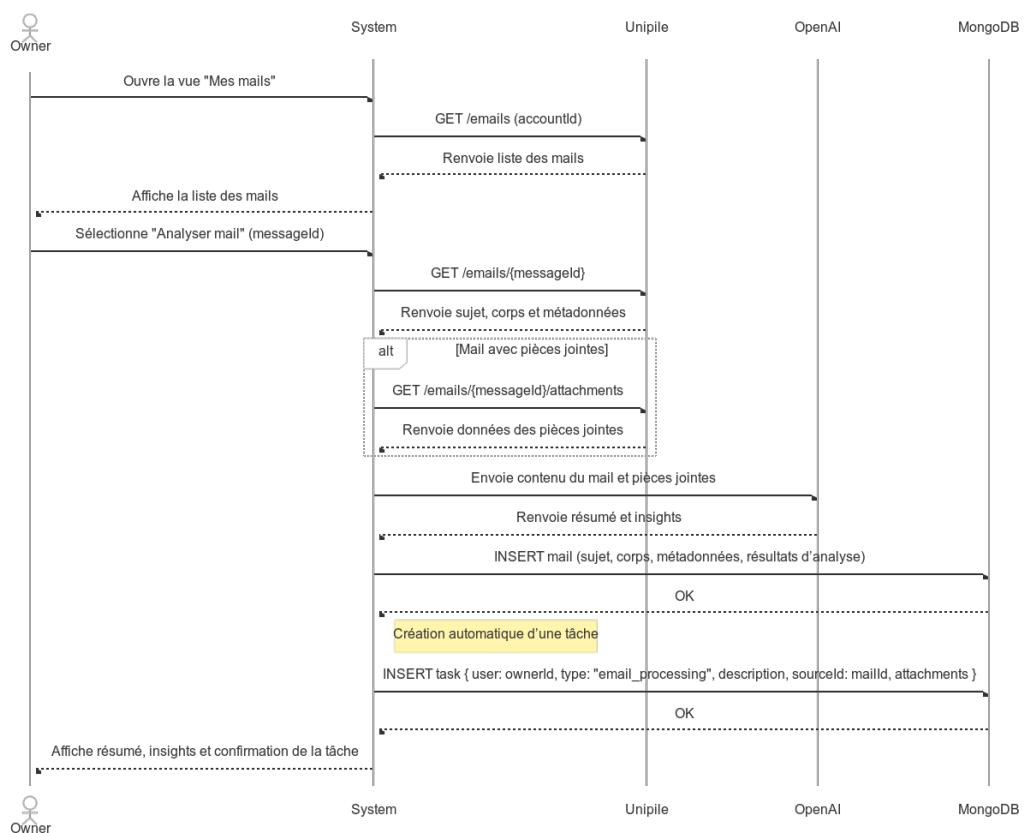
**Table 6.3** – Rôles et descriptions pour l'analyse d'un mail

Rôle	Description
Principal (Owner)	Utilisateur qui consulte et sélectionne un email pour analyse
Secondaire (System)	Composant back-end qui gère l'interface et les requêtes
Secondaire (Unipile)	Service d'accès unifié aux emails
Secondaire ( <a href="#">OpenAI</a> )	Service d'intelligence artificielle pour l'analyse
Secondaire (MongoDB)	Base de données stockant les emails et tâches

#### *Préconditions*

- L'utilisateur est connecté et possède le rôle **Owner**.
- Le compte email de l'utilisateur est correctement configuré dans le système.
- Les services Unipile et OpenAI sont disponibles.

#### *Scénario nominal*



**Figure 6.6 – Diagramme du scénario nominal d’analyse d’un mail**

*Scénarios alternatifs***A1 : Mail avec pièces jointes**

Si le mail contient des pièces jointes, le système effectue une requête supplémentaire pour récupérer ces fichiers. Le système récupère les données des pièces jointes via Unipile. Ces pièces jointes sont incluses dans l'analyse et associées à la tâche créée.

*Scénarios d'erreur***E1 : Échec de récupération des emails**

Si Unipile ne parvient pas à récupérer la liste des emails (timeout, erreur d'authentification), le système affiche « Impossible de récupérer vos emails, veuillez réessayer plus tard. » et termine le cas d'utilisation en échec.

**E2 : Échec de récupération d'un email spécifique**

Si Unipile ne parvient pas à récupérer les détails d'un email spécifique, le système affiche « Impossible de récupérer les détails de cet email. » et revient à la liste des emails.

**E3 : Échec d'analyse par l'IA**

Si OpenAI rencontre une erreur lors de l'analyse du contenu, le système affiche « L'analyse automatique a échoué. Veuillez réessayer ultérieurement. » et propose à l'utilisateur de créer manuellement une tâche.

**E4 : Échec de création de tâche**

Si la création de la tâche dans MongoDB échoue, le système journalise l'erreur, sauvegarde l'analyse effectuée, et affiche « La création de tâche a échoué. L'analyse a été sauvegardée. »

## 6.5 Diagramme d'activité

Une vue simplifiée du moteur d'exécution des flux est présentée par ce diagramme d'activité. À partir d'une condition initiale — ici désignée « autre » — l'algorithme est amené à parcourir séquentiellement l'ensemble des nœuds d'un flow pour en assurer le traitement du courriel.

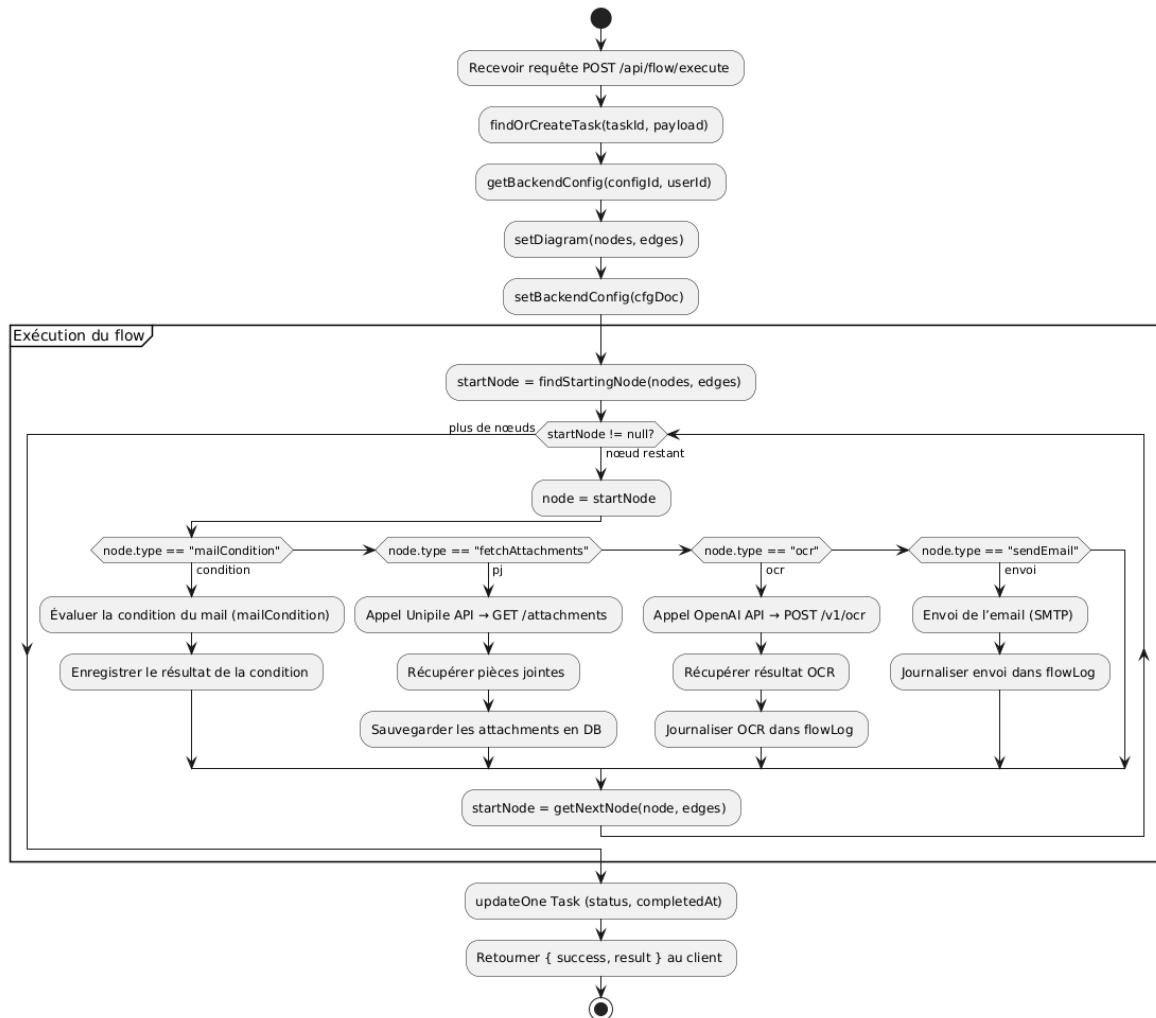


Figure 6.7 – diagramme d'activité MailFlow

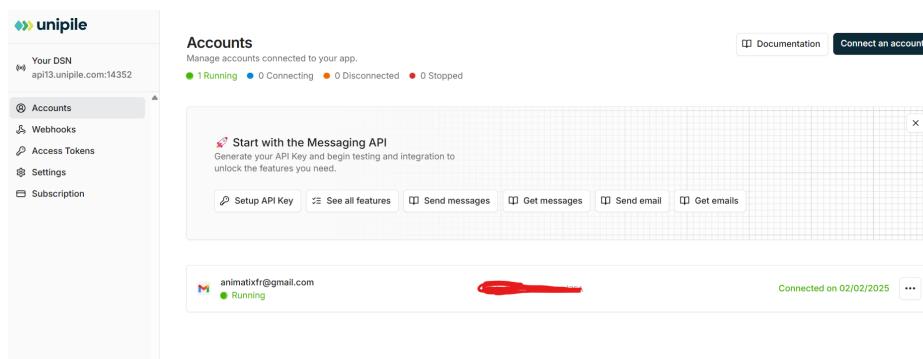
# 7 Conception

## 7.1 APIs utilisées

### 7.1.1 Unipile

Dans le cadre du développement de MailFlow, nous avons fait le choix d'utiliser Unipile comme solution d'accès aux emails. Il est important de préciser que cette décision a été prise principalement pour des raisons pragmatiques liées aux contraintes de temps du projet. Unipile offre une API unifiée permettant d'accéder à différents fournisseurs de messagerie (Gmail, Outlook, etc.) via une interface standardisée, ce qui nous a permis de nous concentrer sur le développement des fonctionnalités métier plutôt que sur l'intégration technique avec chaque fournisseur d'email.

Cependant, il convient de souligner que l'utilisation d'Unipile est considérée comme une solution temporaire. Pour la version de production de MailFlow, il est prévu de développer une solution interne dédiée à la récupération et à l'envoi de courriels. Cette approche nous permettra de mieux contrôler les coûts, d'optimiser les performances et d'assurer une meilleure intégration avec les autres composants du système. La transition vers cette solution interne fait partie des évolutions planifiées pour garantir la viabilité économique et technique du produit à long terme.



**Figure 7.1 – Interface Unipile**  
[4]

### 7.1.2 OpenAI

L'intégration de l'API OpenAI constitue un élément central de l'architecture de MailFlow, apportant des capacités d'intelligence artificielle essentielles au fonctionnement du système. Cette API est utilisée principalement dans deux contextes clés :

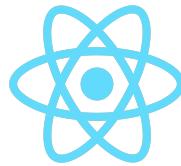
- **Triage automatique des emails** : L'API OpenAI analyse le contenu des emails entrants (objet, corps, expéditeur) pour les catégoriser automatiquement selon les types prédéfinis par l'utilisateur. Cette classification intelligente permet d'orienter chaque email vers le flux de traitement approprié sans intervention humaine.
- **Noeuds IA spécialisés** : Certains noeuds du système exploitent directement les capacités de l'API OpenAI pour des tâches spécifiques, comme l'extraction d'informations structurées à partir du texte d'un email, la génération de réponses contextuelles, ou l'analyse de sentiment pour adapter le traitement selon la tonalité du message.

Contrairement à Unipile, l'utilisation de l'API OpenAI est considérée comme pérenne dans l'architecture de MailFlow. Les modèles d'IA d'OpenAI offrent des performances de pointe en matière de compréhension du langage naturel, ce qui est essentiel pour assurer la précision du triage et la pertinence des traitements automatisés. Dans la version commerciale, les clients seront responsables de la gestion de leurs propres crédits OpenAI, ce qui permettra d'optimiser les coûts tout en maintenant l'accès à ces fonctionnalités avancées.

L'intégration d'OpenAI s'inscrit dans une stratégie plus large visant à exploiter les technologies d'IA pour automatiser intelligemment les processus de gestion des emails, tout en offrant suffisamment de flexibilité pour que les utilisateurs puissent configurer et affiner le comportement du système selon leurs besoins spécifiques.

## 7.2 Technologies front-end

**React** : on utilise React pour construire une interface très réactive et modulable. Chaque écran est découpé en composants réutilisables, ce qui facilite la maintenance et l'évolution de l'UI.



**Figure 7.2 – Logo React**  
[5]

**React Flow** : pour l'édition et la visualisation de vos "flows" (nœuds et liens), nous avons choisi React Flow, une librairie spécialisée qui offre du drag-and-drop, la gestion des positions, des styles et des interactions sur les graphes. Elle s'intègre parfaitement avec React et stocke le diagramme sous forme de tableaux JSON (nodes, edges), ce qui simplifie la sérialisation et le passage au back-end.

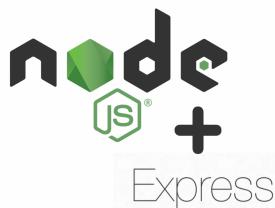
Communication avec le back-end via des appels REST (fetch/axios) vers un API Gateway.



**Figure 7.3 – Logo React flow**  
[6]

## 7.3 Technologies back-end

**Node.js + Express.js** : nous avons choisi Express pour sa légèreté et son écosystème riche (middlewares, routing clair, compatibilité avec l'outil JS). Les routes sont organisées en modules : auth, users, flows, collaborations, conditions, tasks, emails, attachments, backend-configs...



**Figure 7.4 – Logo NodeJS et Express**  
[7]

**Mongoose comme ODM pour MongoDB :**

- Définition de schémas (avec sous-documents pour les versions de flow) et de validations (longueur max, valeurs par défaut, timestamps).
- Gestion automatique des champs createdAt/updatedAt, et transformation JSON via .toJSON() (suppression de \_id et renommage en id).
- Transactions atomiques et hooks si besoin (hooks pre/post), ce qui renforce la robustesse.

**MongoDB comme base de données** : document-oriented, idéale pour stocker des structures de diagrammes (nodes, edges) et des logs de tâches. Son modèle schéma-libre permet d'évoluer facilement, et en mode cloud (Atlas) il apporte haute disponibilité et montée en charge.



**Figure 7.5 – Logo MongoDB**  
[8]

## 7.4 Pourquoi ces choix ?

React + React Flow assurent une expérience utilisateur fluide et un outil d'édition de graphes ultra-paramétrable sans réinventer la roue.

Express.js est suffisamment minimal pour ne pas imposer une surcouche lourde, tout en disposant d'un vaste écosystème de middlewares (authentification, CORS, validation...).

Mongoose + MongoDB permettent de modéliser directement en JavaScript des documents JSON complexes (versions de flow, logs, tâches), d'appliquer des contraintes/schemas tout en gardant la flexibilité nécessaire pour faire évoluer le modèle sans migrations fastidieuses.

Enfin, l'ensemble respecte la séparation des responsabilités : le front gère l'affichage et l'interaction, le back-end expose une API REST simple et le moteur d'exécution orchestre la logique métier lourde (flux, appels externes, journalisation).

Pour ce projet, seul l'IDE Visual Studio Code a été employé, par habitude et en raison de sa simplicité d'utilisation.

# 8 développement

## 8.1 Architecture du moteur d'exécution

Le fonctionnement du moteur d'exécution des flux, ainsi que son algorithme permettant de passer d'un nœud à l'autre en respectant l'enchaînement défini et en renseignant automatiquement les paramètres requis, sera détaillé ici. Étant donné la longueur du TFE, seule cette portion du code sera exposée, à l'aide d'un flux de quatre nœuds, afin d'illustrer la démarche de conception précédemment analysée (moteur d'exécution côté back-end, routes Express.js de pilotage, intégration d'Unipile, etc.).

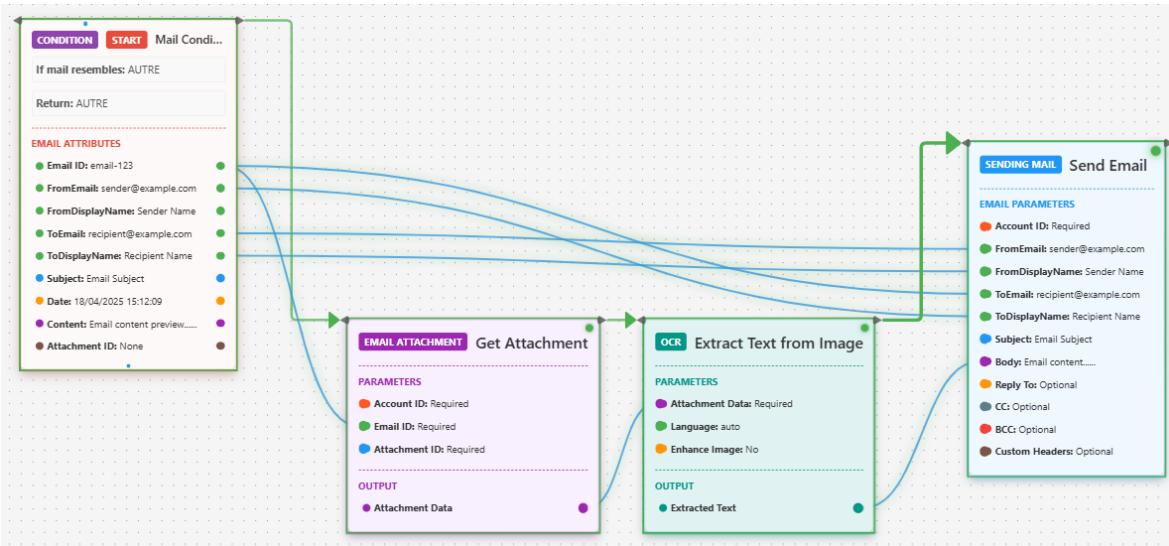


Figure 8.1 – Le flux à 4 nodes qui sera analysé ici

### 8.1.1 Vue d'ensemble du processus d'exécution

Le moteur d'exécution de MailFlow suit une séquence logique d'opérations pour traiter chaque email selon les règles définies. Nous allons examiner ce processus étape par étape, en commençant par le déclenchement de l'exécution jusqu'à la finalisation du traitement.

### *Déclenchement de l'exécution*

Lorsqu'un utilisateur décide de traiter un email, le processus commence par une action dans l'interface utilisateur React. Cette action déclenche une requête HTTP vers notre API backend.

```
1 // Lorsque l'utilisateur clique sur le bouton "Exécuter" dans
   l'interface React,
2 // une requête POST est envoyée au serveur avec l'identifiant
   de la tâche à exécuter
3 await axios.post('/api/flow/execute', { taskId });
```

**Listing 8.1** – Déclenchement de l'exécution depuis l'interface React

Cette requête est reçue par le serveur Express.js, qui la transmet au contrôleur approprié. Le contrôleur appelle ensuite la méthode `executeFlow` du moteur d'exécution, en passant l'identifiant de la tâche à traiter.

```
1 // Dans la méthode executeFlow du moteur d'exécution, nous
   commençons par
2 // récupérer ou créer la tâche associée à l'identifiant
   fourni
3 const task = await findOrCreateTask(taskId, payload);
4 // Par exemple, si task.type === 'facture impayée', task.
   sourceId === 'mail-123'
5 // Ces informations nous permettront d'identifier le type de
   traitement à appliquer
```

**Listing 8.2** – Récupération ou création de la tâche dans le moteur d'exécution

### *Recherche du point de départ*

Une fois la tâche récupérée, le moteur doit déterminer par quel nœud commencer l'exécution du flux. Pour cela, il recherche un nœud de condition marqué comme point de départ et correspondant au type de la tâche.

```

1 // Nous appelons la méthode findStartingNode pour identifier
   le premier nœud à exécuter
2 const startNode = this.findStartingNode(task);
3
4 // Cette méthode filtre tous les nœuds de condition pour
   trouver celui qui est
5 // marqué comme point de départ et qui correspond au type de
   la tâche
6 node.data.isStartingPoint === true
7 && node.data.returnText === task.type

```

**Listing 8.3** – Recherche du nœud de départ dans le flux

Lorsque le nœud de départ est trouvé, le système enregistre cette information dans les logs pour faciliter le débogage et le suivi de l'exécution.

```
1 Found starting node with returnText "facture impayée"
```

**Listing 8.4** – Log indiquant la découverte du nœud de départ

### *Initialisation du contexte d'exécution*

Avant de commencer l'exécution proprement dite, le moteur initialise un contexte d'exécution qui servira à stocker et à transmettre des données entre les différents nœuds du flux. Ce contexte est enrichi avec les attributs extraits de l'email associé à la tâche.

```

1 // Nous récupérons tous les attributs email définis dans le n
   œud de départ
2 // et nous les enrichissons avec les informations de la tâche
3 const emailAttrs = {
4   ...startNode.data.emailAttributes,
5   email_id: task.sourceId,      // Par exemple "mail-123"
6   fromEmail: task.senderEmail,
7   toEmail: task.recipientEmail
8 };
9

```

```

10 // Nous ajoutons ensuite ces attributs au contexte d'exé
   cution
11 // avec un préfixe "attr-" pour les distinguer des autres
    données
12 Object.entries(emailAttrs).forEach(([k, v]) =>
13   this.executionContext.set(`attr-${k}`, v)
14 );

```

**Listing 8.5** – Initialisation du contexte d'exécution avec les attributs de l'email

Cette étape est cruciale car elle permet d'avoir à disposition des informations comme attr-email\_id, attr-fromEmail, etc., qui seront utilisées par les nœuds suivants dans le flux.

### 8.1.2 Boucle principale d'exécution

Une fois le contexte initialisé, le moteur entre dans sa boucle principale d'exécution. Cette boucle parcourt séquentiellement tous les nœuds du flux, en suivant les liens d'exécution définis par l'utilisateur.

```

1 // Nous commençons par le nœud de départ identifié précédemment
2 let current = startNode;
3
4 // Tant qu'il y a un nœud à traiter, nous continuons l'exécution
5 while (current) {
6   // Nous exécutons le nœud courant en appelant la méthode
     executeNode
7   await this.executeNode(current);
8
9   // Puis nous récupérons le nœud suivant à exécuter en
     suivant les liens d'exécution
10  current = this.getNextNode(current);
11 }

```

---

**Listing 8.6 – Boucle principale d'exécution des nœuds**

Cette boucle est le cœur du moteur d'exécution. Elle garantit que tous les nœuds connectés au point de départ sont exécutés dans l'ordre défini par les liens d'exécution, jusqu'à ce qu'il n'y ait plus de nœud à traiter.

### 8.1.3 Exécution des différents types de nœuds

Le moteur d'exécution prend en charge différents types de nœuds, chacun ayant une fonction spécifique dans le traitement des emails. Voici comment sont exécutés les principaux types de nœuds.

#### *Nœud de pièce jointe d'email*

Ce type de nœud est responsable de la récupération des pièces jointes associées à un email. Il utilise l'API Unipile pour accéder aux données des pièces jointes.

```

1 // Nous récupérons l'identifiant de l'email et de la pièce
   jointe depuis le contexte d'exécution
2 const email_id      = this.executionContext.get('attr-
   email_id');
3 const attachment_id = this.executionContext.get('attr-
   attachment_id');
4
5 // Nous effectuons une requête à l'API Unipile pour récupérer
   la pièce jointe
6 const resp = await axios.get(
7   `${UNIPILE_BASE_URL}/emails/${email_id}/attachments/${
      attachment_id}`,
8   { responseType: 'arraybuffer', headers: { 'X-API-KEY':
      API_KEY } }
9 );

```

```

10
11 // Nous convertissons les données binaires en format Data URL
   pour faciliter leur manipulation
12 const dataUrl = `data:${resp.headers['content-type']};base64,
   `
13   + Buffer.from(resp.data).toString('base64');
14
15 // Nous stockons le résultat dans le contexte d'exécution
   pour qu'il soit accessible aux nœuds suivants
16 this.executionContext.set(`${node.id}-output-attachment`,
   dataUrl);

```

**Listing 8.7** – Exécution d'un nœud de pièce jointe d'email

### Nœud *OCR*

Le noeud OCR (Optical Character Recognition) est utilisé pour extraire du texte à partir d'images ou de documents numérisés. Il utilise la bibliothèque Tesseract.js pour effectuer cette reconnaissance de texte.

```

1 // Nous récupérons les données de la pièce jointe depuis le
   contexte d'exécution
2 const dataUrl = this.executionContext.get('attr-
   attachment_data');
3
4 // Nous initialisons un worker Tesseract pour la
   reconnaissance de texte
5 const worker = await createWorker();
6
7 // Nous configurons le worker pour la langue anglaise
8 await worker.loadLanguage('eng');
9 await worker.initialize('eng');
10
11 // Nous effectuons la reconnaissance de texte sur l'image
12 const { data:{ text } } = await worker.recognize(dataUrl);
13

```

```

14 // Nous libérons les ressources du worker
15 await worker.terminate();
16
17 // Nous stockons le texte extrait dans le contexte d'exé
   cution
18 this.executionContext.set('${node.id}-output-text', text);

```

**Listing 8.8** – Exécution d'un nœud OCR*Nœud d'envoi d'email*

Ce noeud est responsable de l'envoi d'emails en réponse à un email reçu. Il utilise également l'API Unipile pour envoyer les messages.

```

1 // Nous récupérons les informations nécessaires depuis le
   contexte d'exécution
2 const to      = this.executionContext.get('attr-toEmail');
3 const subject = this.executionContext.get('attr-subject');
4
5 // Le corps du message peut provenir soit d'un attribut spé
   cifique,
6 // soit de la sortie d'un nœud précédent (comme un nœud OCR)
7 const body    = this.executionContext.get('attr-body')
8           || this.executionContext.get('${prevId}-output-
   text');
9
10 // Nous préparons l'objet email avec toutes les informations
    nécessaires
11 const email = {
12   account_id: API_ID,
13   to:[{ identifier: to }],
14   subject,
15   body
16 };
17
18 // Nous envoyons l'email via l'API Unipile

```

```

19 const resp = await axios.post(
20   `${UNIPILE_BASE_URL}/emails`, email,
21   { headers:{ 'X-API-KEY': API_KEY } }
22 );
23
24 // Nous stockons la réponse de l'API dans le contexte d'exé
25 this.executionContext.set(`${node.id}-output`, resp.data);

```

**Listing 8.9** – Exécution d'un nœud d'envoi d'email

#### 8.1.4 Finalisation de l'exécution

Une fois que tous les nœuds ont été exécutés, le moteur finalise le traitement en mettant à jour le statut de la tâche dans la base de données et en retournant les résultats de l'exécution.

```

1 // Nous mettons à jour la tâche en base de données pour
  indiquer qu'elle a été complétée
2 await updateOne({ _id: task.id }, {
3   $set: { status:'completed', completedAt: new Date() }
4 });
5
6 // Nous retournons un objet indiquant le succès de l'exé
  cution et les résultats obtenus
7 return { success:true, result: executionResults };

```

**Listing 8.10** – Finalisation de l'exécution et mise à jour de la tâche

## 8.2 Résumé du fonctionnement du moteur d'exécution

En résumé, le moteur d'exécution de MailFlow fonctionne selon les étapes suivantes :

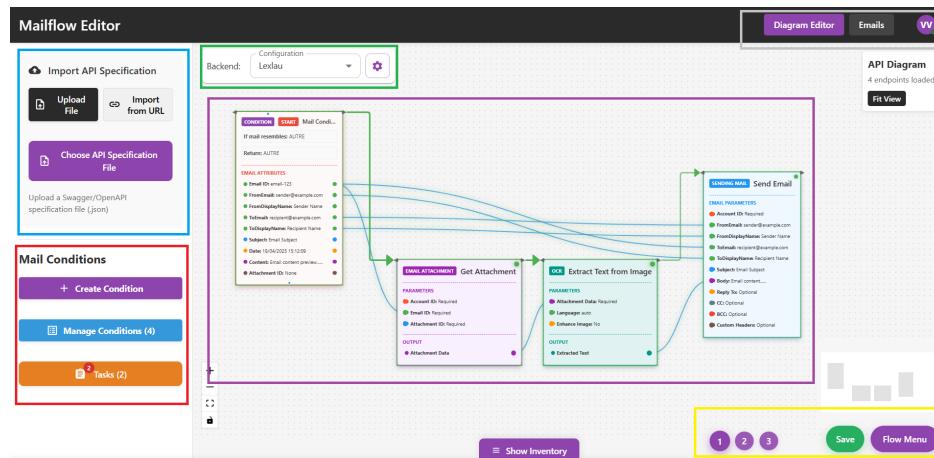
1. **Déclenchement** : L'utilisateur initie l'exécution via l'interface React, ce qui envoie une requête au serveur.
2. **Identification du point de départ** : Le moteur recherche dynamiquement le noeud de départ correspondant au type de tâche à traiter.
3. **Initialisation du contexte** : Un contexte d'exécution est créé et enrichi avec les attributs de l'email à traiter.
4. **Exécution séquentielle** : Le moteur parcourt tous les nœuds connectés par des liens d'exécution, en exécutant chacun selon son type.
5. **Traitement spécifique** : Chaque type de noeud effectue une action spécifique (récupération de pièce jointe, OCR, envoi d'email, etc.) et stocke ses résultats dans le contexte d'exécution.
6. **Finalisation** : Une fois tous les noeuds traités, la tâche est marquée comme complétée et les résultats sont retournés.

Cette approche modulaire et séquentielle rend le moteur d'exécution extrêmement flexible et capable de traiter n'importe quel flux défini par l'utilisateur dans l'interface graphique. La communication entre les noeuds via le contexte d'exécution permet de créer des chaînes de traitement complexes, où chaque noeud peut utiliser les résultats des noeuds précédents.

Le moteur est également conçu pour être robuste, avec une gestion appropriée des erreurs à chaque étape et une journalisation détaillée pour faciliter le débogage en cas de problème.

## 8.3 IHM

L'interface utilisateur de MailFlow a été conçue pour être intuitive et fonctionnelle, permettant aux utilisateurs de créer et gérer facilement des flux de traitement d'emails sans nécessiter de compétences en programmation. Voici une description détaillée des principales zones de l'interface :



**Figure 8.2 – Interface de l’éditeur MailFlow avec ses différentes zones fonctionnelles**

### 8.3.1 Zones fonctionnelles de l’interface

#### *Import de spécification API (Encadré bleu)*

Ce panneau permet l’importation de spécifications OpenAPI/Swagger, facilitant l’intégration avec des services externes. L’utilisateur peut charger une spécification depuis un fichier local ou via URL, ce qui génère automatiquement les définitions d’endpoints et crée les noeuds apiNode correspondants dans l’éditeur. Cette fonctionnalité est particulièrement utile pour connecter MailFlow à des systèmes existants sans configuration manuelle complexe.

#### *Conditions et tâches (Encadré rouge)*

Cette barre latérale centralise la gestion des règles de routage d’emails et le suivi des exécutions de flux :

- **Create Condition** : ouvre le formulaire permettant de définir un nouveau filtre de mail.
- **Manage Conditions (4)** : affiche la liste des conditions existantes (4 dans cet exemple).
- **Tasks (2)** : présente les exécutions de flux en attente ou en cours (2 dans cet exemple).

exemple).

Cette section permet aux utilisateurs de définir précisément quels types d'emails déclencheront quels flux, et de suivre l'état d'avancement des traitements en cours.

### *Configuration Backend (Encadré vert)*

Cette zone contient le sélecteur de Backend et sa roue crantée de paramétrage. Elle permet de choisir et configurer l'instance de l'API (baseUrl, authentification, timeout...) que le moteur d'exécution utilisera pour ses appels HTTP. Cette flexibilité permet d'adapter MailFlow à différents environnements (développement, test, production) ou de connecter plusieurs systèmes simultanément.

### *Zone d'édition du diagramme (Encadré violet)*

Il s'agit du canvas React Flow où l'utilisateur dispose et relie les différents nœuds (condition, emailAttachment, ocr, sendingMail, etc.). C'est le cœur de l'application, où se construit visuellement le flux de traitement des emails. L'utilisateur peut y définir :

- Le positionnement des nœuds
- Les connexions entre les handles d'entrée/sortie
- Les liens d'exécution entre les différentes étapes du traitement

### *Barre de navigation supérieure (Encadré gris)*

Cette barre contient les onglets principaux de l'application :

- **Diagram Editor** : l'éditeur de flux visuel
- **Emails** : la vue "boîte mail" pour consulter les messages
- **Profil utilisateur** : accès aux réglages du compte

Elle permet de naviguer entre les différentes fonctionnalités de l'application.

### *Barre d'actions (Encadré jaune)*

Située en bas à droite de l'interface, cette barre contient :

- **Boutons numérotés (1-2-3)** : représentent les étapes ou versions successives du flux
- **Save** : enregistre la version courante du diagramme en base de données
- **Flow Menu** : ouvre le menu général du flux (duplicer, supprimer, partager...)

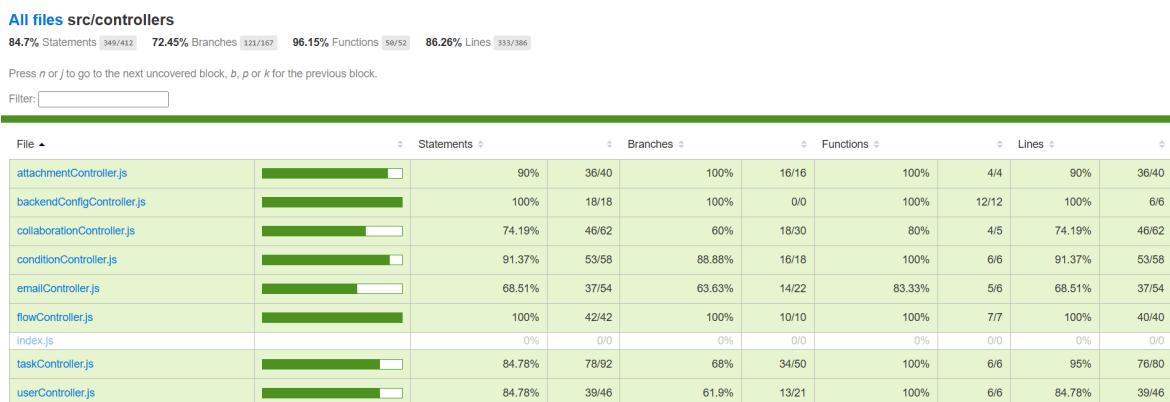
Cette zone permet de gérer rapidement les versions du flux et de déclencher des actions globales comme la sauvegarde ou le partage.

## 8.4 Tests unitaires

Pour s'assurer que chaque point d'entrée de l'API fonctionne correctement, nous avons centré nos premiers tests sur les contrôleurs Express. Ceux-ci valident le nom des routes, le code HTTP retourné et la structure des réponses, tout en simulant (“mockant”) les appels aux services métiers pour isoler la logique de routage.

Concrètement, chaque méthode du contrôleur est invoquée avec des requêtes factices et des services stubés, afin de vérifier qu'elle renvoie bien un succès ou une erreur appropriée (400, 401, 403, 404 ou 500) selon le scénario. Nous complétons ces validations par des appels réels en mémoire (via Supertest et une base MongoDB in-memory), pour tester les routes `/api/flows` dans leur ensemble, cookies JWT à l'appui, sans déployer de serveur ni toucher la base de production.

Le résultat est visible sur la capture de couverture :



**Figure 8.3 – Screen taux de couverture des tests unitaires**

La couche contrôleur atteint 100 % de couverture, et le taux global dans `src/controllers` est d'environ 85 % des instructions et 72 % des embranchements. Nous avions pour objectif minimal 80 % d'instructions et 70 % de branches, ce qui garantit une bonne robustesse dès la première phase.

À l'avenir, nous élargirons ces tests aux services métier, puis ajouterons des scénarios bout en bout pour couvrir l'intégralité du parcours utilisateur et prévenir toute régression.

# 9 Aspects financiers

## 9.1 Coûts de développement

Des dépenses personnelles ont été engagées à plusieurs reprises tout au long du développement de MailFlow : souscription d'un compte Unipile opérationnel, approvisionnement du crédit OpenAI pour tester le filtrage des courriels et les différentes intégrations API, le tout sans aucun soutien financier externe. Les frais de développement ont donc été assumés dans l'espoir que le déploiement en production permette de rentabiliser ces investissements.

Voici une estimation des coûts de développement :

- **Crédits Api OpenAi** : 25€ crédité en tout.
- **Abonnement Unipile** d'octobre à décembre puis de mars à juillet : 500€ (50€ par mois)
- **IA utilisées** dans un cadre d'apprentissage : 150€

Showing invoices within the past 12 months				
INVOICE	STATUS	AMOUNT	CREATED	
51554776-0005	Paid	\$7.26	13 avr. 2025, 18:40	<a href="#">View</a>
51554776-0004	Paid	\$12.10	30 oct. 2024, 08:38	<a href="#">View</a>
51554776-0001	Paid	\$6.05	18 oct. 2024, 11:05	<a href="#">View</a>

**Figure 9.1** – Historique de transactions Api OpenAI  
[9]

La question du modèle économique à adopter lors de la commercialisation a été soulevée à plusieurs reprises. Faute de qualifications spécifiques dans ce domaine, ce volet ne sera pas approfondi dans ce TFE. Il est néanmoins acquis qu'à terme l'API Unipile sera remplacée par une solution interne dédiée à la récupération et à l'envoi de courriels, tandis que le recours à OpenAI sera maintenu ; les clients seront alors responsables du crédit de leurs comptes pour continuer à bénéficier des fonctionnalités d'intelligence artificielle.

# 10 Conclusion

## 10.1 Difficultés rencontrées

Travailler sur ce TFE a représenté un défi de taille. Seul face à un environnement complexe et des technologies inexplorées, j'ai dû me montrer entièrement autonome : organiser mes propres réunions avec l'avocat, définir et respecter un planning précis pour livrer chaque jalon dans les temps, et partir d'une page blanche sans socle existant. Les contraintes de délai, la multiplicité des bugs et la nécessité de concevoir une architecture solide et évolutive ont rendu chaque étape particulièrement exigeante, tant sur le plan technique que personnel. L'une des plus grandes difficultés rencontrées a porté sur le moteur d'exécution des flux. Rendre opérationnelle la transmission des données d'un nœud à l'autre s'est avéré particulièrement complexe : malgré une bonne maîtrise de React, plusieurs jours ont été consacrés à tenter, sans succès, de corriger des bugs tenaces. Confronté à cette impasse, j'ai finalement choisi de tout reprendre depuis le début, en adoptant une approche modulaire : le problème a été découpé en plus petites briques, traitées une à une. Cette méthode, appliquée à plusieurs reprises lors du développement, m'a permis de sortir de l'impasse sans céder à la précipitation, et d'avancer progressivement jusqu'à une solution stable.

Pour surmonter ces obstacles, j'ai appliqué les méthodes et outils appris au cours de ma formation : découpage du projet en livrables intermédiaires, mise en place d'une approche Agile, utilisation rigoureuse des outils de débogage et de suivi de version, et veille constante sur les retours métier. Cette discipline m'a permis de transformer des difficultés en opportunités d'apprentissage : j'ai renforcé ma capacité d'analyse, affiné ma planification et développé une plus grande confiance en mes choix techniques. Au final, ces compétences – planification, autonomie, et adaptation – ont été la clé pour mener à bien ce projet ambitieux.

Bien que j'aie pris beaucoup de plaisir à travailler sur ce projet, si c'était à refaire, je m'y prendrais différemment. J'essaierais d'abord de ne pas être seul face à l'ampleur de la tâche, mais de pouvoir compter sur une équipe pour me soutenir et m'accompagner en cas de blocage : lorsqu'on est confronté à des doutes ou à des problèmes techniques, il est difficile de ne pouvoir s'appuyer que sur ses propres ressources. En outre, j'aurais peut-être choisi un périmètre un peu plus restreint, car ce TFE m'a demandé un investissement considérable en temps—entre trois et cinq heures de travail chaque soir de semaine—ainsi qu'en frais personnels que j'ai dû avancer seul. Cette naïveté initiale

est toutefois largement compensée par tout ce que j'ai appris : même si j'ai dû mettre ma vie personnelle entre parenthèses, j'ai gagné en autonomie, en rigueur et en confiance dans mes capacités techniques.

## 10.2 Bilan

Au cours de la phase de validation, de multiples visites au cabinet ont permis de recueillir des retours systématiquement positifs. Lors du dernier essai, un flux complet—préalablement conçu pour traiter une dizaine de types de courriels réels (adresses anonymisées)—a été exécuté. Sur ces dix scénarios testés, un seul a présenté une anomalie de traitement.

Charlotte, la juriste en charge du tri quotidien des emails, a exprimé sa gratitude : l'outil devrait lui faire gagner environ deux heures de travail chaque jour. Là où elle consacrait précédemment deux heures matinales (9 h–11 h) à la lecture, au classement et à la réponse manuelle des messages, seules cinq minutes seront désormais nécessaires pour lancer le processus automatique et en vérifier le bon déroulement.

Maître Epée et moi-même avons ensuite estimé que, grâce à MailFlow, le cabinet pourrait économiser entre **20 000 € et 50 000 € par an**, en se passant totalement d'un poste de secrétariat dédié (hors coûts d'installation et de maintenance, non inclus dans ce calcul).

Le client se dit particulièrement satisfait de constater que l'application est déjà partiellement opérationnelle, qu'elle répond précisément à ses attentes et qu'elle promet un impact économique significatif.

Pour ma part, je suis fier d'avoir su comprendre, analyser et développer cette solution en mobilisant l'ensemble des compétences acquises au cours de ma formation. Bien que MailFlow ne soit pas encore finalisé, ce TFE constitue un véritable accomplissement personnel.

## 10.3 Améliorations futures

Bien que les retours aient été extrêmement positifs et que le projet soit déjà à un stade avancé, plusieurs pistes d'évolution ont été identifiées pour enrichir l'application une fois ses fonctionnalités principales déployées. En voici une liste non exhaustive :

- **Refonte de l'interface** utilisateur afin de proposer une expérience encore plus intuitive et fluide.
- **Module IA de génération automatique de flux** : l'outil pourrait, à partir d'instructions en langage naturel, construire seul le cheminement des nœuds sans intervention manuelle sur l'éditeur.
- **Renforcement du système de métriques** : collecte de données plus détaillées pour aider l'utilisateur à localiser et optimiser les points de blocage critiques de ses flux.
- **Création de nouveaux types de nœuds** pour couvrir un plus grand nombre de cas d'usage métier et répondre à des besoins spécifiques.
- **Développement d'une solution interne à Unipile** pour prendre en charge directement l'envoi et la réception des courriels via une API propriétaire.

Ces améliorations constituent des axes de développement durables mais ne sont pas indispensables à la finalisation du projet. L'objectif initial — disposer d'une plateforme partiellement fonctionnelle intégrant l'essentiel des fonctionnalités de base — a d'ores et déjà été atteint dans le cadre de ce TFE.



# Bibliographie

- [1] Cabinet Lexlau. *Logo du cabinet d'avocats Lexlau* [image en ligne]. [s. l.], 2024. [consulté le 24 avril 2025]. Disponible à l'adresse : <https://www.facebook.com/lexlau.be/>
- [2] Gmelius. *Logo Gmelius* [image en ligne]. [s. l.], 2024. [consulté le 24 avril 2025]. Disponible à l'adresse : <https://ventures.swisscom.com/portfolio/gmelius/>
- [3] SaneBox. *Logo SaneBox* [image en ligne]. [s. l.], 2024. [consulté le 24 avril 2025]. Disponible à l'adresse : <https://blog.sanebox.com/2013/10/30/were-excited-to-share-the-new-sanebox-logo-with/>
- [4] Unipile. *Capture d'écran de l'Interface d'Unipile* [image en ligne]. [s. l.], 2024. [consulté le 24 avril 2025]. Disponible à l'adresse : <https://unipile.io/>
- [5] React. *Logo React* [image en ligne]. [s. l.], 2024. [consulté le 24 avril 2025]. Disponible à l'adresse : <https://fr.wikipedia.org/wiki/Fichier:React-icon.svg>
- [6] React Flow. *Logo React Flow* [image en ligne]. [s. l.], 2024. [consulté le 24 avril 2025]. Disponible à l'adresse : <https://webcatalog.io/fr/apps/react-flow>
- [7] NodeJs/Express. *Logo Node.js et Express* [image en ligne]. [s. l.], 2024. [consulté le 24 avril 2025]. Disponible à l'adresse : <https://medium.com/@douglas.rochedo/how-to-make-a-simple-server-in-express-js-4ae143cf95e5>
- [8] MongoDB. *Logo MongoDB* [image en ligne]. [s. l.], 2024. [consulté le 24 avril 2025]. Disponible à l'adresse : <https://fr.wikipedia.org/wiki/Fichier:MongoDB-Logo.svg>
- [9] OpenAI. *Capture d'écran de l'historique de transactions API OpenAI* [image en ligne]. [s. l.], 2025. [consulté le 24 avril 2025]. Disponible à l'adresse : <https://platform.openai.com/account/billing/history>

# Glossaire

**API** Interface de Programmation d'Application. Ensemble de règles et de protocoles permettant à différents logiciels de communiquer entre eux.

**CORS** Cross-Origin Resource Sharing. Mécanisme qui permet à des ressources web restreintes sur une page web d'être demandées par un autre domaine que celui qui a servi la première ressource.

**CRM** Customer Relationship Management (Gestion de la Relation Client). Système permettant de gérer les interactions avec les clients actuels et potentiels.

**Express.js** Framework web minimaliste et flexible pour Node.js qui fournit un ensemble robuste de fonctionnalités pour développer des applications web et mobiles.

**flux** Implémentation concrète d'un workflow dans l'application : un document stocké en base (collection Flow) qui regroupe le graphe de ses nœuds et connexions (ses versions) et que le FlowExecutionEngine parcourt pour exécuter le workflow.

**JWT** JSON Web Token. Standard ouvert basé sur JSON pour créer des jetons d'accès qui permettent de propager l'identité et les droits d'un utilisateur.

**MongoDB** Base de données NoSQL orientée documents qui stocke les données sous forme de documents JSON avec des schémas dynamiques.

**Mongoose** Bibliothèque de modélisation d'objets MongoDB pour Node.js qui fournit une solution simple basée sur des schémas pour modéliser les données de l'application.

**Node.js** Environnement d'exécution JavaScript côté serveur qui permet d'exécuter du code JavaScript en dehors d'un navigateur web.

**nœuds** Élément fondamental d'un diagramme de workflow représentant une opération atomique (condition, appel API, OCR, envoi de mail, etc.) doté d'entrées et de sorties pour chaîner les traitements.

**OAuth** Protocole d'autorisation ouvert permettant à des applications tierces d'accéder à des ressources protégées sans exposer les identifiants de l'utilisateur.

**OCR** Reconnaissance Optique de Caractères (Optical Character Recognition). Technologie permettant de convertir différents types de documents, tels que des images numérisées ou des fichiers PDF, en données modifiables et consultables.

**OpenAI** Entreprise de recherche en intelligence artificielle qui développe des modèles de langage avancés utilisés pour l'analyse et la génération de texte.

**React** Bibliothèque JavaScript open-source développée par Facebook pour créer des interfaces utilisateur ou des composants UI.

**React Flow** Bibliothèque React pour créer des éditeurs de flux de travail interactifs avec des nœuds et des connexions.

**Webhook** Mécanisme permettant à une application de fournir à d'autres applications des informations en temps réel via des requêtes HTTP automatisées.

**Workflow** Processus métier automatisé défini comme une suite d'étapes (nœuds) interconnectées visant à réaliser une tâche (récupération de mails, analyse, envoi de réponses...).

**XSS** Cross-Site Scripting. Type de faille de sécurité permettant l'injection de code malveillant dans des pages web consultées par d'autres utilisateurs.