SYSTÈME DE FEEDBACK SUR DES PRODUITS NUMÉRIQUES

Rapport Technique

Réalisé par :

Rayen Ouechrai **Amine Ben Selem**
Sami Ben Slimen **Mohamed Aziz Merai**

Tek-UP

Année universitaire 2024–2025

June 10, 2025

Table des matières

1 Solution proposée

1.1 Analyse du problème

L'entreprise souhaite automatiser la collecte et l'exploitation des retours utilisateurs sur ses produits numériques (SaaS). Actuellement, aucun système structuré ne permet de centraliser ou d'analyser les avis utilisateurs.

1.2 Identification des Entités & Relations

Entités principales (MongoDB) :

— **Feedback** :
  — `id` : Identifiant unique.
  — `name` : Nom de l'utilisateur.
  — `email` : Email.
  — `feedbackType` : Type (ex. : bug, fonctionnalité).
  — `service` : Service évalué.
  — `message` : Commentaire.
  — `rating` : Note (entier).
  — `attachScreenshot` : Capture d'écran.
  — `agreeToTerms` : Accord des conditions.
— **Service** :
  — `id` : Identifiant.
  — `name` : Nom.

Relations :

— `Feedback.service` référence `Service.name`.
— Les analyses agrègent les `Feedback` par service ou globalement.

1.3 Liste des fonctionnalités (sous forme de Web Service)

— Ajouter un feedback
— Approuver automatiquement un feedback (modération)
— Obtenir une analyse globale
— Obtenir une analyse par service

2 Diagrammes

2.1 Diagramme de classes / entités

Le diagramme suivant représente les principales entités du système et leurs relations. On y distingue les entités `Feedback`, `Service`, et les objets d'analyse.

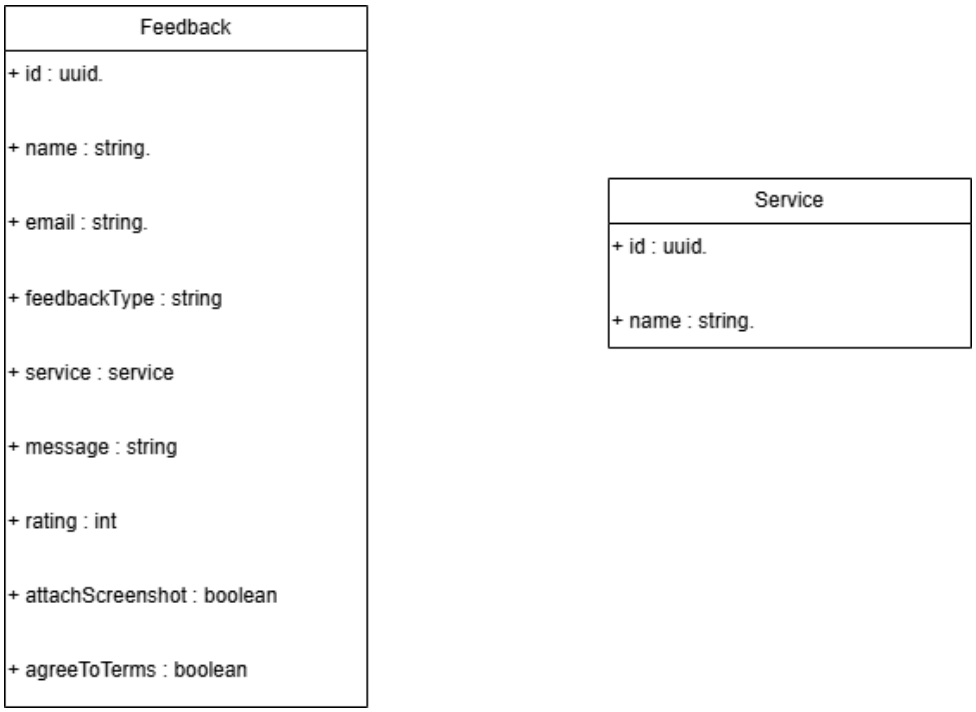

FIGURE 1 – Diagramme de classes / entités

2.2 Schéma des requêtes et réponses (GraphQL)

Ce schéma illustre le fonctionnement global du système à travers les types de requêtes GraphQL disponibles, leurs paramètres, et les structures de réponse attendues.

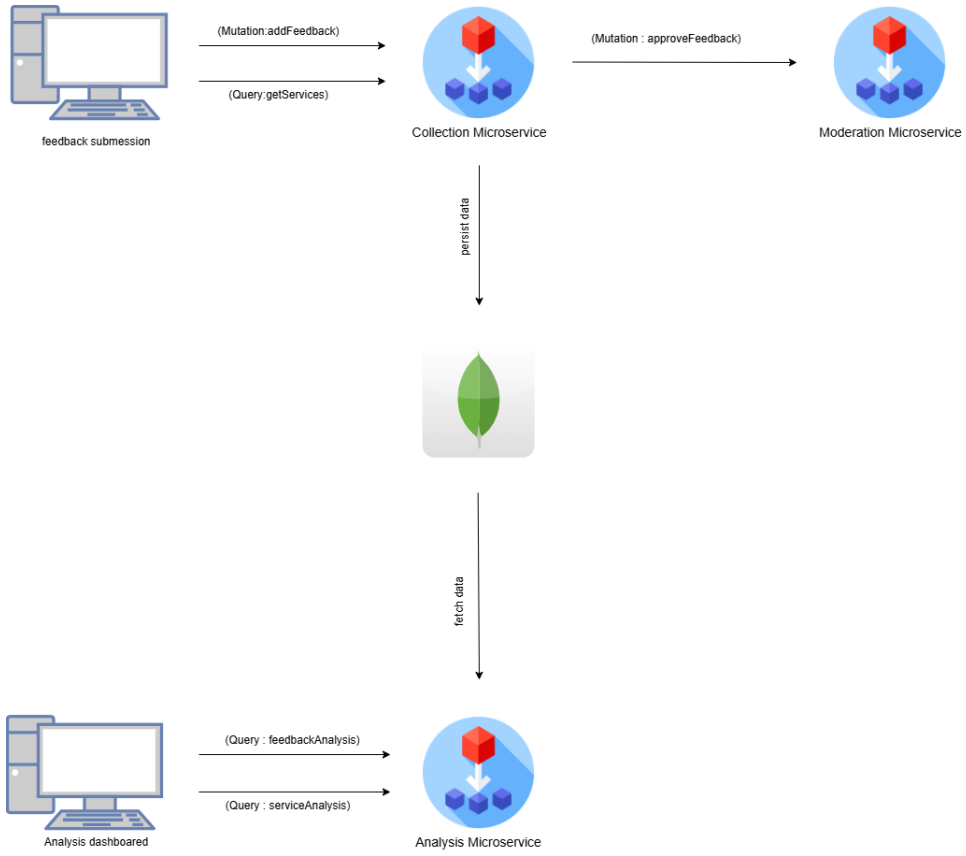

FIGURE 2 – Schéma des requêtes et réponses (GraphQL)

3 Implémentation Technique

3.1 Web Service Fonctionnel

Trois microservices GraphQL :

— **Collecte** : NestJS, soumission d'avis et liste des services.
— **Modération** : Go, validation des commentaires.
— **Analyse** : FastAPI, analyses des avis.

Le frontend Angular consomme les APIs via requêtes HTTP GraphQL.

3.2 Données en Mémoire ou Base Simple

MongoDB stocke :

— `Feedback` : Avis avec tous les champs GraphQL.
— `Service` : Services évalués.

3.3 Structure Propre

Organisation :

— **Collecte** : NestJS avec modules GraphQL et MongoDB.
— **Modération** : Go avec résolveurs GraphQL.

— **Analyse** : FastAPI avec GraphQL et MongoDB.
— **Frontend** : Angular avec Apollo Client.

4 Documentation

4.1 Description Claire des Schémas GraphQL

4.1.1 Collecte (NestJS)

Soumission d'avis et liste des services.

```
input CreateFeedbackInput {
  agreeToTerms: Boolean!
  attachScreenshot: Boolean!
  email: String!
  feedbackType: String!
  message: String!
  name: String!
  rating: Int!
  service: String!
}
type Feedback {
  agreeToTerms: Boolean!
  attachScreenshot: Boolean!
  email: String!
  feedbackType: String!
  message: String!
  name: String!
  rating: Int!
  service: String!
}
type Service {
  id: ID!
  name: String!
}
type Mutation {
  addFeedback(createFeedbackInput: CreateFeedbackInput!): Feedback!
}
type Query {
  getServices: [Service!]!
}
```

4.1.2 Modération Automatique (Go)

Validation des commentaires.

```
input FeedbackInput {
  name: String!
  email: String!
  feedbackType: String!
  service: String!
```

```
  message: String!
  rating: Int!
  attachScreenshot: Boolean!
  agreeToTerms: Boolean!
}
type ApprovalResponse {
  approved: Boolean!
  reason: String
}
type Mutation {
  approveFeedback(feedback: FeedbackInput!): ApprovalResponse!
}
```

4.1.3 Analyse (FastAPI)

Analyses des avis.

```
scalar DateTime
type Feedback {
  id: String!
  name: String!
  email: String!
  feedbackType: String!
  service: String!
  message: String!
  rating: Int!
  attachScreenshot: Boolean!
  agreeToTerms: Boolean!
  createdAt: DateTime!
  updatedAt: DateTime!
  sentiment: String!
  sentimentScore: Float!
  topKeywords: [Keyword!]!
}
type FeedbackAnalysis {
  averageRating: Float!
  totalFeedback: Int!
  feedbackTypeCounts: [KeyValuePair!]!
  sentimentCounts: [KeyValuePair!]!
}
type KeyValuePair {
  key: String!
  value: Int!
}
type Keyword {
  word: String!
  frequency: Int!
}
type SentimentBreakdown {
  positive: Int!
  neutral: Int!
```

```
  negative: Int!
}
type ServiceAnalysis {
  service: String!
  totalFeedback: Int!
  averageRating: Float!
  averageSentiment: Float!
  sentimentBreakdown: SentimentBreakdown!
  topKeywords: [String!]!
}
type Query {
  feedbacks(limit: Int! = 10, skip: Int! = 0): [Feedback!]!
  feedbackById(id: String!): Feedback
  feedbackAnalysis: FeedbackAnalysis!
  serviceAnalysis(service: String!): ServiceAnalysis!
}
```

4.2 Exemples d'Utilisation

4.2.1 Collecte : Mutation addFeedback

```
mutation {
  addFeedback(createFeedbackInput: {
    name: "Jean Dupont"
    email: "jean@example.com"
    feedbackType: "feature"
    service: "API"
    message: "Ajoutez plus de filtres."
    rating: 4
    attachScreenshot: false
    agreeToTerms: true
  }) {
    name
    email
    feedbackType
    service
    message
    rating
  }
}
```

Réponse :

```
{
  "data": {
    "addFeedback": {
      "name": "Jean Dupont",
      "email": "jean@example.com",
      "feedbackType": "feature",
      "service": "API",
      "message": "Ajoutez plus de filtres.",
```

```
      "rating": 4
    }
  }
}
```

4.2.2 Modération : Mutation approveFeedback

```
mutation {
  approveFeedback(feedback: {
    name: "Jean Dupont"
    email: "jean@example.com"
    feedbackType: "feature"
    service: "API"
    message: "Ajoutez plus de filtres."
    rating: 4
    attachScreenshot: false
    agreeToTerms: true
  }) {
    approved
    reason
  }
}
```

Réponse :

```
{
  "data": {
    "approveFeedback": {
      "approved": true,
      "reason": null
    }
  }
}
```

4.2.3 Analyse : Query serviceAnalysis

```
query {
  serviceAnalysis(service: "API") {
    service
    totalFeedback
    averageRating
    averageSentiment
    sentimentBreakdown {
      positive
      neutral
      negative
    }
    topKeywords
  }
}
```

Réponse :


```
{
  "data": {
    "serviceAnalysis": {
      "service": "API",
      "totalFeedback": 50,
      "averageRating": 4.2,
      "averageSentiment": 0.75,
      "sentimentBreakdown": {
        "positive": 35,
        "neutral": 10,
        "negative": 5
      },
      "topKeywords": ["filtres", "performance", "interface"]
    }
  }
}
```


5 Conclusion

Le système développé offre une solution complète et efficace pour la collecte, la modération et l'analyse des avis utilisateurs à travers une architecture microservices basée sur GraphQL. Cette approche apporte plusieurs avantages :

— Une séparation claire des responsabilités entre les différents services
— Une flexibilité d'évolution pour chaque composant indépendamment
— Des performances optimisées grâce à l'utilisation de technologies adaptées à chaque besoin spécifique
— Une facilité d'intégration avec le frontend Angular via Apollo Client

5.1 Suggestions d'Améliorations Futures

Pour enrichir et perfectionner ce système, plusieurs pistes d'amélioration peuvent être envisagées :

— **Sécurité renforcée** : Implémentation d'un système d'authentification JWT pour sécuriser l'accès aux APIs GraphQL et ajout de validation avancée des données entrantes.
— **Performance** : Mise en place d'un système de cache avec Redis pour les requêtes d'analyse fréquentes et optimisation des requêtes MongoDB avec indexation appropriée.
— **Évolutivité** : Conteneurisation via Docker et orchestration avec Kubernetes pour permettre une mise à l'échelle automatique selon la charge.
— **Modération avancée** : Intégration d'un système de modération basé sur l'IA pour détecter automatiquement les contenus inappropriés et les spams.
— **Analyse enrichie** : Développement d'algorithmes d'analyse sémantique plus sophistiqués pour extraire des insights plus précis et pertinents des commentaires utilisateurs.

— **Intégration continue** : Mise en place d'un pipeline CI/CD complet pour automatiser les tests, le déploiement et la surveillance du système.
— **Expérience utilisateur** : Implémentation de notifications en temps réel pour informer les administrateurs des nouveaux feedbacks et amélioration du tableau de bord d'analyse avec des visualisations interactives.
— **Multi-langues** : Support de l'analyse des sentiments et du traitement des feedbacks dans plusieurs langues pour une portée internationale.

Ces améliorations permettraient de faire évoluer le système vers une solution encore plus robuste, scalable et riche en fonctionnalités, capable de répondre aux besoins croissants de l'entreprise en matière d'analyse des retours utilisateurs.