RAPPORT DE PROJET

Architecture des Applications Web

Enseignant :
Karim BENYAHIA

Réalisé par :
Malek SAHLI
Lyna KESSOURI
Nhu HO

Cursus : 1ère année de master Informatique
Année universitaire 2023/2024

Remerciements

Avant de débuter ce rapport, nous tenons à exprimer notre gratitude envers vous, notre professeur d'UE Architecture des Applications Web. Votre soutien tout au long des travaux pratiques et du développement de notre projet a revêtu une importance capital

Table des matières

Première partie : étude de projet

1 - Introduction

Dans le cadre de notre première année de master en informatique à l'Université de Poitiers, nous avons l'opportunité de réaliser une application de l'unité d'enseignement "Architecture des Applications Web".
Ce projet nous offre l'occasion d'acquérir une première expérience dans le développement de l'application web dynamique de côté client et serveur, de mettre en pratique les connaissances et d'explorer les technologies telles que NodeJS, React ainsi que pour la partie base de données c'est MongoDB Compass qu'on a choisie pour notre application.
Nous utilisons également le logiciel Postman pour tester et gérer les API exploitant des fonctionnalités qui incluent la possibilité d'envoyer des requêtes HTTP (GET, POST, PUT, DELETE, ect.)

2 - Structure de projet

2.1. Configuration du projet

2.1.1. Backend avec Node.js et MongoDB

Le projet Node.js est créé en utilisant un gestionnaire de packages npm.
Utilise la dépendance comme Express pour la gestion des routes, Mongoose pour l'interaction avec MongoDB, et d'autres modules.

La gestion des routes :

auth.routes.js :
Les routes commencent par "/api/auth".
Les middlewares de vérification (comme 'checkDuplicateUsernameOrEmail' et 'checkRolesExisted') sont utilisés pour valider les données avant d'appeler les fonctions de contrôleur correspondantes du fichier 'auth.controller'.

post-like.routes.js :
Les routes commencent par "/api/post-like".
Le middleware verifyToken de authJwt est utilisé pour assurer l'authentification avant d'appeler les fonctions de contrôleur du fichier 'post-like.controller'.

post.routes.js :
Les routes commencent par "/api/posts".
Les middlewares de vérification de token (verifyToken) et de rôle administrateur (haveAdminRole) sont utilisés pour sécuriser certaines routes avant d'appeler les fonctions de contrôleur du fichier 'post.controller'.

user.routes.js :
Les routes commencent par "/api/admin/users".
Les middlewares de vérification de token (verifyToken) et de rôle administrateur (isAdmin) sont utilisés pour sécuriser certaines routes avant d'appeler les fonctions de contrôleur du

fichier ‘user.controller’.

Dans ‘server.js’, nous avons créé une instance d’Express avec ‘const app = express()’ et nous avons ensuite passé cet objet ‘app’ à chaque fichier de route spécifié.

La connexion à MongoDB avec Mongoose :

Nous établions la connexion à la base de données MongoDB en utilisant l’URL fournie, avec des options pour le nouveau moteur de surveillance et le nouveau passeur.

```
const url =
    "mongodb+srv://admin:webm1@cluster0.7mbewn7.mongodb.net/quote_app?retryWrites=true&w=majority";
```

“mongodb+srv://admin:webm1@cluster0.7mbewn7.mongodb.net/quote_app?retryWrites=true&w=majority”

2.1.2. Frontend avec React

‘App.js’

La configuration des Routes :
En utilisant le composant ‘Router’ pour envelopper les composants et définir les routes à l’intérieur de ‘Routes’.
Chaque Route spécifie un chemin et le composant React correspondant qui doit être rendu lorsque ce chemin est atteint.
‘PrivateRoute’ semble être un composant personnalisé pour gérer les routes privées.

‘index.js’

Redux Provider et Store:
Nous utilisons le composant Provider de react-redux pour envelopper l’application et fournir le store Redux à tous les composants de l'application.
Nous créons le store Redux avec ‘createStore’ en utilisant les réducteurs (reducers) combinés et le middleware Saga (sagaMiddleware).

Exécution du Middleware Saga :
Nous créons une instance du middleware Saga et l'ajoutons au store en utilisant applyMiddleware.
Nous ensuite exécutons le saga principale (mySaga) avec sagaMiddleware.run.

Rendu de l'Application :
Nous utilisons le Provider pour envelopper le composant App et lui fournir l'accès au store Redux.
ReactDOM.render pour rendre l'application dans la racine de notre document HTML.

Configuration de l’API :
Une constante exportée ‘API_URL’ qui contient l’URL de base de notre API backend.

2.2. Intégration Frontend et Backend

API REST :

- Établissement d'une API RESTful pour permettre la communication entre le frontend et le backend.
- Utilisation des méthodes telles que GET, POST, PUT, DELETE pour les opérations CRUD.

Connexion avec Redux-Saga :

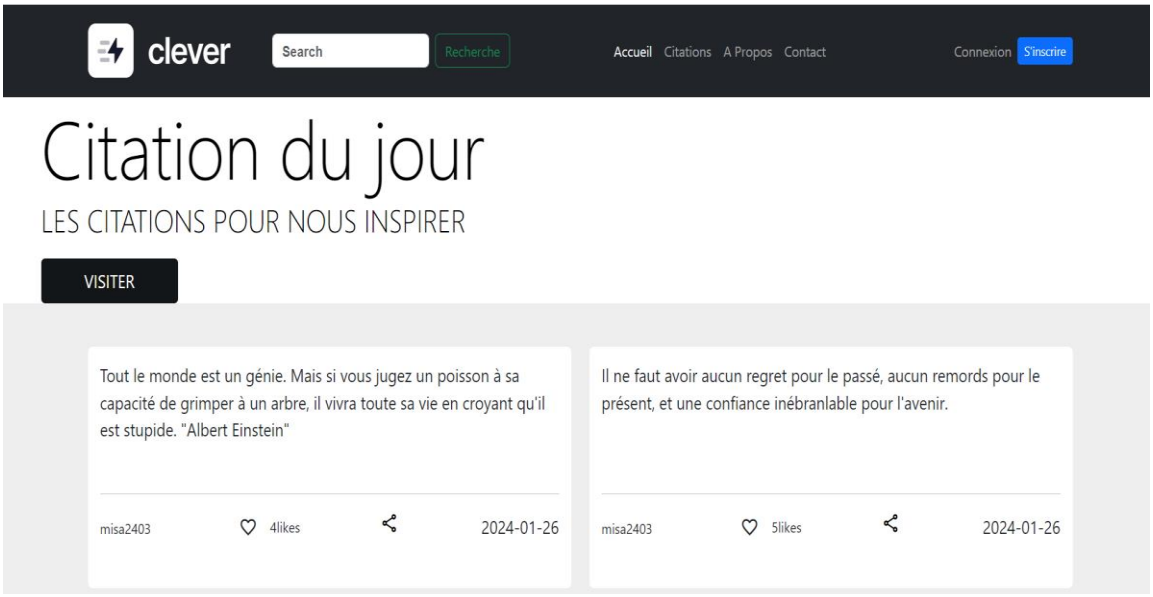
À l'aide de Redux-Saga, on effectue des appels API asynchrones depuis le frontend.
On utilise les sagas pour intercepter les actions Redux spécifiques, ainsi qu'effectuer des tâches asynchrones et dispatch des nouvelles actions pour mettre à jour le store.

Deuxième partie : travail réalisé

1. Site web / (non) connecté

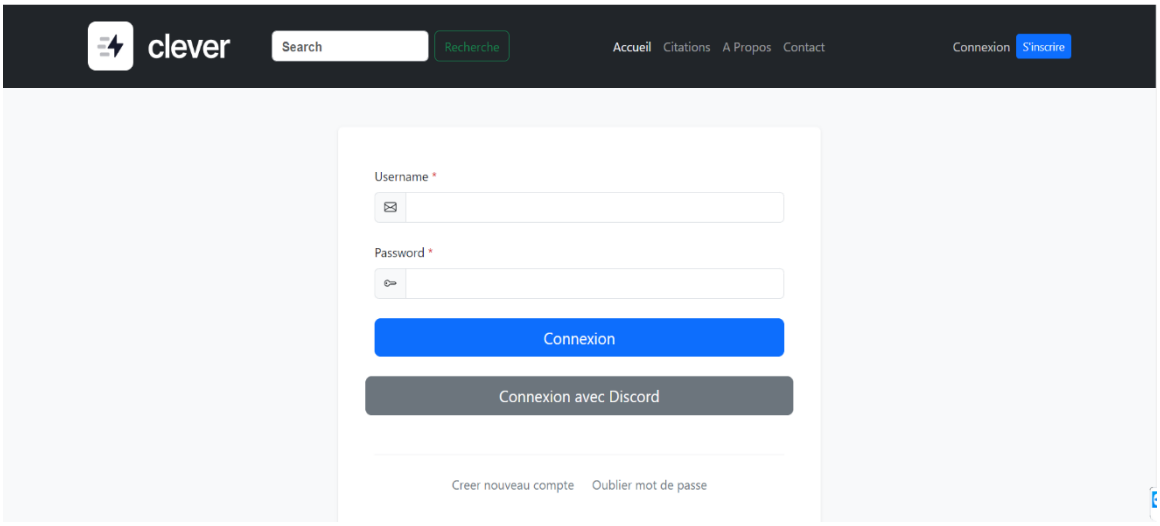
La page d'accueil – menu

Sur la page d'accueil, on dispose d'un menu et d'un ensemble de texte décrivant notre site.
Le menu comporte un lien de connexion (Button Connexion).
Le menu comporte un lien d'inscription (Button S'inscrire).
Le menu comporte un lien vers la liste des citations (Button Citations / Visiter).



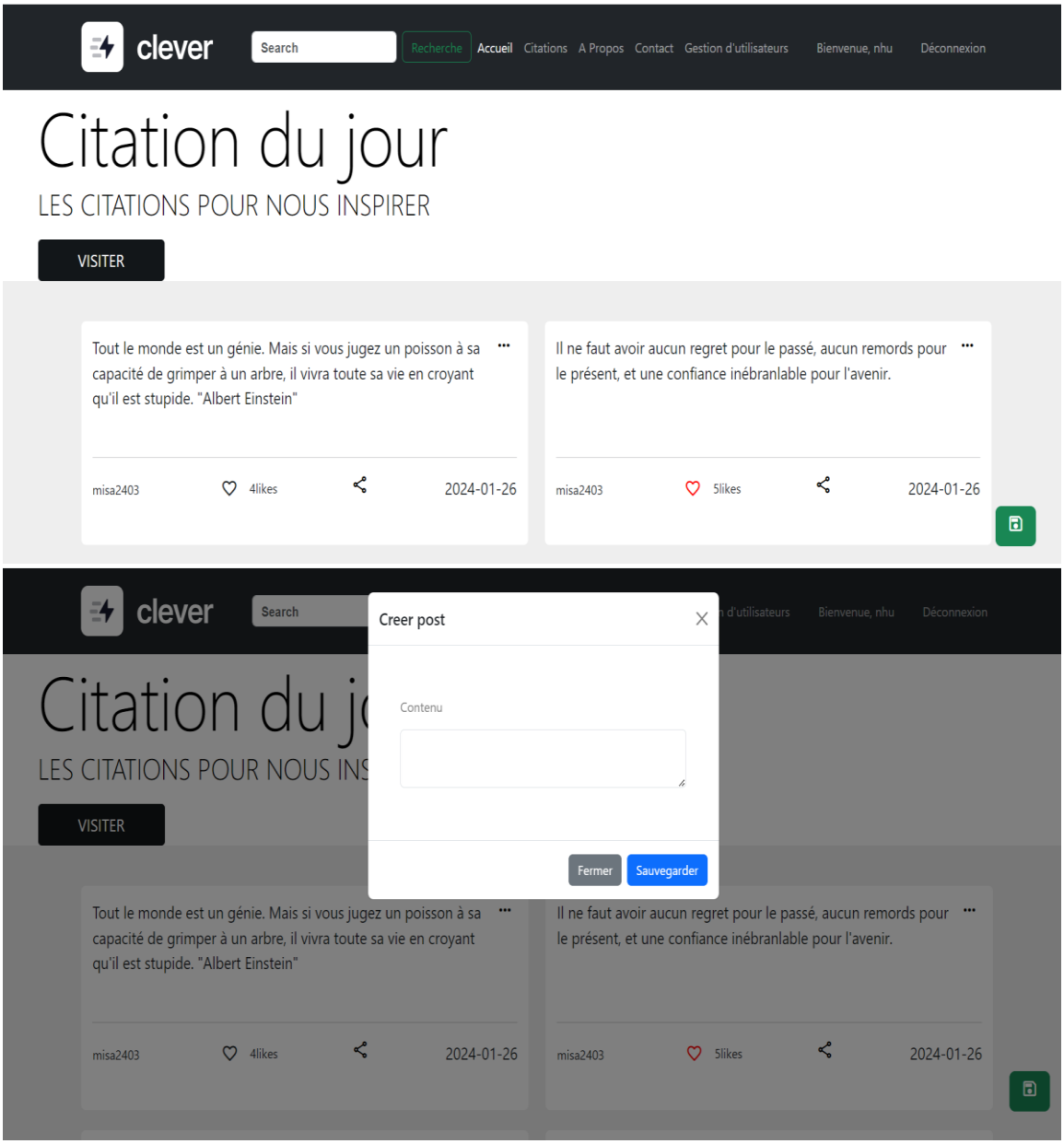
2. Connexion

Nous avons deux modes de connexion, connexion avec le username et le mot de passe ou avec discord. Vous pouvez choisir le serveur dont vous voulez ajouter votre BOT tant que vous êtes administrateur .



3. Site web / connecté

Une fois que nous nous connectons au site, il affiche un bouton vert en bas à droite, qui permet a l'utilisateur connecté d'ajouter une citation. L'utilisateur peut également aimer d'une citation. L'utilisateur peut 'Modifier/Supprimer' sa propre citation en tant qu'utilisateur mais non pas les citations des autres. Le droit de 'Modifier/Supprimer' toutes les citations s'appliquent que pour l'administrateur.



4. BOT discord

Pour les utilisateurs qui se connectent avec Discord, ils peuvent également faire toutes ces actions via le bot Citation's Server.

- *Ajouter une citation :*
  /cquote --content contenue_de_citation
  Par exemple : /cquote --content La famille est le plus important dans ma vie

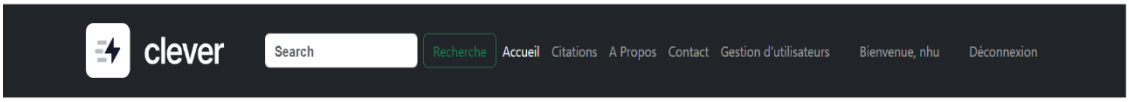
- *Afficher toutes les citations qui ont étaient publiées sur l'application :*
  /lquote

- *Afficher les citations aimées :*
  /likequote

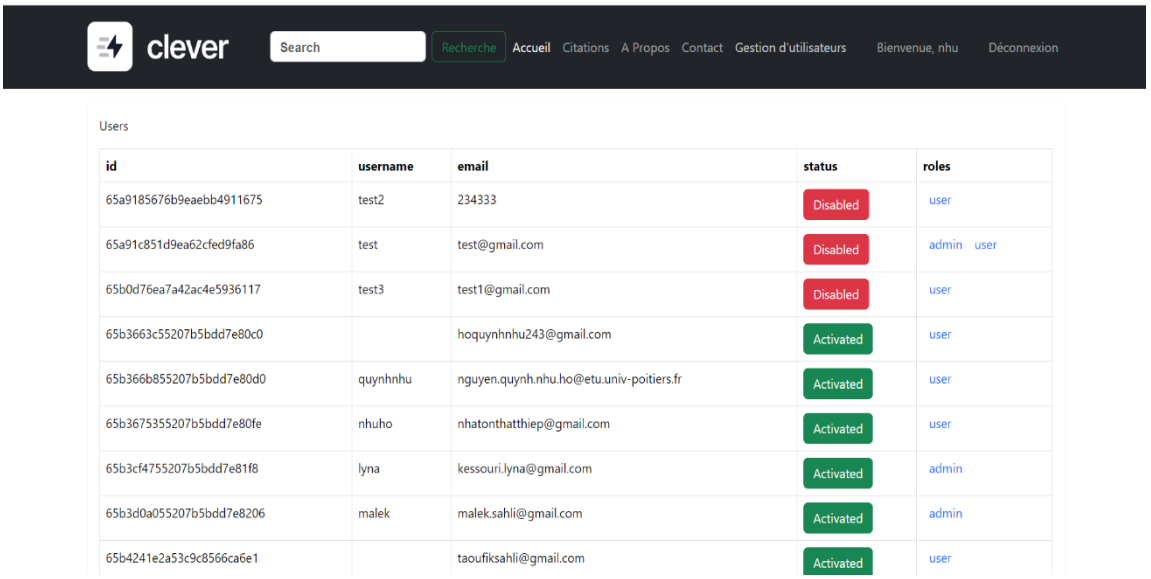
5. Administration

Il y a un rôle administrateur, si un utilisateur est admin, il peut ajouter/ supprimer une citation de n'importe utilisateur.

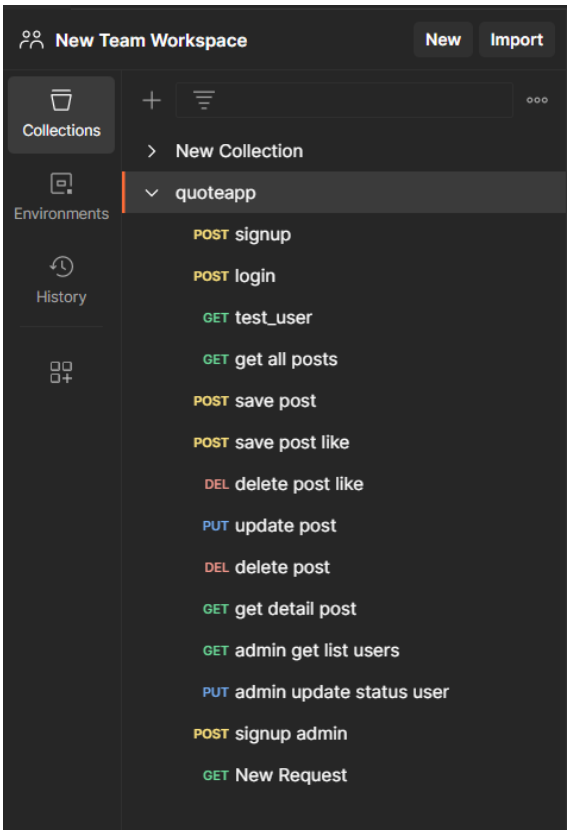
Gérer tous les utilisateurs via Gestion d'utilisateurs. Ou il peut déconnecter un utilisateur.



Ici vous trouvez tous les utilisateurs avec leur id, username, email, statut du compte (activé ou désactivé) et leur rôle.



6. Tester avec Postman



Pour évaluer et tester les fonctionnalités de notre application, nous avons adopté l'utilisation de Postman, un outil dédié à la création et à l'exécution de requêtes HTTP. Postman facilite le processus de test en permettant une interaction simplifiée avec notre API.

Exemple de test n°1 :

Authentification (Login)

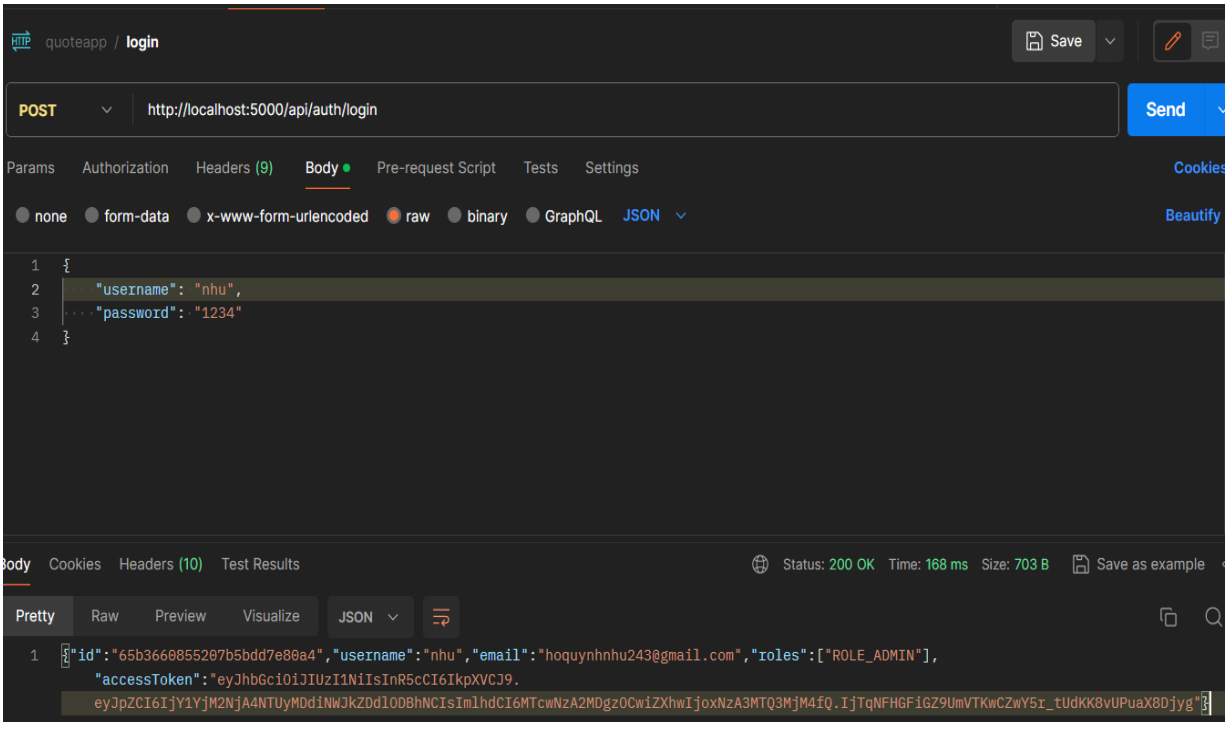
Nous vérifions la réponse pour confirmer le succès de l'authentification et récupérons éventuellement le jeton d'accès pour les requêtes ultérieures.

Cette requête vise à authentifier l'utilisateur avec les informations suivantes :

- Nom d'utilisateur (username): "nhu"
- Mot de passe (password): "1234"

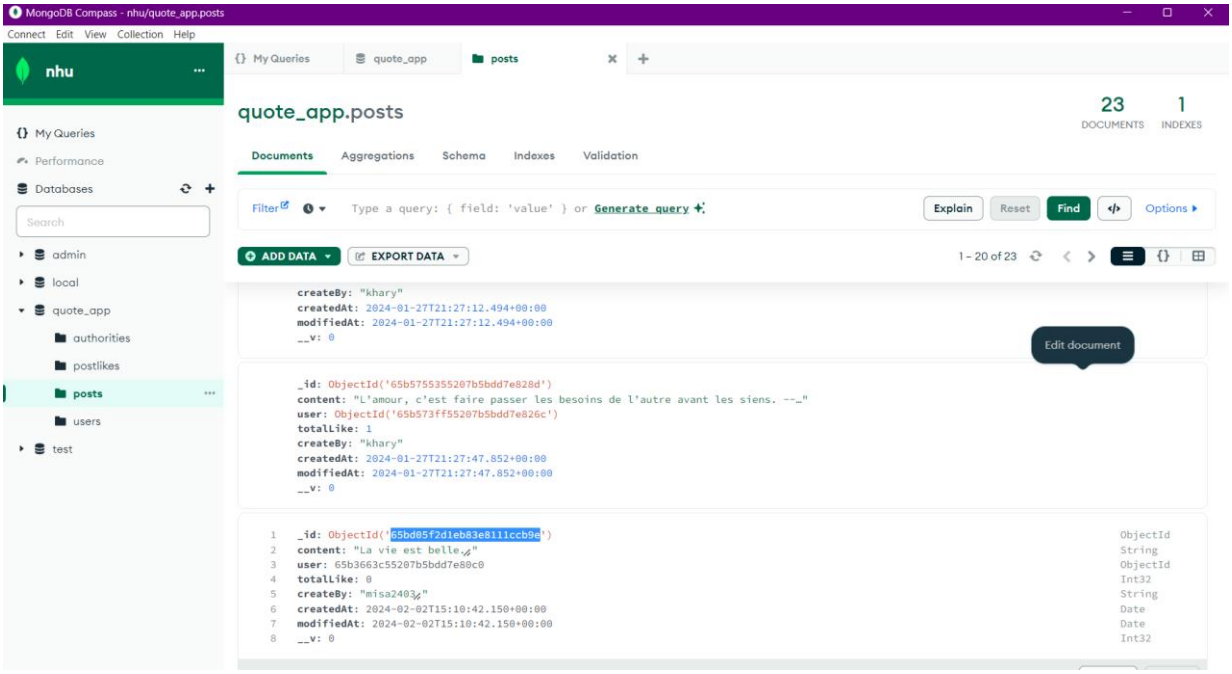
La réponse de cette requête contient les détails de l'utilisateur authentifié, y compris un jeton d'accès (accessToken) qui peut être utilisé pour les futures requêtes nécessitant une authentification.

Note : Vous pouvez vous connecter avec username karim et le mot de passe est 1234 en tant qu' administrateur.

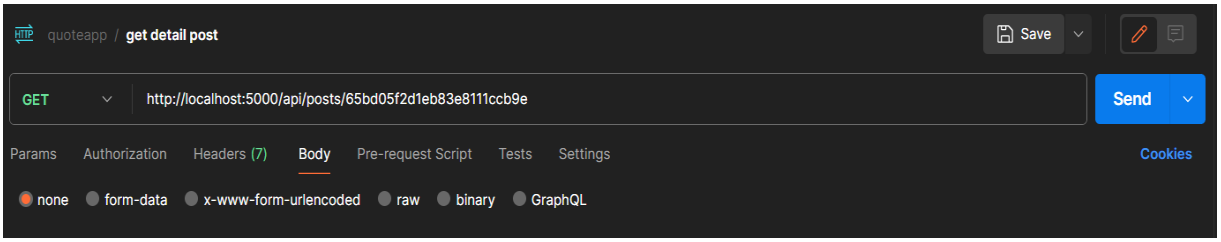


Exemple de test n°2 : Récupération le contenu d'une citation

Dans la base de données de MongoBD, nous souhaitons récupérer le contenu de la citation de l'identifiant 65bd05f2d1eb83e8111ccb9e.



Sur Postman, nous envoyons une requête avec méthode ‘GET’ comme suivant :



Le résultat obtenu de requête est :