

RAPPORT DE PROJET DE FIN D'ANNÉE

Présenté en vue de l'obtention de la  
LICENCE EN INGÉNIERIE DES SYSTÈMES INFORMATIQUES

Parcours : Systèmes Embarqués et Internet des Objets

---

## SurveillAPI Tableau de bord de supervision des APIs

---

*Par*  
MERDESSI EMNA

Réalisé au sein de VERMEG



Encadrant professionnel :

Mr Mohamed Salama ZIADI, Architecte Principal Senior  
des Opérations du Marché Bancaire

Année Universitaire : 2025-2026

RAPPORT DE PROJET DE FIN D'ANNÉE

Présenté en vue de l'obtention de la  
LICENCE EN INGÉNIERIE DES SYSTÈMES INFORMATIQUES

Parcours : Systèmes Embarqués et Internet des Objets

---

## SurveillAPI

### Tableau de bord de supervision des APIs

---

Par  
MERDESSI EMNA

Réalisé au sein de VERMEG



AUTORISATION DE DÉPÔT DU RAPPORT DE PROJET DE FIN D'ANNÉE :

Encadrant professionnel : Mr Mohamed Salama ZIADI, Architecte  
Principal Senior des Opérations du Marché Bancaire

Le : 16 / 10 / 2025

Signature :

Mohamed Salama ZIADI  
  


# Remerciements

Je tiens à remercier le groupe **VERMEG** pour m'avoir offert l'opportunité de réaliser ce stage enrichissant. J'exprime toute ma gratitude à mon encadrant, **M. Mohamed Salama ZIADI**, pour son accompagnement, ses conseils et sa disponibilité, ainsi qu'à l'ensemble de l'équipe de **VERMEG** pour son accueil chaleureux et son aide précieuse.

# Table des matières

|                                                                  |           |
|------------------------------------------------------------------|-----------|
| <b>Introduction Générale</b>                                     | <b>6</b>  |
| <b>1 Cadre général du projet</b>                                 | <b>7</b>  |
| Introduction . . . . .                                           | 7         |
| 1.1 Présentation de l'entreprise d'accueil . . . . .             | 7         |
| 1.1.1 L'organisation de VERMEG Services . . . . .                | 7         |
| 1.1.2 Les activités de Vermeg Services . . . . .                 | 8         |
| 1.1.3 Les produits de Vermeg . . . . .                           | 8         |
| 1.1.4 Les clients de Vermeg . . . . .                            | 9         |
| 1.2 Présentation du projet . . . . .                             | 9         |
| 1.2.1 Contexte de projet . . . . .                               | 9         |
| 1.2.2 Problématique . . . . .                                    | 9         |
| 1.2.3 Étude de l'existant . . . . .                              | 9         |
| 1.2.4 Solution proposée . . . . .                                | 10        |
| Conclusion . . . . .                                             | 11        |
| <b>2 Analyse des exigences</b>                                   | <b>12</b> |
| Introduction . . . . .                                           | 12        |
| 2.1 Analyse et spécification des exigences . . . . .             | 12        |
| 2.1.1 Exigences fonctionnelles . . . . .                         | 12        |
| 2.1.2 Exigences non fonctionnelles . . . . .                     | 13        |
| 2.2 Architecture . . . . .                                       | 13        |
| Conclusion . . . . .                                             | 13        |
| <b>3 Modélisation du système</b>                                 | <b>14</b> |
| Introduction . . . . .                                           | 14        |
| 3.1 Diagramme de cas d'utilisation . . . . .                     | 14        |
| 3.2 Tableaux des cas d'utilisation . . . . .                     | 15        |
| 3.3 Diagramme de classes . . . . .                               | 16        |
| 3.3.1 Diagramme de classes métier . . . . .                      | 16        |
| 3.3.2 Diagramme de classes – modèle de données MongoDB . . . . . | 17        |
| 3.4 Diagramme de séquence . . . . .                              | 18        |
| Conclusion . . . . .                                             | 18        |
| <b>4 Implémentation</b>                                          | <b>19</b> |
| Introduction . . . . .                                           | 19        |
| 4.1 Environnement de travail . . . . .                           | 19        |
| 4.1.1 Environnement matériel . . . . .                           | 19        |
| 4.1.2 Environnement logiciel . . . . .                           | 20        |
| 4.2 Technologies utilisées . . . . .                             | 21        |
| 4.3 Tâches Réalisées . . . . .                                   | 21        |
| Conclusion . . . . .                                             | 27        |
| <b>Conclusion Générale</b>                                       | <b>28</b> |
| <b>Netographie</b>                                               | <b>29</b> |
| <b>A Guide d'installation</b>                                    | <b>30</b> |

|          |                                  |           |
|----------|----------------------------------|-----------|
| <b>B</b> | <b>Mini-extrait de code</b>      | <b>32</b> |
| <b>C</b> | <b>Lien vers le dépôt GitHub</b> | <b>34</b> |

# Table des figures

|      |                                                                     |    |
|------|---------------------------------------------------------------------|----|
| 1.1  | Organigramme de l'entreprise . . . . .                              | 8  |
| 1.2  | Architecture générale de la solution de supervision d'API . . . . . | 11 |
| 2.1  | Architecture de surveillance d'API . . . . .                        | 13 |
| 3.1  | Diagramme de cas d'utilisation du système . . . . .                 | 14 |
| 3.2  | Diagramme de classes métier . . . . .                               | 16 |
| 3.3  | Diagramme de classes représentant le modèle de données MongoDB . .  | 17 |
| 3.4  | Diagramme de séquence . . . . .                                     | 18 |
| 4.1  | Page GitHub : Paramètres des applications OAuth . . . . .           | 21 |
| 4.2  | Page de connexion GitHub pour accéder au Dashboard . . . . .        | 22 |
| 4.3  | Accès au Dashboard après authentification GitHub . . . . .          | 23 |
| 4.4  | Requête faite dans Postman . . . . .                                | 24 |
| 4.5  | Log enregistré dans MongoDB Compass . . . . .                       | 24 |
| 4.6  | Requête incorrecte dans Postman . . . . .                           | 25 |
| 4.7  | Réponse d'erreur (404) . . . . .                                    | 25 |
| 4.8  | Log de l'erreur dans MongoDB Compass . . . . .                      | 26 |
| 4.9  | Réponse JSON de l'endpoint /stats . . . . .                         | 26 |
| 4.10 | Résultat de la recherche avancée /search . . . . .                  | 27 |
| 4.11 | QR code du dépôt GitHub du projet <i>SurveillAPI</i> . . . . .      | 27 |

# Table des tableaux

|     |                                                                     |    |
|-----|---------------------------------------------------------------------|----|
| 1.1 | Organisation 2025 de VERMEG par pôles métiers . . . . .             | 7  |
| 1.2 | Comparaison des principale solutions de monitoring d'API . . . . .  | 10 |
| 2.1 | Exigences Fonctionnelles . . . . .                                  | 12 |
| 3.1 | Cas d'utilisation « Authentification via GitHub » . . . . .         | 15 |
| 3.2 | Cas d'utilisation « Consultation des statistiques » . . . . .       | 15 |
| 3.3 | Cas d'utilisation « Enregistrement automatique des logs » . . . . . | 15 |
| 4.1 | Environnement matériel . . . . .                                    | 19 |
| 4.2 | Environnement logiciel . . . . .                                    | 20 |

# Introduction Générale

Dans un monde de plus en plus connecté, les systèmes d'information des banques, des start-ups ou encore des associations reposent massivement sur des applications interconnectées échangeant des données via Internet. Ces échanges sont rendus possibles grâce aux **API** (Interfaces de Programmation d'Applications), qui servent de points d'accès logiciels permettant à différents programmes de communiquer entre eux.

Cependant, ces interfaces ne sont pas exemptes de dysfonctionnements : ralentissements, erreurs ou pannes peuvent survenir et dégrader significativement l'expérience utilisateur, voire rendre les services temporairement inaccessibles. Il est donc indispensable de superviser en temps réel les API afin d'analyser :

- le nombre d'appels effectués ;
- les temps de réponse ;
- les consommateurs les plus actifs ;
- ainsi que les anomalies éventuelles.

Dans ce contexte, le présent projet vise à développer une application légère basée sur **Node.js** et **Express.js**, offrant les fonctionnalités suivantes :

- l'enregistrement automatique de chaque appel API ;
- le stockage des logs dans une base de données **MongoDB** ;
- l'affichage des statistiques via un **tableau de bord web interactif** ;
- la possibilité d'effectuer des **recherches avancées** parmi les logs enregistrés.

La validation du fonctionnement de l'application est assurée à l'aide de **Postman**, un outil couramment utilisé pour tester les requêtes HTTP. L'application est conçue pour être simple, modulaire et facilement intégrable dans un environnement de développement ou de test. Par ailleurs, l'accès au tableau de bord est sécurisé par une authentification **GitHub (OAuth2)**, implémentée à l'aide du module **passport-github2**, garantissant ainsi un accès restreint aux seuls utilisateurs autorisés.

Ce rapport est structuré en quatre chapitres :

- **Chapitre 1** : Présentation de l'entreprise d'accueil **VERMEG**, du contexte et des enjeux du projet ;
- **Chapitre 2** : Analyse des besoins et description de l'architecture retenue ;
- **Chapitre 3** : Modélisation du système à l'aide des diagrammes **UML** ;
- **Chapitre 4** : Implémentation, tests réalisés et résultats obtenus.

Enfin, une **conclusion générale** clôture le rapport en dressant un bilan du travail effectué et en proposant des perspectives d'amélioration futures.

# Chapitre 1

## Cadre général du projet

### Introduction

L'étude du projet constitue une étape essentielle permettant de présenter l'environnement de l'entreprise d'accueil ainsi que les objectifs à atteindre. Dans un premier temps, un aperçu de l'organisme d'accueil sera fourni afin de mieux situer le contexte. Par la suite, le projet sera décrit en précisant la problématique identifiée, l'étude de l'existant ainsi que la solution proposée pour y répondre.

### 1.1 Présentation de l'entreprise d'accueil

Le projet a été réalisé au sein de la société « Vermeg Services » située aux Berges du Lac. « Vermeg » est une société de services et d'ingénierie informatique dont l'activité majeure est axée sur l'édition de logiciels et l'intégration de systèmes. Elle a été fondée en 1994 en mains privées, elle est inscrite au registre en France et aux Pays-Bas. Elle emploie près de 250 personnes réparties dans plusieurs sièges : Tunis, Paris, Madrid, Luxembourg, Londres (Siège de vente seulement) et Johannesburg (Partenariat local). [1]



Logo officiel de Vermeg Services

#### 1.1.1 L'organisation de VERMEG Services

Depuis 2024, le Groupe VERMEG s'est réorganisé autour de trois pôles métiers, détaillés dans le tableau 1.1.

TABLE 1.1 – Organisation 2025 de VERMEG par pôles métiers.

| Pôle                        | Mission principale                                                                                                                                                                                                               | Produits / cibles                                               |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|
| Global Custody & Collateral | Développer et maintenir les solutions destinées aux infrastructures de marché, aux conservateurs d'actifs et aux banques d'investissement.                                                                                       | MEGARA, MEGALEND, modules de clearing & collateral management   |
| Global Insurance            | Répondre aux besoins des assureurs vie et non-vie concernant la gestion du passif, l'actif général, les unités de compte et la modernisation des parcours clients.                                                               | Suite OMEGA Insurance, MEGARA Life                              |
| Vermeg Applied              | Exploiter l'intelligence artificielle pour améliorer l'assurance qualité, le delivery et le conseil sur tous les environnements clients, afin de réduire les délais d'implémentation et d'augmenter la qualité des déploiements. | Framework IA interne, bots de tests, accélérateurs data-quality |

L'organigramme ci-dessous illustre la structure hiérarchique de VERMEG Services ainsi que la répartition de ses principaux pôles métiers.

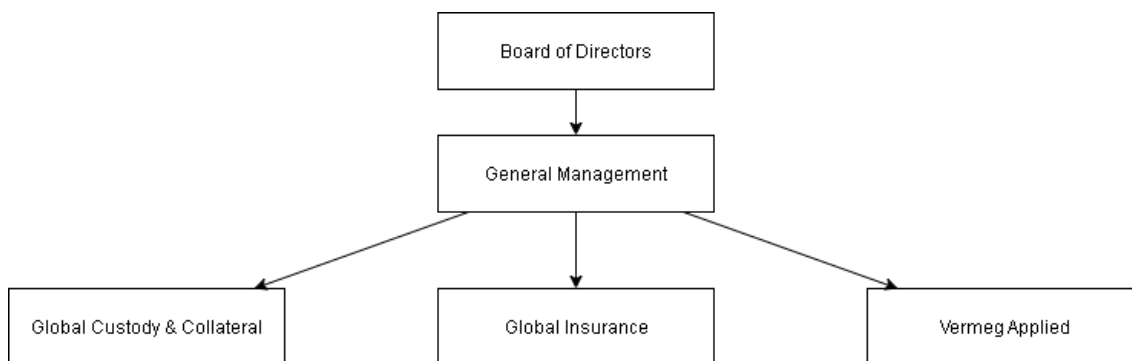


FIGURE 1.1 – Organigramme de l'entreprise

### 1.1.2 Les activités de Vermeg Services

Vermeg Services est incluse dans le pôle de recherche et de développement du groupe. Elle est principalement considérée comme étant un éditeur de progiciels bancaires. Les domaines d'activités de Vermeg sont les suivants :

- Asset Management (Front et Back-office) : Le Front Office (appelé également Front line) désigne la partie frontale de l'entreprise, visible par la clientèle et en contact direct avec celle-ci, comme les équipes de marketing, de support utilisateur ou de service après-vente.
- Le Back Office, à l'inverse, désigne l'ensemble des parties du système d'information auxquelles l'utilisateur final n'a pas accès. Il s'agit donc de tous les processus internes à l'entreprise (production, logistique, stocks, comptabilité, gestion des ressources humaines, etc.).
- Titres : Clearing, OST<sup>1</sup>, Négó : opérations sur les titres.
- Télé compensation et RTGS<sup>2</sup> : Télé compensation ou clearing : Opération de mise à jour des virements et prélèvements bancaires entre établissements financiers, effectuée à travers un réseau.

### 1.1.3 Les produits de Vermeg

Vermeg Services offre une gamme de solutions logicielles pour les gestionnaires des actifs, les banques et les institutions financières impliquées dans la gestion de portefeuille et l'administration de fonds. Voici quelques produits clés :

**MEGALEND** : C'est une solution globale pour la gestion des échanges financiers sécurisés, les emprunts et les repos. Cette solution est destinée aux banques, aux compagnies d'assurance et aux gestionnaires de fonds.

**OMEGA** : C'est une suite de solutions logicielles dédiées au marché de gestion des capitaux, fournissant les fonctionnalités front, middle et back office.

**MEGARA** : C'est une suite des composants de solutions traitant la pleine portée des services de sécurité et de la gestion de portefeuille.

**MEGACOR** : C'est une solution permettant d'automatiser entièrement le processus de gestion des actions d'entreprise.

---

1. OST : Opérations sur Titres

2. RTGS : Real-Time Gross Settlement

### 1.1.4 Les clients de Vermeg

Les produits de « Vermeg Services » ont été choisis par plus de 100 organismes bancaires et financiers dans le monde entier, situés dans plus de 15 pays et répartis sur quatre continents. La liste suivante présente quelques-uns des principaux clients de Vermeg :

- Société Générale (Londres, Madrid, Athènes, Kiev, Bucarest, Afrique du Sud).
- Banque de France.
- Banque Nationale de Paris (BNP Paribas).
- Banque de Luxembourg (Belgique).
- Credit Suisse Asset Management.
- Banque et Caisse d'Épargne de l'État (BCEE, Luxembourg).
- Banco Santander Central Hispano (BSCH, Madrid).
- Banco Bilbao Vizcaya Argentaria (BBVA, Madrid).
- BNP (Hong Kong, Taipei, Montréal, New York).

## 1.2 Présentation du projet

### 1.2.1 Contexte de projet

À l'ère du numérique, les entreprises et organisations dépendent fortement des systèmes connectés pour échanger et traiter des données en temps réel. Les API (Interfaces de Programmation Applicative) jouent un rôle central dans cette communication, permettant aux applications de s'interconnecter et de fonctionner ensemble de manière fluide. Cependant, des problèmes tels que la lenteur, les erreurs ou l'indisponibilité des API peuvent affecter la qualité du service et impacter l'expérience utilisateur. Il est donc crucial de surveiller et analyser les appels API pour garantir la performance et la fiabilité des systèmes.

### 1.2.2 Problématique

Dans les environnements modernes, les applications consomment de plus en plus d'API, rendant leur supervision complexe. Les principales problématiques identifiées sont :

- Difficulté à suivre en temps réel le comportement des API et les requêtes des clients,
- Manque de visibilité sur les performances et les temps de réponse,
- Impossibilité de détecter rapidement les anomalies, erreurs ou surcharges,

Difficulté à identifier les clients ou services les plus consommateurs, ce qui complique l'optimisation et la maintenance des systèmes.

### 1.2.3 Étude de l'existant

Avant de concevoir une solution spécifique, il est nécessaire d'analyser les outils existants de monitoring d'API afin d'identifier leurs forces et leurs limites. Parmi les solutions les plus utilisées, on retrouve **Postman**, **Swagger**, **AWS CloudWatch** et **Datadog**.

**Postman** est principalement utilisé pour tester les API et automatiser certains scénarios de requêtes, tandis que **Swagger** se distingue par sa capacité à générer et maintenir une documentation interactive basée sur le standard OpenAPI. De leur côté, **AWS CloudWatch** et **Datadog** représentent des solutions cloud avancées, offrant un suivi en temps réel, des alertes et des visualisations adaptées aux grandes infrastructures.

Le tableau suivant présente une comparaison entre ces outils selon leurs avantages et leurs limites :

TABLE 1.2 – Comparaison des principale solutions de monitoring d’API

| Outil                      | Avantages principaux                                                              | Limites identifiées                                                          |
|----------------------------|-----------------------------------------------------------------------------------|------------------------------------------------------------------------------|
| <b>Postman</b> [4]         | - Facilité d’utilisation<br>- Tests d’API automatisés<br>- Interface conviviale   | - Moins adapté au suivi en temps réel<br>- Fonctionnalités avancées payantes |
| <b>Swagger</b> [5]         | - Documentation interactive<br>- Standard OpenAPI largement adopté                | - Pas de monitoring en temps réel<br>- Peu adapté à l’analyse des logs       |
| <b>AWS Cloud-Watch</b> [6] | - Intégration native avec AWS<br>- Suivi en temps réel<br>- Alertes configurables | - Dépendance au cloud AWS<br>- Coût élevé pour usage intensif                |
| <b>Datadog</b> [7]         | - Solution complète (logs, métriques, alertes)<br>- Visualisations avancées       | - Coût élevé pour petites structures<br>- Configuration parfois complexe     |

Cette analyse montre que, malgré leur efficacité, ces outils présentent des contraintes liées au coût, à la complexité de configuration ou à l’absence de certaines fonctionnalités spécifiques, notamment pour l’analyse personnalisée des logs. Ces limites justifient le développement d’une solution sur mesure, mieux adaptée aux besoins de supervision et d’analyse dans ce projet.

#### 1.2.4 Solution proposée

Le projet vise à développer une application complète de **supervision et d’analyse des appels API**, sécurisée et interactive. La solution proposée comprend les fonctionnalités principales suivantes :

- **Collecte automatique des requêtes API** : chaque requête est interceptée par le Middleware et ses informations pertinentes (endpoint, méthode, IP, temps de réponse, date/heure) sont enregistrées dans **MongoDB**. [3]
- **Middleware personnalisé** : mesure le temps de réponse, détecte les anomalies et prépare les données pour l’enregistrement.
- **Tableau de bord web interactif** : affiche le nombre de requêtes, les temps de réponse, la répartition par endpoint, les clients les plus actifs et l’historique des appels.
- **Endpoints supplémentaires** :
  - **/search** : recherche avancée des logs selon différents critères (mot-clé, méthode, client, plage de dates).
  - **/count** : comptage rapide des logs correspondant aux critères.
- **Sécurité et authentification** :
  - **Passport.js + passport-github2** : middleware Node.js pour gérer l’authentification via GitHub (OAuth2) [9] et sécuriser l’accès au Dashboard.
  - **Sessions Express** : stockage sécurisé des sessions utilisateur après authentification [2].

Cette solution permet de :

- Surveiller les API en temps réel.
- Détecter rapidement les anomalies et erreurs (404, 500, etc.).
- Optimiser la maintenance et améliorer la performance des systèmes.
- Assurer un accès sécurisé et contrôlé au Dashboard grâce à l’authentification GitHub.

L'architecture présentée ci-dessous illustre la structure globale de la solution de supervision d'API. Elle met en évidence les principales composantes du système : la collecte des logs, le stockage dans MongoDB, le traitement et l'analyse en temps réel, ainsi que l'affichage des statistiques sur le tableau de bord interactif.

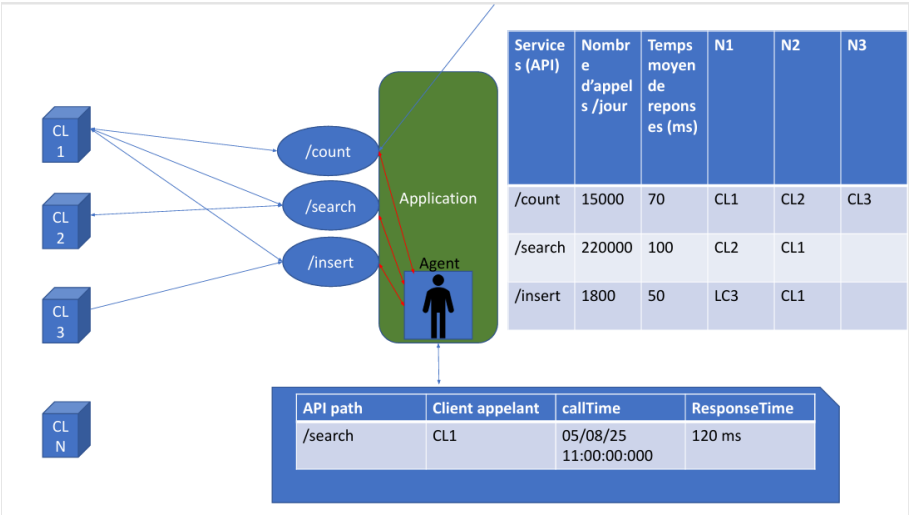


FIGURE 1.2 – Architecture générale de la solution de supervision d'API

### Conclusion

Tout au long de ce chapitre, nous avons présenté l'organisme d'accueil et ses domaines d'activités, ainsi que la solution adéquate adaptée au projet. Le chapitre suivant sera dédié à l'étude des exigences fonctionnelles et non fonctionnelles, ainsi qu'à l'architecture de notre application.

# Chapitre 2

## Analyse des exigences

### Introduction

Dans ce chapitre, nous allons spécifier les exigences fonctionnelles et non fonctionnelles, ainsi que l'architecture de l'application, afin de clarifier et structurer le projet.

### 2.1 Analyse et spécification des exigences

La spécification des exigences permet d'analyser les fonctionnalités de l'application afin de définir clairement les exigences fonctionnelles et non fonctionnelles.

#### 2.1.1 Exigences fonctionnelles

Les exigences fonctionnelles définissent les comportements et les fonctionnalités que l'application doit fournir afin de répondre aux besoins des utilisateurs et aux objectifs du projet.

TABLE 2.1 – Exigences Fonctionnelles

| Exigence                          | Description                                                                                                                                                                                                                                              |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Collecte automatique des logs     | Le système doit intercepter chaque requête API entrante via un middleware. Les informations suivantes doivent être capturées : URL, méthode HTTP, IP client, temps de réponse, code statut HTTP et horodatage.                                           |
| Stockage des données              | Les logs collectés doivent être persistés dans une base de données MongoDB optimisée pour l'écriture rapide et les opérations d'agrégation.                                                                                                              |
| Affichage du tableau de bord      | Le tableau de bord web doit présenter les statistiques globales en temps quasi-réel, incluant : nombre total de requêtes, temps de réponse moyen/min/max, répartition par endpoint (URL), répartition des codes HTTP, liste des clients les plus actifs. |
| Consultation de l'historique      | L'utilisateur doit pouvoir consulter une liste paginée des dernières requêtes enregistrées avec toutes les informations associées (URL, IP, méthode, etc.).                                                                                              |
| Recherche et filtrage avancé      | L'utilisateur doit pouvoir effectuer des recherches avec filtres combinables : mot-clé, méthode HTTP, adresse IP, plage de dates, code statut HTTP. Les résultats doivent être paginés.                                                                  |
| Authentification GitHub OAuth2    | L'accès au tableau de bord doit être sécurisé. Seuls les utilisateurs authentifiés via GitHub et approuvés par l'administrateur peuvent se connecter.                                                                                                    |
| Gestion de la session utilisateur | Après authentification, le système doit maintenir une session sécurisée et permettre à l'utilisateur de se déconnecter de manière sécurisée.                                                                                                             |
| Compteur de logs                  | Le système doit afficher le nombre total de logs enregistrés dans la base de données, visible sur le tableau de bord.                                                                                                                                    |

### 2.1.2 Exigences non fonctionnelles

Les exigences non fonctionnelles définissent le comportement et les qualités du système. Les principales exigences non fonctionnelles du projet sont les suivantes :

- **Performance** : le système doit traiter et enregistrer chaque requête rapidement, sans générer de ralentissements significatifs du serveur.
- **Fiabilité** : les logs doivent être correctement stockés et pouvoir être récupérés à tout moment, assurant l'intégrité des données.
- **Sécurité** : l'accès au tableau de bord et aux informations sensibles doit être protégé contre toute utilisation non autorisée, notamment via l'authentification GitHub OAuth2.
- **Modifiabilité et évolutivité** : la solution doit permettre l'ajout futur de nouvelles fonctionnalités, comme les alertes automatiques, sans perturber le fonctionnement existant.

## 2.2 Architecture

Le système est basé sur une architecture **client-serveur**, simple, légère et sécurisée, permettant de suivre le parcours complet d'une requête, depuis l'authentification jusqu'à la consultation des statistiques et l'enregistrement des logs.

L'utilisateur, représentant l'administrateur ou le développeur, s'authentifie via GitHub (OAuth2). GitHub gère le processus d'authentification et renvoie un callback au serveur Node.js/Express, qui crée et valide la session utilisateur. Une fois l'utilisateur connecté, il peut accéder au **Dashboard Web** pour consulter les statistiques en temps réel.

Chaque requête envoyée au serveur est interceptée par le middleware **SurveillAPI**, qui capture les métadonnées essentielles : endpoint, méthode, IP, code HTTP, durée et date/heure. Ces informations sont ensuite stockées de manière persistante dans MongoDB, permettant un suivi détaillé et une analyse complète des appels API.

Les principales étapes du fonctionnement du système sont les suivantes :

1. L'utilisateur initie l'authentification via GitHub.
2. GitHub renvoie un callback au serveur contenant le jeton OAuth2.
3. Le serveur valide le jeton et accorde l'accès au Dashboard.
4. L'utilisateur consulte les statistiques depuis le Dashboard.
5. Le serveur interagit avec MongoDB pour récupérer ou enregistrer les logs des requêtes.

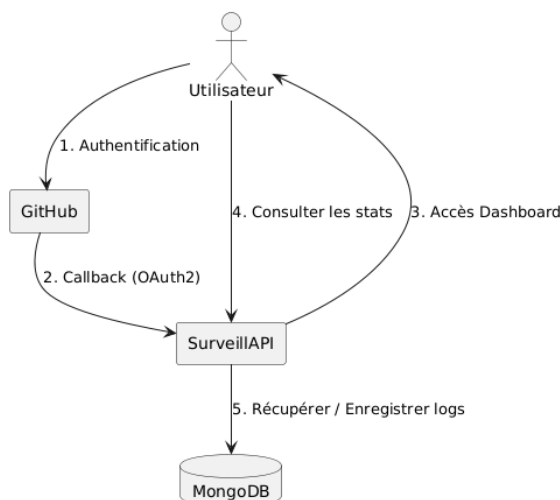


FIGURE 2.1 – Architecture de surveillance d'API

## Conclusion

Dans ce chapitre, nous avons précisé les exigences fonctionnelles et non fonctionnelles de notre système. Le chapitre suivant sera consacré à la modélisation.

## Chapitre 3

# Modélisation du système

### Introduction

Dans ce chapitre, nous modéliserons le système pour visualiser sa structure et son fonctionnement avant de le développer.

Ce chapitre présente les principaux diagrammes UML : le diagramme de cas d'utilisation pour les interactions utilisateur-système, le diagramme de classes pour la structure des entités, et le diagramme de séquence pour le déroulement des échanges. Ces représentations offrent une vision globale et cohérente du système et servent de référence pour le développement.

### 3.1 Diagramme de cas d'utilisation

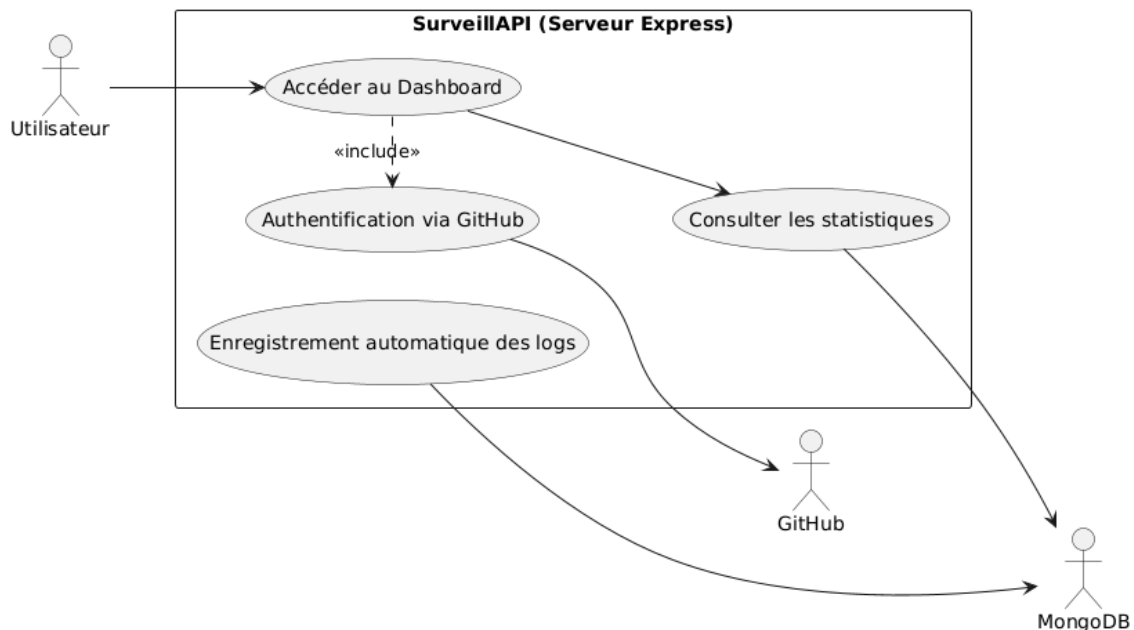


FIGURE 3.1 – Diagramme de cas d'utilisation du système

Le diagramme de cas d'utilisation (figure 3.1) illustre les interactions entre l'administrateur/-développeur et le système. Il met en évidence les fonctionnalités réellement développées dans le cadre de ce projet :

- Authentification via GitHub pour accéder au Dashboard
- Consultation des statistiques depuis le Dashboard
- Enregistrement automatique des logs par le middleware, assurant la traçabilité complète des appels API dans la base de données MongoDB.

## 3.2 Tableaux des cas d'utilisation

TABLE 3.1 – Cas d'utilisation « Authentification via GitHub »

|                          |                                                                                                                               |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <b>Cas d'utilisation</b> | Authentification via GitHub pour accéder au Dashboard                                                                         |
| Acteur                   | Administrateur / Développeur                                                                                                  |
| Pré-condition            | L'utilisateur doit posséder un compte GitHub valide.                                                                          |
| Condition nominale       | L'utilisateur initie l'accès au Dashboard, est redirigé vers GitHub pour authentification, puis le serveur valide la session. |
| Post-condition           | L'utilisateur est connecté et peut accéder au Dashboard.                                                                      |
| Exception                | Si l'authentification échoue, l'accès au Dashboard est refusé.                                                                |

TABLE 3.2 – Cas d'utilisation « Consultation des statistiques »

|                          |                                                                                                                                                                                    |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Cas d'utilisation</b> | Consultation des statistiques depuis le Dashboard                                                                                                                                  |
| Acteur                   | Administrateur / Développeur                                                                                                                                                       |
| Pré-condition            | L'utilisateur est authentifié et des logs existent dans la base de données.                                                                                                        |
| Condition nominale       | L'utilisateur sélectionne une période ou des critères depuis le Dashboard, et le système affiche les statistiques (temps de réponse, endpoints les plus utilisés, clients actifs). |
| Post-condition           | Les statistiques sont affichées et analysables par l'utilisateur.                                                                                                                  |
| Exception                | Si aucune donnée ne correspond aux critères, le système affiche « Aucune statistique disponible ».                                                                                 |

TABLE 3.3 – Cas d'utilisation « Enregistrement automatique des logs »

|                          |                                                                                                                                                          |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Cas d'utilisation</b> | Enregistrement automatique des logs par le middleware                                                                                                    |
| Acteur                   | Administrateur / Développeur                                                                                                                             |
| Pré-condition            | Une requête API est envoyée par un client.                                                                                                               |
| Condition nominale       | Le middleware intercepte chaque requête, capture les métadonnées (endpoint, méthode, IP, durée, statut HTTP, date/heure) et les enregistre dans MongoDB. |
| Post-condition           | Les logs sont stockés dans MongoDB, assurant la traçabilité complète.                                                                                    |
| Exception                | Si l'enregistrement échoue, une erreur est générée et consignée.                                                                                         |

### 3.3 Diagramme de classes

#### 3.3.1 Diagramme de classes métier

La figure 3.2 synthétise l'architecture logicielle de la solution de supervision. Il met en évidence les sept entités principales, leurs responsabilités et leurs interactions :

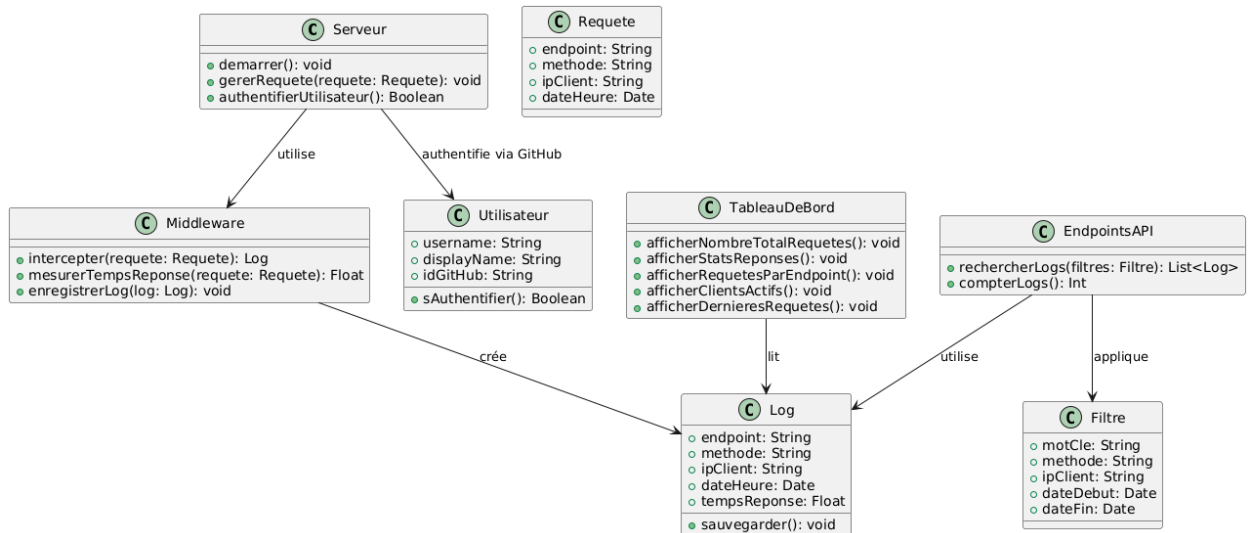


FIGURE 3.2 – Diagramme de classes métier

- **Utilisateur** : représente l'utilisateur authentifié via GitHub.
- **Serveur (Node.js / Express)** : reçoit et traite les requêtes.
  - Méthodes principales : `demarrer()`, `gererRequete()`, `authentifierUtilisateur()` (OAuth2 GitHub)
- **Middleware** : intercepte chaque requête et enregistre les logs.
  - Méthodes : `intercepter()`, `mesurerTempsReponse()`, `enregistrerLog()`
- **Requete** : représente une requête entrante. *Attributs* : endpoint, méthode, ipClient, date-Heure.
- **Log** : stocke les données de chaque requête. *Attributs* : mêmes que Requete + tempsReponse. Méthode : `sauvegarder()`.
- **TableauDeBord** : visualise les statistiques. Méthodes : `afficherNombreTotalRequetes()`, `afficherStatsReponses()`, `afficherRequetesParEndpoint()`, `afficherClientsActifs()`, `afficherDernieresRequetes()`.
- **EndpointsAPI** : fournit les endpoints `/search` et `/count`.
- **Filtre** : contient les critères de recherche (*motCle*, *méthode*, *ipClient*, *dateDebut*, *dateFin*).

Relations principales :

- Le Serveur utilise le Middleware pour traiter les requêtes.
- Le Middleware crée et enregistre les Logs.
- Le TableauDeBord lit les Logs pour afficher les statistiques.
- EndpointsAPI interagit avec Log et Filtre.

Ainsi, le diagramme illustre le parcours complet : **authentification via GitHub** → **traitement des requêtes** → **enregistrement dans la base** → **affichage des statistiques sur le Dashboard**.

### 3.3.2 Diagramme de classes – modèle de données MongoDB

Le diagramme suivant représente le **modèle de données** utilisé dans la base MongoDB, incluant les collections principales et leurs relations.

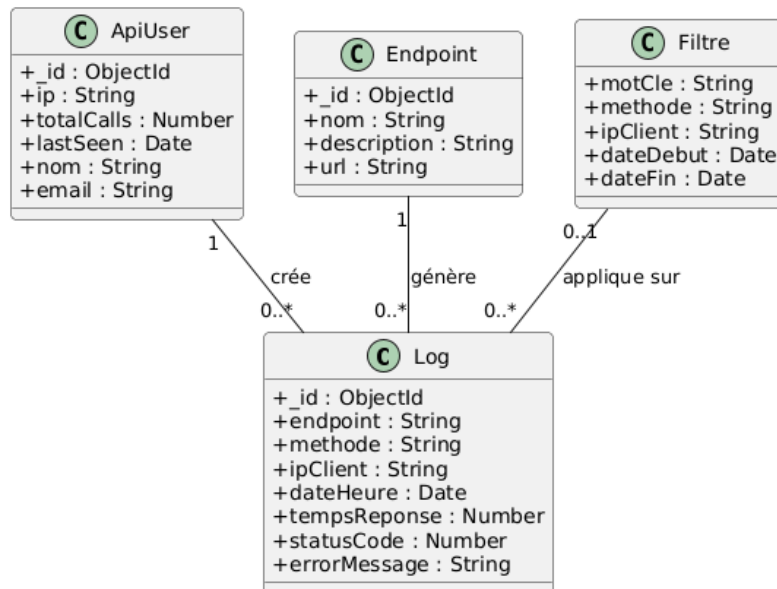


FIGURE 3.3 – Diagramme de classes représentant le modèle de données MongoDB

#### Explications :

- **Log** : collection principale stockant toutes les requêtes entrantes, leurs résultats et erreurs.
- **ApiUser** : représente les utilisateurs de l'API et permet de suivre leur activité (*totalCalls*, *lastSeen*, etc.).
- **Endpoint** : définit chaque endpoint surveillé dans l'API.
- **Filtre** : contient les critères de recherche pour effectuer des requêtes sur les logs (/search).

#### Relations :

- Un **ApiUser** peut générer plusieurs **Logs**.
- Chaque **Endpoint** peut produire plusieurs **Logs**.
- Un **Filtre** peut être appliqué sur plusieurs **Logs** pour des recherches spécifiques.

### 3.4 Diagramme de séquence

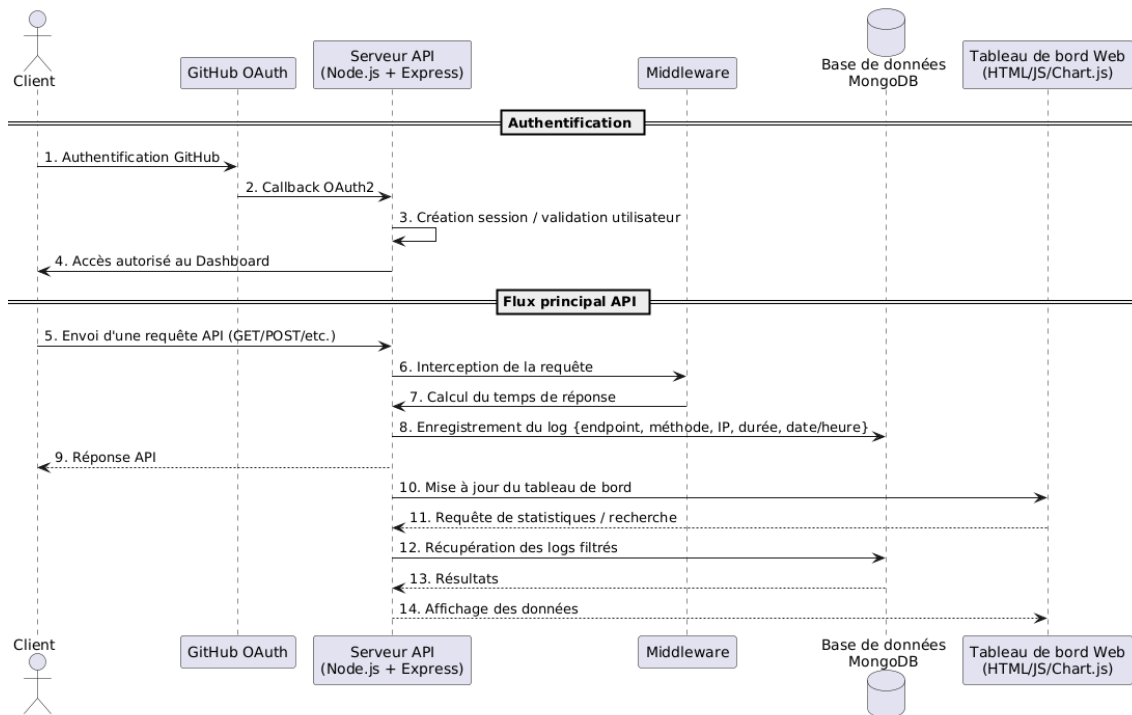


FIGURE 3.4 – Diagramme de séquence

La figure 3.4 présente le déroulement des interactions entre les différents composants du système de supervision d'API.

- **Client** : envoie une requête HTTP (GET, POST, etc.) et initie l'authentification via GitHub pour accéder au Dashboard.
- **GitHub OAuth** : gère l'authentification, renvoie un callback au serveur pour valider l'utilisateur.
- **Serveur API (Node.js + Express)** : reçoit la requête, valide la session utilisateur, la transmet au Middleware pour traitement, et gère l'enregistrement des logs ainsi que la communication avec le Dashboard.
- **Middleware** : calcule le temps de réponse, capture les informations essentielles (endpoint, méthode HTTP, IP du client, durée, date/heure) et les transmet au serveur pour stockage.
- **MongoDB** : enregistre de manière persistante chaque log d'appel API pour permettre consultation et analyse ultérieures.
- **Dashboard Web (HTML/JS/Chart.js)** : récupère les statistiques ou résultats filtrés depuis le serveur et les affiche en temps réel sous forme de graphiques et tableaux.

Ce diagramme met en évidence le parcours complet : **authentification via GitHub** → **traitement des requêtes** → **enregistrement dans MongoDB** → **affichage des statistiques sur le Dashboard**, avec le Middleware jouant un rôle central dans la collecte et l'analyse des données.

## Conclusion

Dans ce chapitre, les principaux diagrammes UML ont été exposés : diagramme de cas d'utilisation, diagramme de classes et diagramme de séquence qui nous aident à présenter l'architecture de notre projet.

Dans le chapitre suivant, une vision globale de la réalisation de l'application sera proposée, en présentant l'environnement de développement, ainsi que les tâches réalisées.

# Chapitre 4

## Implémentation

### Introduction

Le but de ce chapitre est de montrer Le travail effectué. Tout d'abord, nous présentons l'environnement matériel, logiciel et les différents outils de développement utilisés. Deuxièmement, nous illustrons la réalisation de notre travail par des imprimes d'écran des interfaces les plus importantes de notre application.

### 4.1 Environnement de travail

#### 4.1.1 Environnement matériel

L'environnement matériel est détaillé dans le tableau 4.1.

TABLE 4.1 – **Environnement matériel**

| Caractéristique        | Détails                                            |
|------------------------|----------------------------------------------------|
| Ordinateur             | DELL                                               |
| Propriétaire           | Emna Merdessi                                      |
| Processeur             | Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz - 1.80GHz |
| RAM                    | 8 Go                                               |
| Disque dur             | 1 To                                               |
| Système d'exploitation | Windows 10, 64 bits, processeur x64                |
| Carte graphique        | 128 Mo                                             |

### 4.1.2 Environnement logiciel

Le tableau ci-dessous résume les logiciels utilisés pour le développement et le déploiement du projet.

TABLE 4.2 – Environnement logiciel

| Logiciel           | Logo                                                                                | Description                                                                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Visual Studio Code |    | Éditeur de code utilisé pour écrire et organiser le projet (backend et frontend).                                                                           |
| Node.js            |    | Environnement d'exécution JavaScript côté serveur.                                                                                                          |
| Express.js         |    | Framework minimaliste de Node.js pour gérer les routes et les middlewares.                                                                                  |
| MongoDB            |    | Base de données NoSQL orientée documents pour stocker les logs de chaque requête.                                                                           |
| Mongoose           |    | ODM facilitant la manipulation de MongoDB via des schémas et modèles.                                                                                       |
| Postman            |  | Outil pour tester les différentes routes de l'API.                                                                                                          |
| Chart.js           |  | Bibliothèque JavaScript pour afficher les statistiques en graphiques dans le dashboard.                                                                     |
| Git/GitHub         |  | Outil de gestion de versions et d'hébergement du code source du projet, utilisé aussi pour l'authentification afin de sécuriser l'accès au tableau de bord. |
| PlantUML           |  | Outil de création de diagrammes UML, utilisé pour représenter graphiquement l'architecture et les relations entre les composants de l'application.          |

## 4.2 Technologies utilisées

L'API a été développée en utilisant Node.js et Express, en suivant les bonnes pratiques pour les API RESTful [10] :

### Backend

- **Express.js** : Gestion des routes (/stats, /search, /count) et des middlewares.
- **MongoDB** : Base de données NoSQL orientée documents pour stocker les logs de chaque requête.
- **Mongoose** : ODM facilitant la manipulation de MongoDB via des schémas et modèles.
- **Dotenv** : Gestion des variables d'environnement (connexion à la base, port du serveur).
- **CORS** : Middleware pour autoriser les requêtes cross-origin.
- **Passport.js + passport-github2** : authentification GitHub
- **Express Sessions** : gestion sécurisée des sessions utilisateur

### Frontend (Dashboard)

- **HTML5** : Création de la structure du tableau de bord.
- **CSS3** : Mise en forme et design du tableau de bord.
- **JavaScript Vanilla** : Récupération des données depuis l'API (/stats) et manipulation du DOM.
- **Chart.js** : Représentation graphique des données (diagrammes en anneau, courbes, statistiques).

## 4.3 Tâches Réalisées

Au cours de ce projet, plusieurs tâches ont été menées de manière progressive afin d'aboutir à la mise en place d'un système de supervision efficace des appels API.

### Configuration de l'application GitHub

Avant d'utiliser l'authentification GitHub dans le projet, il est nécessaire de configurer une application OAuth sur GitHub. La figure 4.1 montre la page des applications sur GitHub où sont gérés les **Client ID** et **Client Secret**.

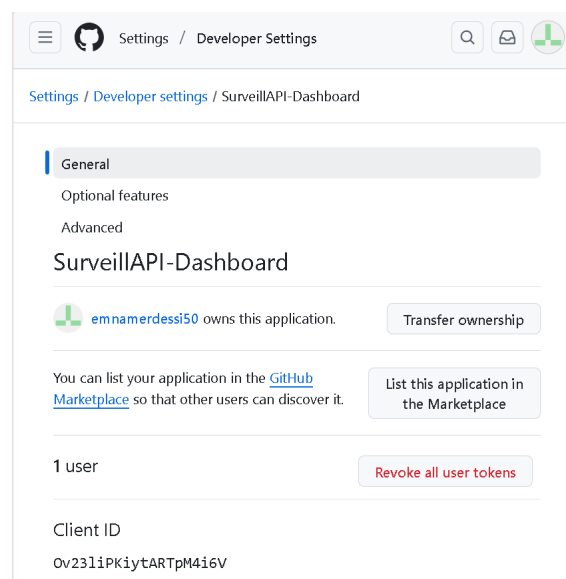


FIGURE 4.1 – Page GitHub : Paramètres des applications OAuth

## Test de l'authentification GitHub

Pour accéder au Dashboard, l'utilisateur doit s'authentifier via GitHub. La figure 4.2 illustre la page de connexion et la redirection vers GitHub.

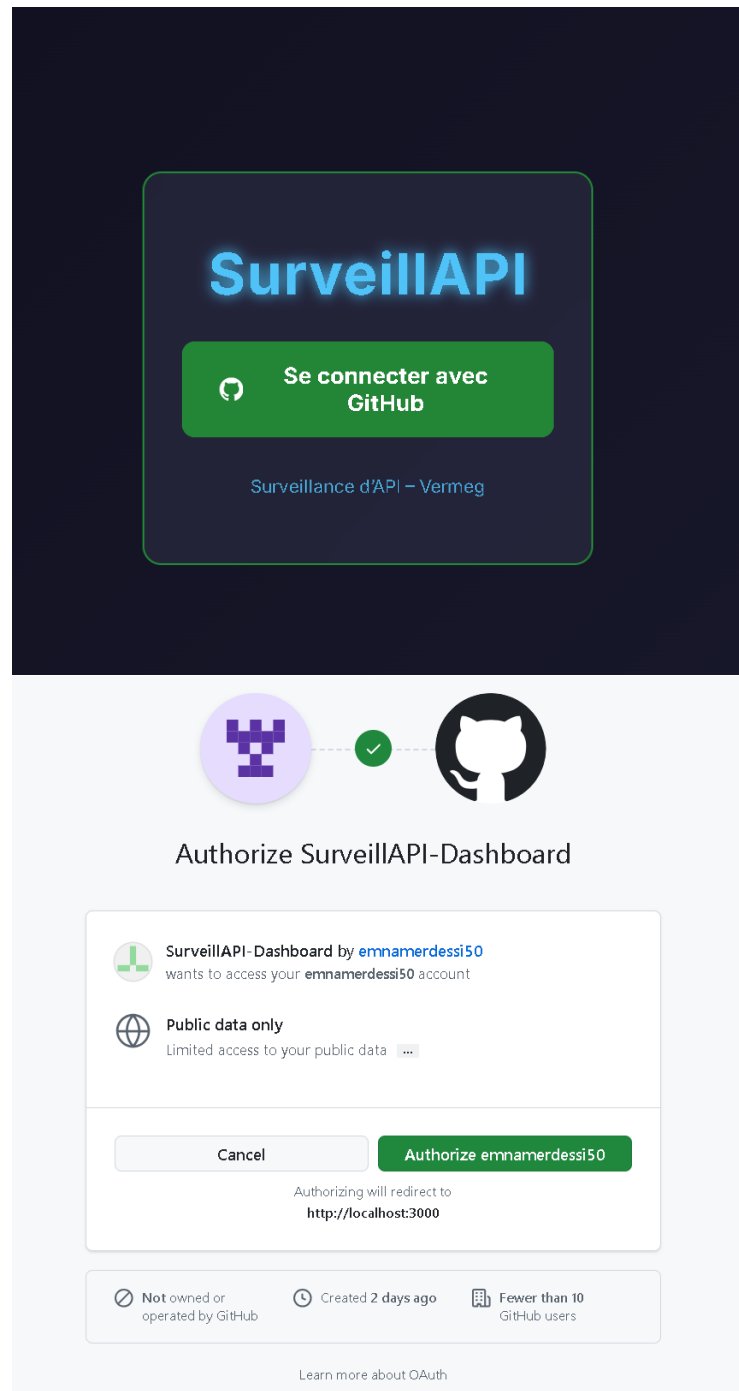


FIGURE 4.2 – Page de connexion GitHub pour accéder au Dashboard

Une fois l'authentification réussie, l'utilisateur est redirigé vers le Dashboard sécurisé :

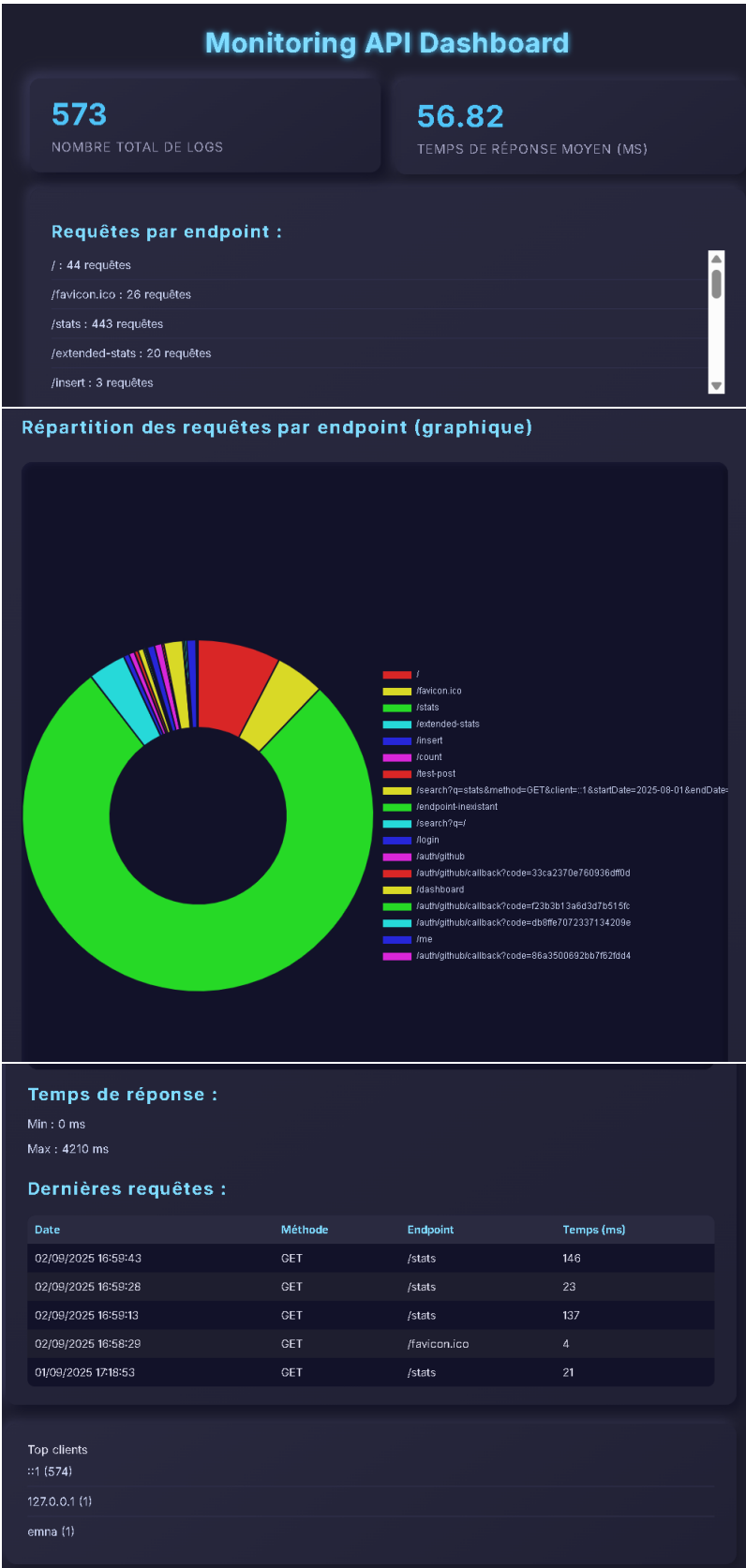


FIGURE 4.3 – Accès au Dashboard après authentification GitHub

## Test des logs

Pour vérifier que les appels API sont bien enregistrés, nous avons effectué une requête correcte via Postman. Le résultat de cette requête a ensuite été confirmé dans MongoDB Compass, où l'enregistrement correspondant est bien apparu dans la base de données.

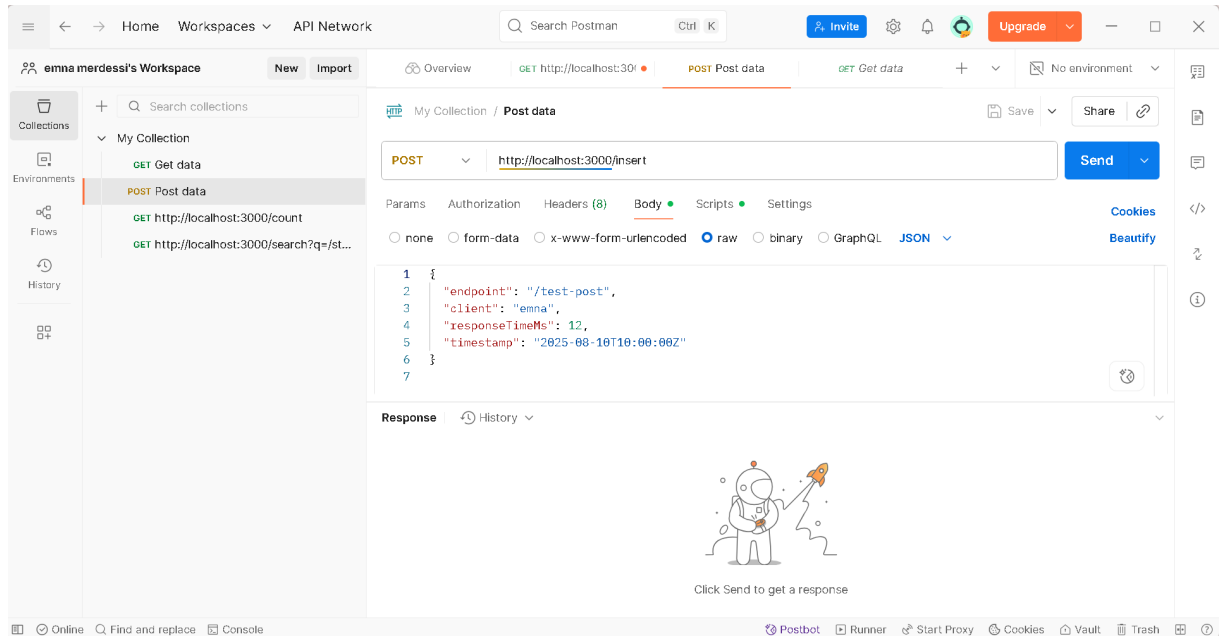


FIGURE 4.4 – Requête faite dans Postman

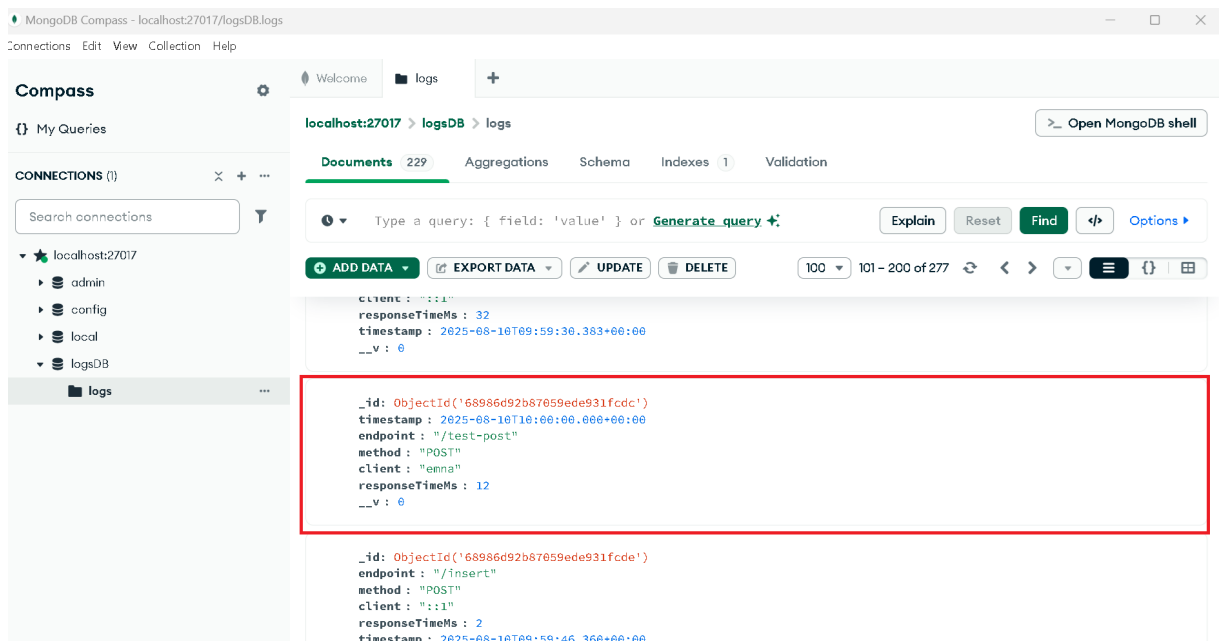


FIGURE 4.5 – Log enregistré dans MongoDB Compass

Chaque appel API est automatiquement enregistré dans la base de données, ce qui permet de suivre toutes les requêtes et d'analyser leur statut, méthode et temps de réponse.

## Test des alertes et anomalies

Pour vérifier le fonctionnement des alertes et du logging des erreurs, nous effectuons une requête incorrecte et observons les résultats.

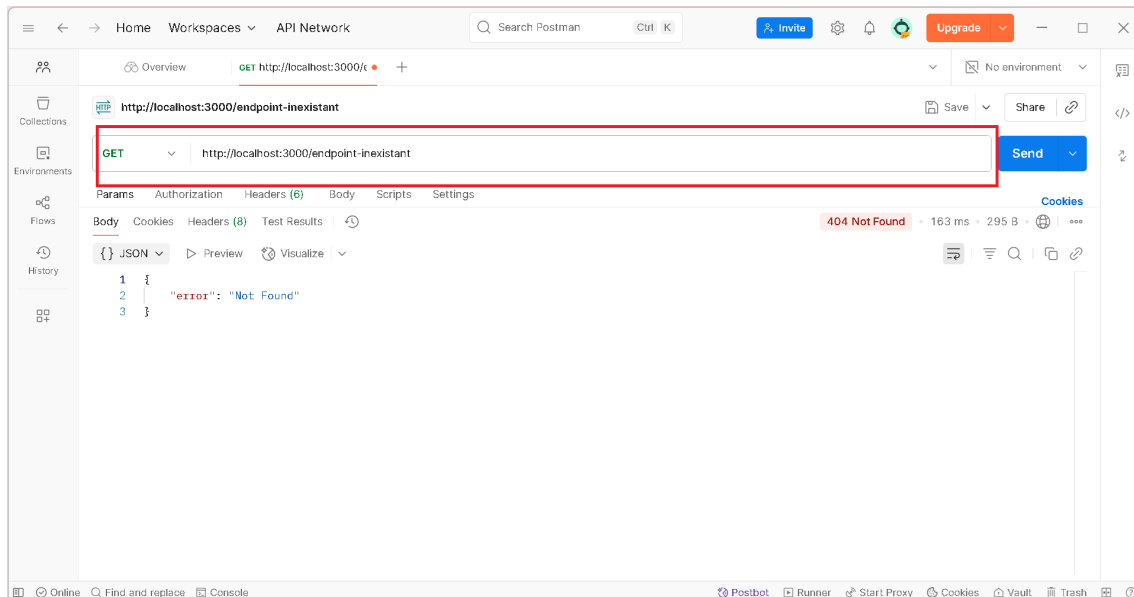


FIGURE 4.6 – Requête incorrecte dans Postman

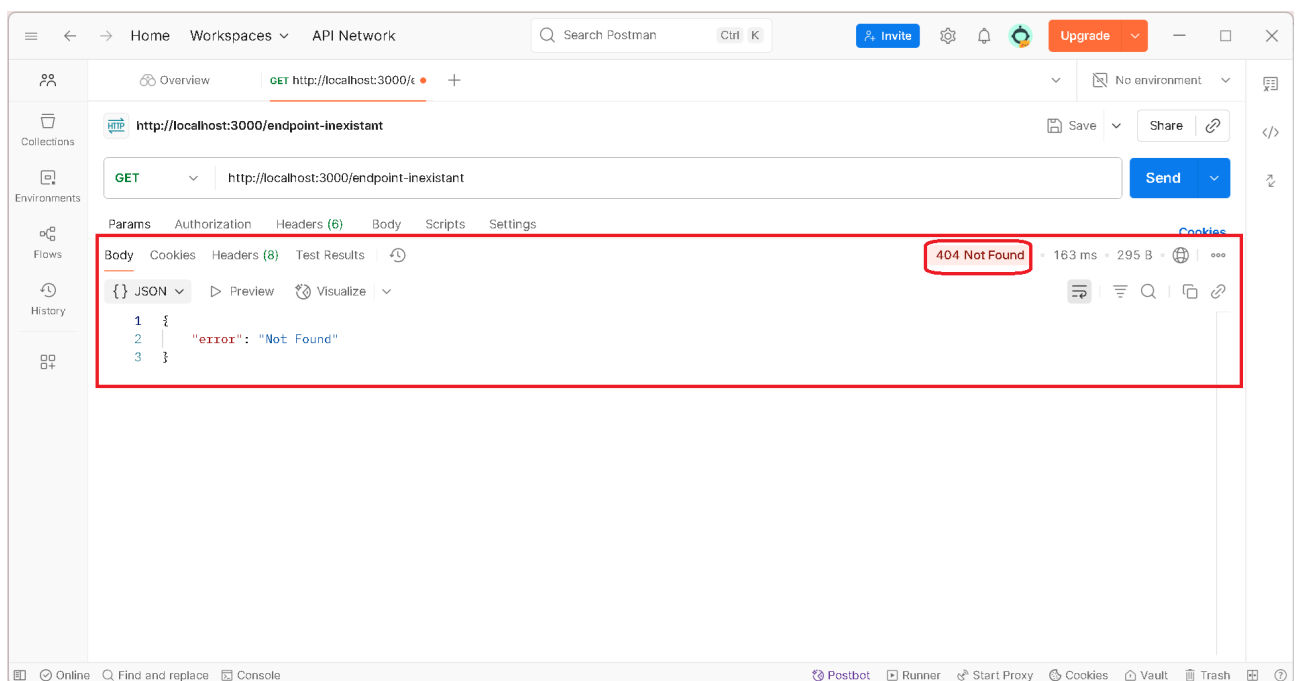


FIGURE 4.7 – Réponse d'erreur (404)

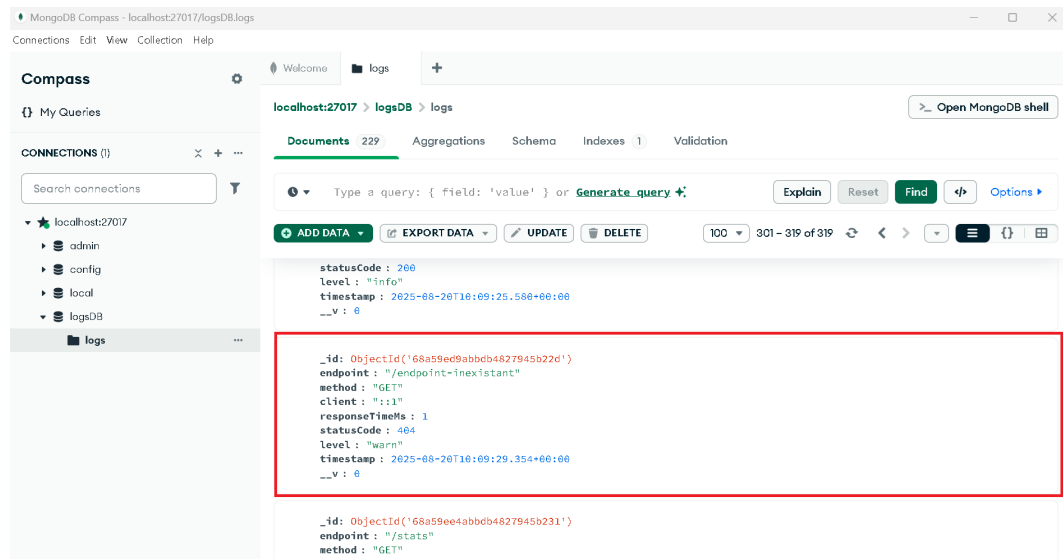


FIGURE 4.8 – Log de l'erreur dans MongoDB Compass

Lorsqu'une requête incorrecte est effectuée, le middleware enregistre automatiquement l'anomalie dans la base de données avec le niveau correspondant : *warn* pour les erreurs 404 et *error* pour les erreurs 500, ce qui permet un suivi et une analyse détaillée des incidents.

## Test des fonctionnalités du Dashboard

Nous avons testé les endpoints et le dashboard de supervision.

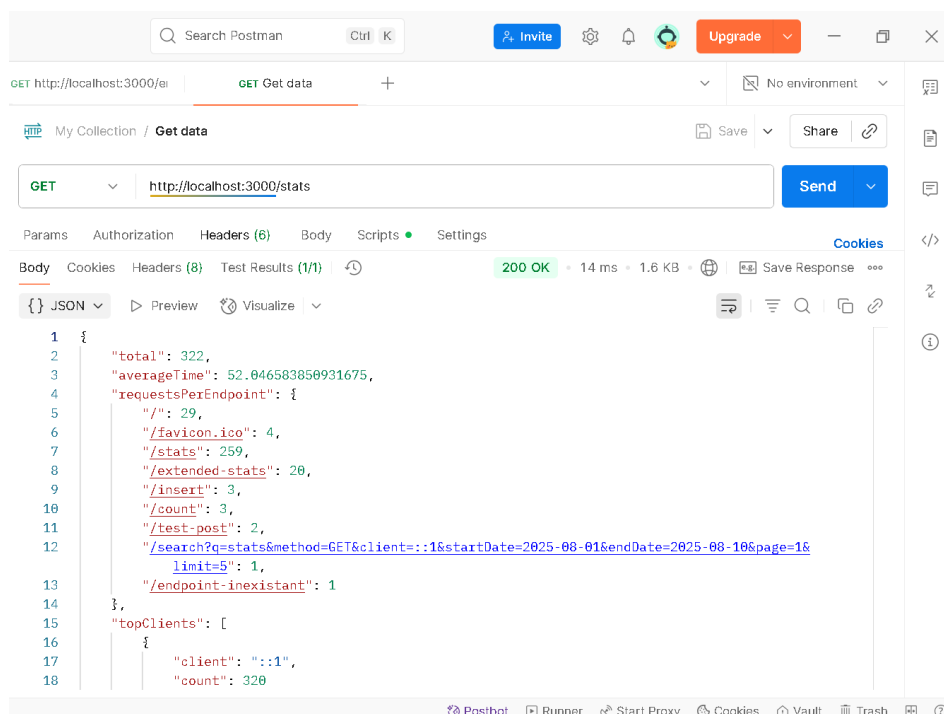


FIGURE 4.9 – Réponse JSON de l'endpoint /stats

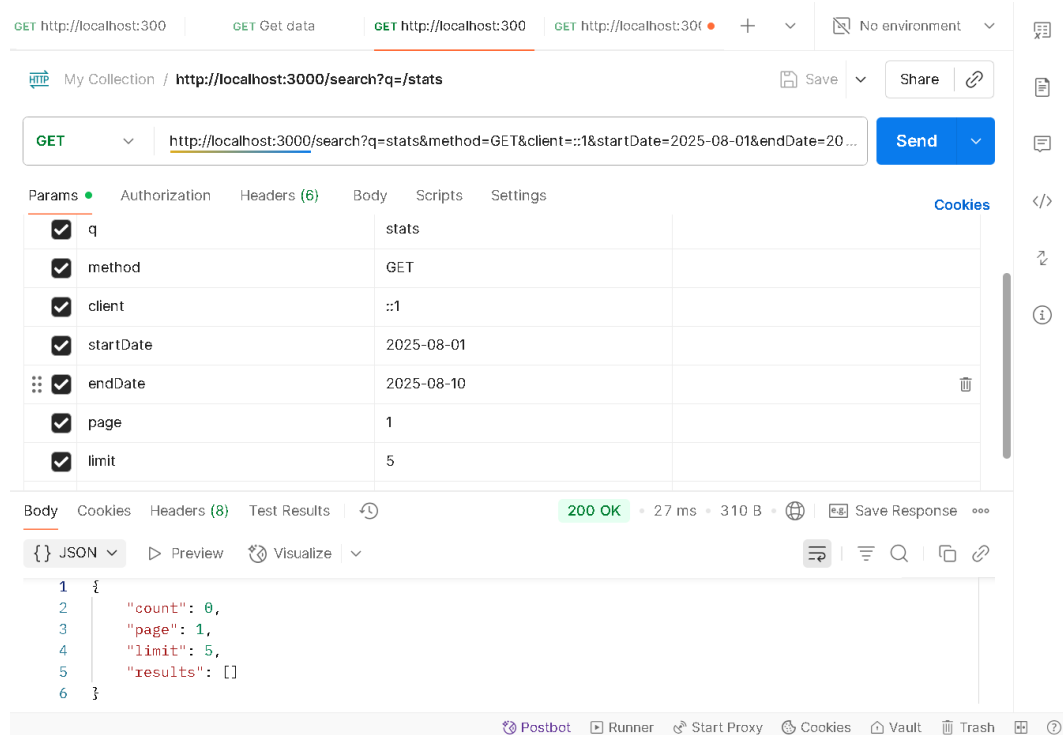


FIGURE 4.10 – Résultat de la recherche avancée /search

Le dashboard récupère automatiquement les logs stockés dans MongoDB et calcule diverses statistiques, telles que le temps moyen de réponse, les endpoints les plus sollicités et les 5 dernières requêtes. La recherche avancée permet de filtrer les logs selon plusieurs critères, et la pagination affiche un nombre limité de résultats pour une consultation facile.

Scannez ce QR code pour accéder directement au dépôt GitHub du projet *SurveillAPI*.

FIGURE 4.11 – QR code du dépôt GitHub du projet *SurveillAPI*

## Conclusion

Pour conclure, nous pouvons distinguer à partir de ce chapitre l'environnement matériel et logiciel que nous avons utilisé pour réaliser notre application. En plus, nous avons clarifié les technologies utilisées pour mener à bien notre travail.

# Conclusion Générale

Ce rapport présente **SurveillAPI**, une application simple et efficace pour superviser les appels d'API. Elle permet de collecter les logs, de stocker les données, d'afficher des statistiques en temps réel et de sécuriser l'accès via **GitHub**.

Le rapport est organisé en quatre parties :

- la présentation de l'entreprise **VERMEG**, du problème étudié et de la solution proposée ;
- la définition des besoins et de l'architecture du système ;
- la modélisation du projet à l'aide de diagrammes **UML** ;
- la description de l'implémentation, des outils utilisés et des tests réalisés.

Plusieurs améliorations sont possibles , telles que l'ajout d'alertes automatiques, la supervision de plusieurs API, la gestion des rôles utilisateurs, l'export des données ou encore le déploiement en ligne.

Certaines fonctionnalités n'ont pas pu être finalisées par manque de temps, notamment les alertes en temps réel, l'export automatique, l'interface d'administration ou les tests de charge.

Avec plus de temps, il aurait été possible d'ajouter une authentification multi-fournisseurs, une analyse intelligente des logs ou encore une version mobile de l'interface.

En résumé, le projet est **fonctionnel, fiable et prêt à évoluer**. Il répond aux objectifs fixés et constitue une base solide pour de futurs développements.

# Netographie

- [1] Vermeg, *Site officiel*. [En ligne]. Disponible sur : <https://www.vermeg.com/> Consulté le 8 août 2025.
- [2] Express.js, *Express.js Guide*. [En ligne]. Disponible sur : <https://expressjs.com/> Consulté le 19 août 2025.
- [3] MongoDB, *MongoDB Documentation*. [En ligne]. Disponible sur : <https://www.mongodb.com/docs/> Consulté le 19 août 2025.
- [4] Postman, *API Platform Documentation*. [En ligne]. Disponible sur : <https://learning.postman.com/> Consulté le 19 août 2025.
- [5] Swagger, *OpenAPI Documentation*. [En ligne]. Disponible sur : <https://swagger.io/docs/> Consulté le 19 août 2025.
- [6] Amazon Web Services, *Amazon CloudWatch*. [En ligne]. Disponible sur : <https://aws.amazon.com/cloudwatch/> Consulté le 19 août 2025.
- [7] Datadog, *Monitoring and Security Platform*. [En ligne]. Disponible sur : <https://www.datadoghq.com/> Consulté le 19 août 2025.
- [8] Chart.js, *Chart.js Documentation*. [En ligne]. Disponible sur : <https://www.chartjs.org/> Consulté le 19 août 2025.
- [9] Passport.js, *Passport.js Documentation*. [En ligne]. Disponible sur : <http://www.passportjs.org/> Consulté le 1 septembre 2025.
- [10] Jaimal Dullat, *A Step-by-Step Guide to Creating a RESTful API Using Node.js and Express (Including CRUD Operations and Authentication)*, Dev.to. [En ligne]. Disponible sur : <https://dev.to/jaimaldullat/a-step-by-step-guide-to-creating-a-restful-api-using-nodejs-and-express-including-crud-operations> Consulté le 1 septembre 2025.

# Annexe A

## Guide d'installation

### Prérequis

Avant d'installer le projet, assurez-vous d'avoir :

- Node.js et npm
- MongoDB
- Compte GitHub valide pour OAuth2

### Cloner le dépôt

```
git clone https://github.com/emnamerdessi50/SurveillAPI.git
cd SurveillAPI
```

### Installer les dépendances

```
npm install
```

### Configurer les variables d'environnement

Créer un fichier `.env` à la racine du projet :

```
1 PORT=3000
2 MONGO_URI=mongodb://localhost:27017/logsDB
3 GITHUB_CLIENT_ID=0v231iPKiytARTpM4i6V
4 GITHUB_CLIENT_SECRET=50a088e6726cc675b8760e3b3dd2ebe531822083
5 SESSION_SECRET=keyboard cat
```

### Explications des variables d'environnement

- `PORT` : port du serveur
- `MONGO_URI` : URI pour se connecter à MongoDB
- `GITHUB_CLIENT_ID` et `GITHUB_CLIENT_SECRET` : identifiant et secret pour OAuth2
- `SESSION_SECRET` : clé pour sécuriser la session Express

### Configurer GitHub OAuth2

1. Aller sur GitHub Developer Settings
2. Créer une New OAuth App :
  - Application Name : SurveillAPI

- Homepage URL : `http://localhost:3000`
  - Authorization callback URL : `http://localhost:3000/auth/github/callback`
3. Copier le Client ID et Client Secret dans `.env`

## Lancer le serveur

`npm start`

Accéder au Dashboard : `http://localhost:3000`

## Tester avec Postman

Endpoints principaux :

- `GET /stats` : statistiques globales
- `GET /search` : recherche avancée des logs
- `GET /count` : nombre total de logs

## Notes importantes

- Chaque appel API est enregistré automatiquement dans MongoDB
- Les erreurs (404, 500) sont loguées avec le niveau correspondant (`warn` ou `error`)
- Le Dashboard affiche les statistiques en temps réel avec graphiques et tableaux

## Annexe B

### Mini-extrait de code

#### Middleware principal (logger.js)

```
1 const Log = require("../models/LogModel");
2 function logger(req, res, next) {
3   const start = Date.now();
4   res.on("finish", async () => {
5     const duration = Date.now() - start;
6     let level = "info";
7     if (res.statusCode >= 400 && res.statusCode < 500) level = "warn";
8     else if (res.statusCode >= 500) level = "error";
9     const log = new Log({
10      endpoint: req.originalUrl,
11      method: req.method,
12      client: req.ip,
13      responseTimeMs: duration,
14      statusCode: res.statusCode,
15      level: level
16    });
17    try {
18      await log.save();
19      console.log(`[${level.toUpperCase()}] ${req.method} ${req.originalUrl}
20        } -> ${res.statusCode} (${duration}ms)`);
21    } catch (err) { console.error("Erreur lors de l'enregistrement du log
22      :", err); }
23  });
24  next();
25 }
26 module.exports = logger;
```

#### Schéma Mongoose (LogModel.js)

```
1 const mongoose = require('mongoose');
2 const logSchema = new mongoose.Schema({
3   timestamp: { type: Date, default: Date.now },
4   endpoint: { type: String, required: true },
5   method: { type: String, required: true },
6   client: { type: String, default: "unknown" },
7   responseTimeMs: { type: Number },
8   statusCode: { type: Number, required: true },
9   level: { type: String, enum: ["info", "warn", "error"], default: "info" }
10 });
11 module.exports = mongoose.model('Log', logSchema);
```

## Endpoint /stats (extrait)

```
1 app.get('/stats', async (req, res) => {
2   try {
3     const logs = await Log.find().lean();
4     const total = logs.length;
5     const averageTime = total > 0 ? logs.reduce((sum, l) => sum + (l.
      responseTimeMs || 0), 0) / total : 0;
6     const requestsPerEndpoint = {};
7     logs.forEach(l => { requestsPerEndpoint[l.endpoint] = (
      requestsPerEndpoint[l.endpoint] || 0) + 1; });
8     const clientsMap = {};
9     logs.forEach(l => { clientsMap[l.client] = (clientsMap[l.client] || 0)
      + 1; });
10    const topClients = Object.entries(clientsMap)
11      .map(([client, count]) => ({ client, count }))
12      .sort((a, b) => b.count - a.count)
13      .slice(0, 5);
14    const minResponseTime = total > 0 ? Math.min(...logs.map(l => l.
      responseTimeMs || 0)) : 0;
15    const maxResponseTime = total > 0 ? Math.max(...logs.map(l => l.
      responseTimeMs || 0)) : 0;
16    const last5Requests = await Log.find().sort({ timestamp: -1 }).limit(5)
      .lean();
17    res.json({ total, averageTime, requestsPerEndpoint, topClients,
      minResponseTime, maxResponseTime, last5Requests });
18  } catch (err) { console.error(err); res.status(500).send('Error fetching
      stats'); }
19 });
```

## Annexe C

### Lien vers le dépôt GitHub

`https://github.com/emnamerdessi50/SurveillAPI`