**DEPARTEMENT MATHEMATIQUES ET INFORMATIQUE**

**Filière Ingénieur :**

**« Ingénierie Informatique : Big Data et Cloud Computing »**

*II-BDCC 1*

**Rapport du Projet de Fin de Module**



Application de Blogging basée sur Node.js et Express : une plateforme web pour les blogs



**Encadré par :**

Pr Abdelaziz DAAIF

**Réaliser par :**

Nouhayla MOUAKKAL

Table of Contents

**Type chapter title (level 1)1**

Type chapter title (level 2)2

Type chapter title (level 3)3

**Type chapter title (level 1)4**

Type chapter title (level 2)5

Type chapter title (level 3)6

# Introduction :

Avec la montée en puissance du développement web et l'expansion continue des besoins, Node.js s'est établi comme un pilier incontestable pour la création d'applications cross-plateformes. Que ce soit pour le développement d'applications mobiles avec des Framework tels que Ionic ou la conception d'applications pour ordinateurs portables grâce à Electron, Node.js offre une polyvalence incomparable. En outre, il est largement utilisé dans le domaine du développement de serveurs web et la mise en place d'API REST. Face à cette demande croissante, les développeurs se tournent vers des Framework tels qu'Express.js, qui leur permettent de construire rapidement des bases solides pour leurs applications tout en favorisant une collaboration harmonieuse et le respect de normes communes. Ainsi, ces Framework offrent une solution efficace pour accélérer le processus de développement, répondre aux exigences croissantes et offrir des résultats de qualité supérieure.

# Présentation du projet :

* Objectif :

L'objectif de ce projet est de créer une plateforme de blogs permettant aux utilisateurs d'afficher et de gérer des articles liés à différentes catégories. Les blogs seront accessibles au public, offrant ainsi une plateforme de partage d'informations et d'interactions.

* Description :

Ce projet vise à fournir une solution complète pour les blogs, offrant des fonctionnalités spécifiques en fonction du rôle de chaque utilisateur. Les utilisateurs peuvent être des administrateurs, des utilisateurs normaux ou des blogueurs.

En tant qu'administrateur, l'utilisateur aura le pouvoir de gérer les autres utilisateurs, de gérer les articles existants ainsi que les catégories. Cette fonctionnalité permet de maintenir l'ordre et de garantir que le contenu du blog reste pertinent et de haute qualité.

Les utilisateurs normaux auront la possibilité de consulter les blogs, de lire les articles et d'ajouter des commentaires. Cela favorise l'engagement de la communauté en permettant aux utilisateurs de donner leur avis, de poser des questions ou de partager des idées.

Les blogueurs, quant à eux, auront des privilèges supplémentaires. Ils pourront créer leurs propres articles et les gérer, mais uniquement pour les articles qu'ils ont créés. Cela permet aux blogueurs de partager leurs connaissances et leurs expériences spécifiques dans leur domaine d'expertise.

En résumé, ce projet vise à offrir une plateforme de blogs complète et interactive, permettant aux utilisateurs d'accéder à une multitude de sujets et de catégories, de partager leurs propres articles et de participer à des discussions enrichissantes au sein de la communauté des blogs.

# Cahier de Charge :

## 1-Contexte générale du projet :

Nous avons l'intention de créer un blog dont les parties backend et frontend seront développées et testées de manière indépendante. Le backend sera conçu sous la forme d'une API JSON. Voici les technologies que nous prévoyons d'utiliser pour ce projet :

Du côté **backend** :

Le Framework Node.js Express pour le serveur web.

* L'ORM Prisma et une base de données MySQL/MariaDB pour la persistance des données.

Du côté **frontend** :

* HTML/CSS/JavaScript.

Vous pouvez utiliser la bibliothèque JavaScript jQuery et le Framework CSS Bootstrap.

## 2- Contraint fonctionnelles :

L'application Blog doit respecter les contraintes fonctionnelles suivantes, en gérant quatre entités :

**1. Utilisateur :**

**3. Catégorie :**

- Nom

**4. Commentaire :**

- Email

- Contenu

- Nom

- Email

- Mot de passe

- Rôle (ADMIN || AUTHOR)

**2. Article :**

- Titre

- Contenu

- Image

- Date de création

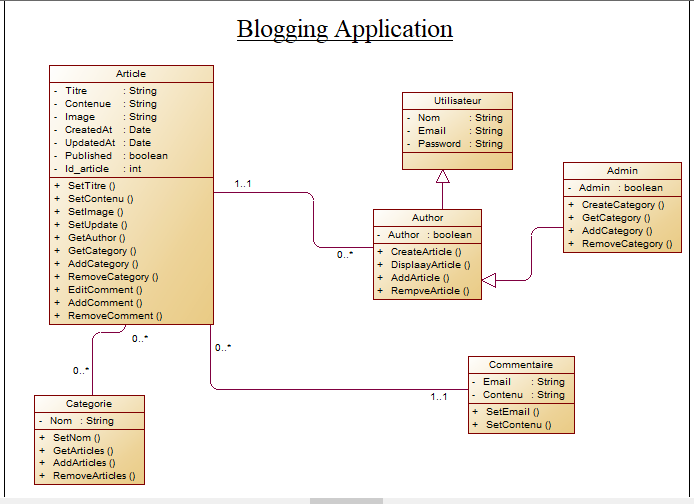
- Date de mise à jour

- Statut de publication sous forme booléenne

## 3-Relations entre les entités :

* Un **article** est associé **à un et un seul** **utilisateur** (Cet utilisateur devrait avoir le role AUTHOR)
* Un **utilisateur** (ayant le rôle AUTHOR) peut écrire **zéro ou plusieurs** **articles**
* Un **article** est associé à **zéro ou plusieurs catégories**
* Une **catégorie** est associée à **zéro ou plusieurs** **articles**
* Un **commentaire** est associé à **exactement un** **article**
* Un **article** est associé à **zéro ou plusieurs** **commentaires**.

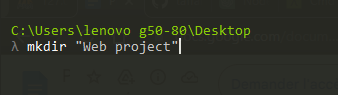
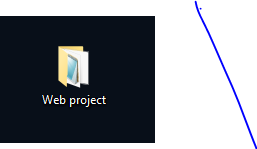
# Diagramme de Classe :



Réalisation du projet :

# Backend :

1. J’ai crée le répertoire de notre projet :



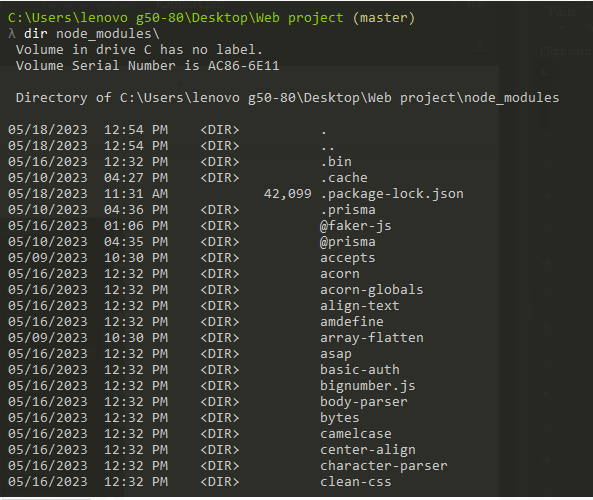
1. J’ai saisi les commandes nécessaires pour commencer :

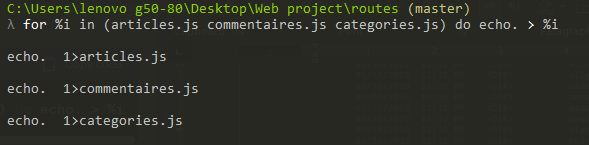




* Pour installer le répertoire node modules :







1. Prisma ORM :



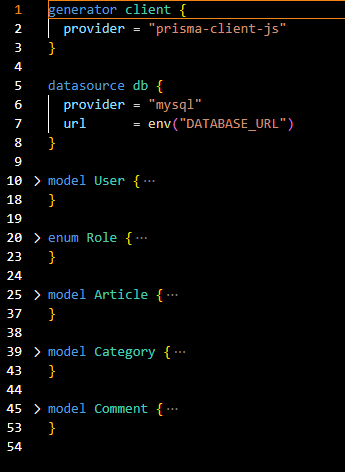




1. .env

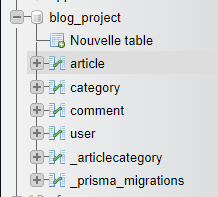


1. Après l’ajout des modèles reflétant les entités et les relations entre elles et la migration :



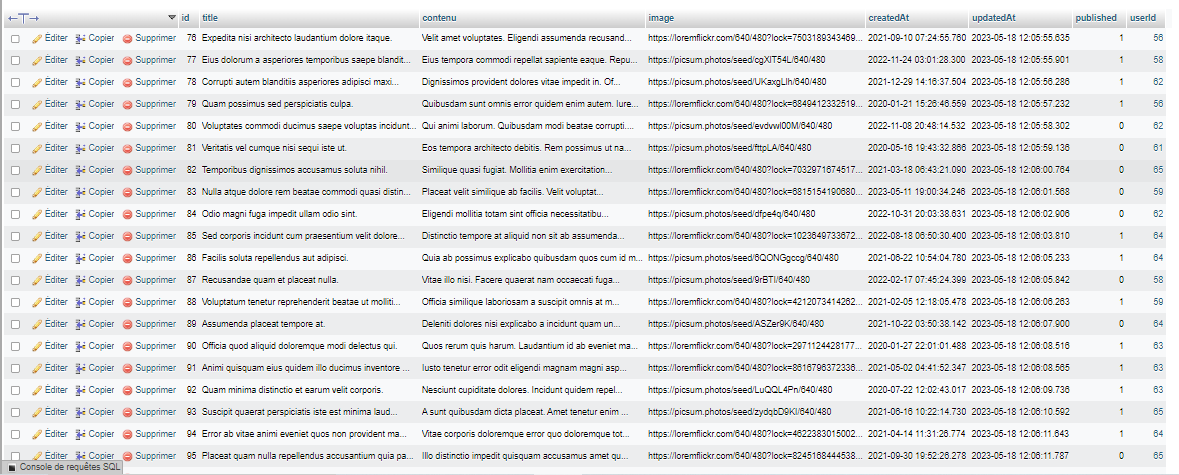
On exécute la commande : **npx prisma migrate dev --name init**

* **Donc notre base de données a été créer :**



1. Dans le fichier ‘’web project\routes\seeds\seed.js’’ on exécute un code utilisant la bibliothèque Faker qui va nous permettre de créer :
   * 10 utilisateurs ayant le rôle “AUTHOR”
   * 1 utilisateur ayant le rôle “ADMIN”
   * 10 catégories
   * 100 articles appartenant à (de 1 à 4 catégories aléatoires) et écrit par l’un des 10 utilisateurs (AUTHOR)
   * Pour chaque article, créer de 0 à 20 commentaires)

Voilà un example de l’execution :



* Le code utilise le framework Express.js pour créer un routeur authRouter qui gère les routes d'authentification.
* La route /login est configurée en tant que route POST pour l'authentification des utilisateurs. Lorsqu'une requête POST est reçue sur cette route, elle est gérée par la fonction authenticateUser.
* La route /logout est également configurée en tant que route POST pour permettre aux utilisateurs de se déconnecter. Lorsqu'une requête POST est reçue sur cette route, elle est gérée par la fonction logoutUser.
* Vous devez implémenter les fonctions authenticateUser et logoutUser dans le contrôleur authenticationController en fonction de votre logique spécifique d'authentification et de déconnexion.

Le router authRouter est exporté pour pouvoir être utilisé dans votre application principale.

En résumé, ce code met en place un router Express pour gérer les routes d'authentification. Il définit des routes pour l'authentification et la déconnexion des utilisateurs, qui sont associées à des fonctions spécifiques dans le contrôleur authenticationController. Cela fournit une infrastructure de base pour implémenter l'authentification et la déconnexion des utilisateurs dans votre application.

# Frontend :