# 

# Rapport Technique : API de Messagerie avec Node.js et Express.js

**Réaliser par :** Belaroussia Nabila

Date : 27/12/2024

Sommaire

[Rapport Technique : API de Messagerie avec Node.js et Express.js 1](#_Toc186197082)

[Introduction : 3](#_Toc186197083)

[Objectif : 3](#_Toc186197084)

[Choix des Technologies : 4](#_Toc186197085)

[Structure de l'API 5](#_Toc186197086)

[i. Routes Principales 5](#_Toc186197087)

[ii. Fonctionnalités Clés 5](#_Toc186197088)

[Schéma de la Base de Données 5](#_Toc186197089)

[1. Tables Principales 5](#_Toc186197090)

[Justification du Choix de la Base de Données 6](#_Toc186197091)

[1. Base de Données SQL (MySQL) 6](#_Toc186197092)

[Architecture du projet 7](#_Toc186197093)

[iii. 7](#_Toc186197094)

[1. Structure des dossiers 7](#_Toc186197095)

[2. Description des fichiers 7](#_Toc186197096)

[Instructions pour tester l'API 8](#_Toc186197097)

[iv. Prérequis 8](#_Toc186197098)

[v. Étapes 8](#_Toc186197099)

[Fonctionnalités principales 9](#_Toc186197100)

[1. Envoi de messages 9](#_Toc186197101)

[2. Consultation de l'historique 9](#_Toc186197102)

[3. Mise à jour du statut d'un message 10](#_Toc186197103)

[Bonnes pratiques 10](#_Toc186197104)

[Conclusion 11](#_Toc186197105)

# Introduction :

Ce document détaille la conception, le développement et le fonctionnement d'une API de messagerie permettant la communication entre enseignants et élèves. L'API inclut des fonctionnalités pour envoyer, consulter et gérer les messages.

# Objectif :

L'objectif est de créer une API backend qui offre les fonctionnalités suivantes :

1. Les enseignants peuvent envoyer des messages à des élèves ou groupes d'élèves.
2. Les élèves peuvent répondre à des messages ou poser des questions.
3. Tous les utilisateurs peuvent consulter l'historique des messages avec des filtres (par destinataire, date ou statut).
4. Les utilisateurs peuvent marquer les messages comme lu/non lu.

# Choix des Technologies :

Pourquoi Node.js et Express.js ?

**Performance et Scalabilité :**

* + Node.js est basé sur un moteur JavaScript V8 très rapide.
  + Il est particulièrement adapté pour gérer un grand nombre de connexions simultanées grâce à son architecture non-bloquante (event-driven).

**Facilité de développement :**

* + Express.js est un framework minimaliste qui simplifie la création de routes, middleware et gestion des requêtes HTTP.
  + Une grande communauté et une abondance de bibliothèques facilitent le développement rapide et efficace.

**Adapté aux applications en temps réel :**

* + Avec Node.js facilite le développement d'applications nécessitant des fonctionnalités en temps réel (par exemple, notifications pour les nouveaux messages).

**Écosystème riche :**

* + NPM offre un vaste choix de modules pour intégrer des fonctionnalités comme l'authentification, la gestion des bases de données et la validation des données.

# Structure de l'API

## Routes Principales

* + **/api/messages** : Gère l'envoi, la récupération et la mise à jour des messages.
  + **/api/users** : Permet de récupérer les informations des utilisateurs.

## Fonctionnalités Clés

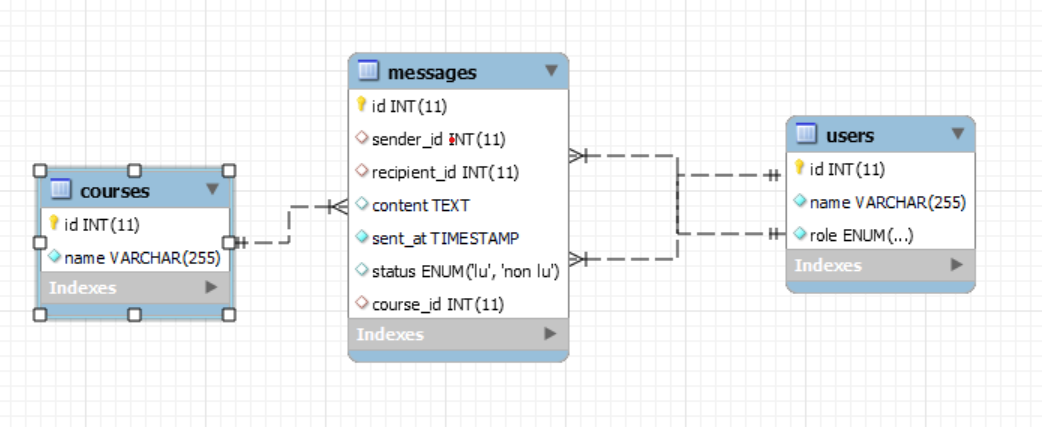
1. **Envoi de messages :**
   * Les enseignants peuvent envoyer des messages aux élèves.
   * Les élèves peuvent répondre ou poser des questions.
2. **Consultation de l'historique :**
   * Historique des messages échangés avec des options de filtrage (par destinataire, date, statut).
3. **Gestion du statut :**
   * Permet de marquer les messages comme lus ou non lus.

# Schéma de la Base de Données

## Tables Principales

1. **Users :**
   * id : Identifiant unique de l'utilisateur.
   * name : Nom de l'utilisateur.
   * role : Rôle de l'utilisateur (enseignant ou élève).
2. **Messages :**
   * id : Identifiant unique du message.
   * sender\_id : Référence à l'utilisateur ayant envoyé le message.
   * recipient\_id : Référence à l'utilisateur destinataire.
   * content : Contenu du message.
   * course\_id : Référence au cours (optionnel).
   * status : Statut du message (lu/non lu).
   * sent\_at : Date et heure d'envoi du message.
3. **Courses**
   * id : Identifiant unique du cours.
   * name : Nom du cours.

* **Voici le schéma de diagramme de classe :**



# Justification du Choix de la Base de Données

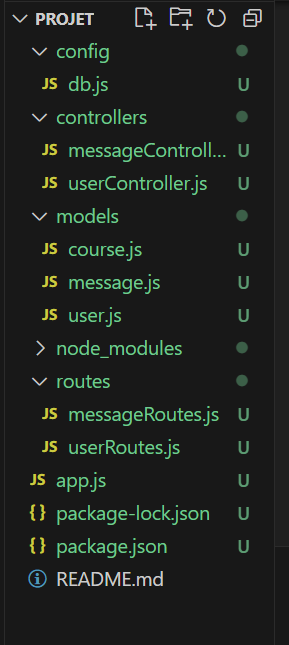
## Base de Données SQL (MySQL)

1. **Relations bien définies :**
   * Les messages sont fortement liés aux utilisateurs et éventuellement aux cours. MySQL permet une modélisation relationnelle claire avec des clés étrangères.
2. **Intégrité des données :**
   * Garantit la cohérence grâce à des contraintes (par exemple, éviter les messages orphelins).
3. **Requêtes optimisées :**
   * MySQL offre des outils performants pour les requêtes complexes (joins, filtres).
4. **Évolutivité verticale :**
   * Adapté pour gérer des millions de messages avec des optimisations d'indexation.

# Architecture du projet

## 

## Structure des dossiers



## Description des fichiers

* **app.js** : Point d'entrée principal, configure les middlewares et routes.
* **config/db.js** : Gestion de la connexion à la base de données MySQL.
* **controllers/** : Contient les contrôleurs pour gérer les requêtes.
* **models/** : Modélisation des entités avec des fonctions pour interagir avec la base de données.
* **routes/** : Définition des routes pour les fonctionnalités de l'API.

# Instructions pour tester l'API

## Prérequis

* Node.js et npm installés.
* Base de données MySQL configurée.

## Étapes

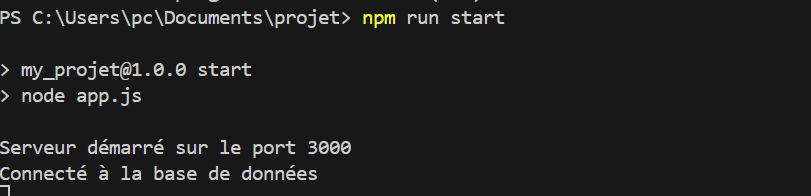
Installez les dépendances :



Configurez la base de données :

* Créez une base de données nommée my\_projet.
* Exécutez le script SQL pour créer les tables.

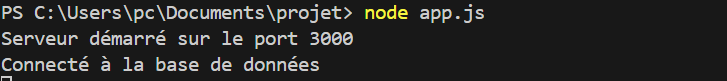
Lancer le serveur :



Ou



Ou



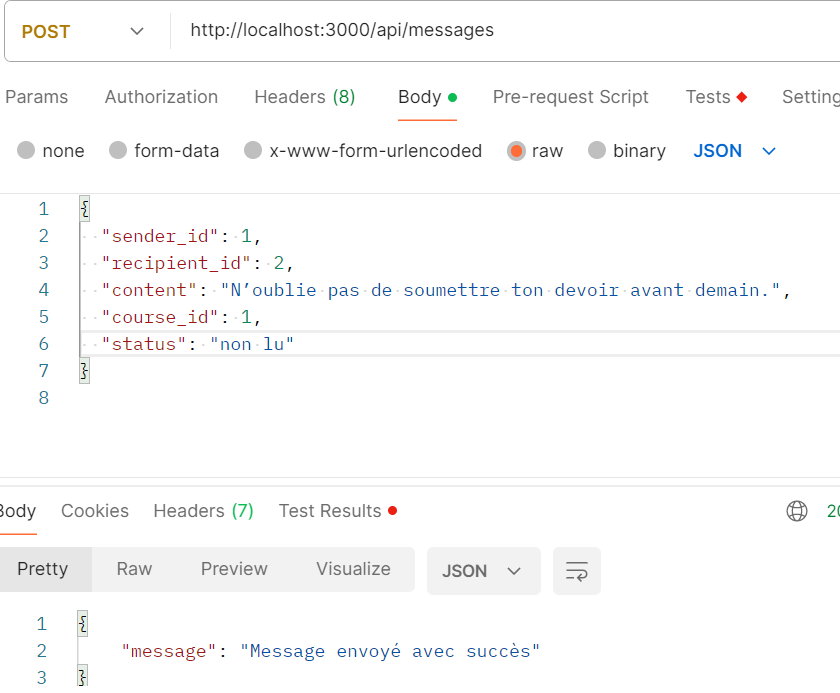
Tester les routes :

Postman pour interagir avec l'API.

# Fonctionnalités principales

## Envoi de messages

* **Route** : POST /api/messages
* **Description** : Permet à un enseignant ou élève d'envoyer un message.
* **Exemple** :



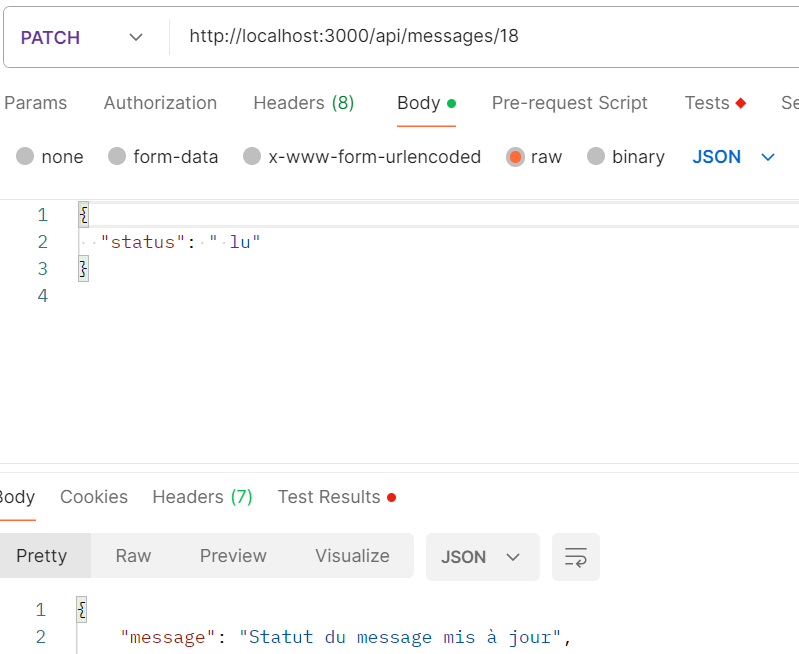
## Consultation de l'historique

* **Route** : GET /api/messages
* **Description** : Permet de consulter les messages avec des filtres.
* **Filtres disponibles** :
  + sender\_id
  + recipient\_id
  + date\_from, date\_to
  + status
* **Exemple de requête** :

http://localhost:3000/api/messages?sender\_id=1&recipient\_id=2&date\_from=2024-12-01&date\_to=2024-12-25&status=unread&limit=10&offset=0

## Mise à jour du statut d'un message

* **Route** : PATCH /api/messages/:id
* **Description** : Permet de marquer un message comme lu ou non lu.
* **Payload exemple** :



# Bonnes pratiques

* **Validation des données** : Les entrées utilisateur sont validées dans les contrôleurs pour éviter les injections SQL.
* **Gestion des erreurs** : Codes d'erreur standardisés pour une meilleure clarté.
* **Pagination** : Implémentée pour les grandes quantités de messages.
* **Modularité** : Séparation claire entre modèles, contrôleurs et routes.

# Conclusion

*L'API est conçue pour répondre efficacement aux besoins de messagerie entre enseignants et élèves, avec une structure claire et des performances optimales grâce à MySQL. Les bonnes pratiques garantissent une extensibilité et une maintenance aisée.*