

Projet Base de données NoSQL



Présenté par :
Astrid Aurelien NKUMBE ENONGENE
Anne-Josée LOUIS

Sommaire :

Sommaire :	1
1. Introduction	2
Contexte.....	2
Objectifs.....	2
Technologies Utilisées.....	2
2. Architecture Système	3
Détail des Composants.....	3
Interaction entre les Composants.....	4
3. Modèle de Données	4
MongoDB.....	4
Elasticsearch.....	5
Transformation des Données pour l'Indexation.....	5
4. Processus d'Indexation	6
Préparation des Données.....	6
Indexation dans Elasticsearch.....	6
Indexation Batch et Unitaire.....	6
Automatisation de l'Indexation.....	7
Gestion des Erreurs.....	7
5. Conclusion	7
6. Annexes	8

1. Introduction

Contexte

Dans un monde de plus en plus connecté, la capacité de mettre en relation des individus ou des services en fonction de leur position géographique devient cruciale. Que ce soit pour des besoins sociaux, professionnels, ou de services de proximité, l'accès à des informations précises et en temps réel sur la localisation des individus ou des ressources offre des avantages significatifs. Notre projet s'inscrit dans cette dynamique, en proposant un outil de mise en relation par géolocalisation. Cet outil permettra aux utilisateurs de s'enregistrer, de fournir des informations personnelles ainsi que leurs coordonnées géographiques, et de rechercher des personnes disponibles dans un rayon défini.

Objectifs

Le projet vise à développer une application qui répond aux besoins suivants :

- **Enregistrement des Utilisateurs** : Permettre aux individus de s'enregistrer en fournissant des informations personnelles et leurs coordonnées de géolocalisation. L'enregistrement peut se faire soit individuellement, soit par lot à travers l'importation de fichiers JSON.
- **Indexation des Données** : Les données fournies par les utilisateurs seront indexées à l'aide d'un moteur d'indexation. Cette étape est cruciale pour optimiser les performances de recherche au sein de l'application.
- **Recherche géolocalisée** : Offrir une fonction de recherche permettant de trouver des utilisateurs en fonction de leur position géographique et dans un rayon spécifié par le demandeur.
- **Affichage des Résultats** : Les résultats de chaque recherche devront présenter les informations complètes des personnes trouvées, permettant ainsi des mises en relation efficaces et pertinentes dans un monde connecté, géolocalisation cruciale pour besoins sociaux/professionnels..

Technologies Utilisées

Pour répondre aux exigences fonctionnelles et techniques de ce projet, nous avons sélectionné un ensemble de technologies éprouvées et performantes :

- **Langage de Développement** : Python, choisi pour sa flexibilité, sa facilité d'utilisation et son vaste écosystème de bibliothèques, notamment pour le traitement des données et l'intégration avec les systèmes de bases de données et d'indexation.
- **Moteur d'Indexation** : Elasticsearch, retenu pour sa capacité à gérer de grandes quantités de données géolocalisées et à effectuer des recherches rapides et précises basées sur la géolocalisation.
- **SGBD** : MongoDB, une base de données NoSQL orientée documents, sélectionnée pour sa flexibilité dans le stockage et la gestion des données structurées de manière hiérarchique, ce qui est particulièrement adapté aux

données personnelles et géolocalisées des utilisateurs.

- **Système d'Exploitation** : Linux, choisi pour sa stabilité, sa sécurité et sa performance, ainsi que pour sa large adoption dans les environnements de serveurs et de développement.

Le projet s'articule autour de ces technologies, chacune jouant un rôle clé dans la réalisation d'un système robuste, performant et capable de répondre aux besoins spécifiques de mise en relation géolocalisée.

2. Architecture Système

L'architecture est structurée de manière à faciliter les opérations d'enregistrement des utilisateurs, de stockage et d'indexation des données, ainsi que la recherche et l'affichage des informations. Les utilisateurs interagissent avec l'application principale via une interface utilisateur ou une API, l'application communique avec MongoDB pour le stockage des données et avec Elasticsearch pour l'indexation et la recherche géographique.

Voici un aperçu détaillé de chaque composant et de leur interaction.

Détail des Composants

- Application Principale (Python)
 - Fonctionnalités : Gère les interactions utilisateurs, l'enregistrement des données, les opérations de transformation et d'indexation, et les requêtes de recherche.
 - Intégration avec MongoDB : Enregistre et récupère les informations des utilisateurs, en utilisant le modèle de données conçu pour optimiser le stockage des informations complexes.
 - Intégration avec Elasticsearch : Transfère les données nécessaires depuis MongoDB vers Elasticsearch, applique les transformations requises pour l'indexation, et gère les requêtes de recherche géographique.
 - Interface Utilisateur/API : Offre aux utilisateurs un moyen d'interagir avec le système, que ce soit via une interface graphique ou une API pour l'enregistrement et la recherche.
- Base de Données (MongoDB)
 - Stockage des Données : Contient les informations détaillées des utilisateurs, y compris les données personnelles et géolocalisées, structurées selon le modèle de données spécifié.
 - Avantages : La flexibilité du modèle de données document et la capacité à gérer de grandes quantités de données avec des performances élevées.
- Moteur d'Indexation (Elasticsearch)
 - Indexation des Données : Indexe les informations des utilisateurs nécessaires pour la recherche géographique, en utilisant le mapping spécifié pour optimiser les requêtes.

- Recherche Géographique : Permet des recherches rapides et précises basées sur la localisation géographique, en utilisant des critères tels que la distance par rapport à un point donné.
- Système d'Exploitation (Linux)
 - Environnement d'Exécution : Fournit une plateforme stable et sécurisée pour l'exécution de l'application, de MongoDB et d'Elasticsearch, favorisant les performances et la fiabilité.

Interaction entre les Composants

Enregistrement des Utilisateurs : Les utilisateurs fournissent leurs informations via l'interface de l'application, qui les stocke ensuite dans MongoDB.

Transformation et Indexation : L'application extrait les données nécessaires de MongoDB, les transforme selon le modèle requis par Elasticsearch, et les indexe pour optimiser la recherche géographique.

Recherche : Lorsqu'une requête de recherche est soumise via l'interface utilisateur, l'application interroge Elasticsearch pour trouver les utilisateurs correspondants selon les critères géographiques. Les résultats sont ensuite récupérés de MongoDB pour afficher les informations complètes.

Affichage des Résultats : Les informations détaillées des utilisateurs trouvés sont présentées à l'utilisateur ayant initié la recherche.

Cette architecture offre une séparation claire des préoccupations entre le stockage des données, leur indexation pour la recherche géographique, et l'interaction utilisateur, assurant ainsi une gestion efficace des ressources et une expérience utilisateur optimale.

3. Modèle de Données

La conception du modèle de données est un élément fondamental de notre application de recherche géographique. Il détermine comment les informations des utilisateurs sont stockées, organisées et interconnectées. Pour ce projet, nous utilisons MongoDB comme base de données et Elasticsearch pour l'indexation des données géolocalisées. Voici le détail du modèle de données pour ces deux systèmes :

MongoDB

MongoDB, une base de données orientée documents, offre une grande flexibilité dans la modélisation des données. Les documents JSON (JavaScript Object Notation) permettent une structure hiérarchique, facilitant ainsi le stockage de données complexes comme celles des utilisateurs de notre application.

- Structure du Document Utilisateur :
 - `id` (long) : Identifiant unique de l'utilisateur.
 - `firstname` (text) : Prénom de l'utilisateur.

- `lastName` (text) : Nom de famille de l'utilisateur.
- `email` (text) : Adresse email de l'utilisateur.
- `birthDate` (date) : Date de naissance de l'utilisateur.
- `login` (objet) : Informations de connexion de l'utilisateur, comprenant un identifiant unique (`uuid`), un nom d'utilisateur (`username`), un mot de passe (`password`) et des informations de sécurité (`md5`, `sha1`, `enregistrement`).
- `address` (objet) : Adresse de l'utilisateur, incluant la rue (`street`), le numéro d'appartement (`suite`), la ville (`city`), le code postal (`zipcode`) et les coordonnées géographiques (`geo`) avec latitude (`lat`) et longitude (`lng`).
- `phone` (text) : Numéro de téléphone de l'utilisateur.
- `website` (text) : Site web de l'utilisateur.
- `company` (objet) : Informations sur l'entreprise de l'utilisateur, incluant le nom (`name`), le slogan (`catchPhrase`) et le secteur d'activité (`bs`).

Ce modèle de document est conçu pour stocker de manière exhaustive les informations des utilisateurs, permettant ainsi une gestion complète et une recherche efficace au sein de l'application.

Cette étape de transformation garantit que les données sont optimisées pour la recherche géographique, tout en conservant toutes les informations nécessaires pour une mise en relation efficace entre les utilisateurs.

Elasticsearch

Elasticsearch utilise des structures de données appelées "indices" pour stocker les données de manière optimisée pour la recherche. Le mapping de notre indice `users` définit comment les documents doivent être stockés et indexés.

- Mapping de l'Indice `users` :
 - `id` (long) : Identifiant unique de l'utilisateur, correspondant à l'`id` dans MongoDB.
 - `name` (text) : Nom complet de l'utilisateur, combinant `firstname` et `lastName` de MongoDB.
 - `username` (text) : Nom d'utilisateur, extrait de l'objet `login` de MongoDB.
 - `birthDate` (date) : Date de naissance de l'utilisateur.
 - `address` (text) : Adresse complète de l'utilisateur, formatée à partir des données de l'objet `address` de MongoDB.
 - `geo_point_2d` (geo_point) : Coordonnées géographiques de l'utilisateur, stockées sous forme d'objet `geo_point` pour permettre des recherches géographiques efficaces dans Elasticsearch.

Le type `geo_point` est particulièrement important car il permet d'effectuer des recherches basées sur la localisation géographique, telles que trouver des utilisateurs dans un rayon spécifique autour d'un point donné.

Transformation des Données pour l'Indexation

La transformation des données de MongoDB vers Elasticsearch est cruciale pour assurer que les données soient correctement formatées et indexées pour la recherche. Ce processus implique :

- La combinaison des champs `firstname` et `lastname` de MongoDB pour créer le champ `name` dans Elasticsearch.
- La conversion de l'adresse de l'utilisateur en MongoDB en un format texte unique pour le champ `address` dans Elasticsearch.
- L'extraction et le formatage des coordonnées géographiques de MongoDB pour les adapter au type `geo_point` dans Elasticsearch.

4. Processus d'Indexation

L'indexation des données dans notre système de recherche géographique est cruciale pour la performance et l'efficacité des recherches. Ce processus implique la transformation des données stockées dans MongoDB pour qu'elles soient optimisées pour la recherche dans Elasticsearch. Voici les étapes détaillées de ce processus d'indexation :

Préparation des Données

Avant l'indexation, les données doivent être préparées et transformées pour correspondre au mapping défini dans Elasticsearch. Cette préparation inclut :

- Extraction des Données de MongoDB : Sélection des documents utilisateurs à partir de MongoDB qui nécessitent une indexation ou une réindexation.
- Transformation des Données : Adaptation des données au format attendu par Elasticsearch. Cela inclut la combinaison des champs `firstname` et `lastname` en un seul champ `name`, la mise en forme de l'adresse et la conversion des coordonnées géographiques en un type `geo_point`.

Indexation dans Elasticsearch

Une fois les données préparées, elles sont indexées dans Elasticsearch selon le processus suivant :

- Création/Mise à Jour de l'Index : Les documents transformés sont envoyés à Elasticsearch. Si un document avec l'ID spécifié existe déjà, il est mis à jour avec les nouvelles données; sinon, un nouveau document est créé.
- Utilisation du Type `geo_point` : Pour les champs de coordonnées géographiques, le type `geo_point` est utilisé, permettant à Elasticsearch d'optimiser les recherches basées sur la localisation.

Indexation Batch et Unitaire

Le système prend en charge deux modes d'indexation :

- Indexation Batch : Permet l'indexation de plusieurs documents en une seule opération. Typiquement utilisé lors de l'initialisation du système ou pour l'ajout en masse de nouveaux utilisateurs à partir de fichiers CSV ou JSON.
- Indexation Unitaire : Permet l'indexation d'un seul document à la fois, utilisé pour l'enregistrement ou la mise à jour des informations d'un utilisateur via l'interface utilisateur ou l'API.

Automatisation de l'Indexation

Pour maintenir l'index à jour avec les dernières données disponibles dans MongoDB, plusieurs stratégies peuvent être mises en œuvre :

- Indexation sur Modification : Chaque fois qu'un document utilisateur est ajouté ou modifié dans MongoDB, une opération d'indexation correspondante est automatiquement déclenchée pour mettre à jour l'index Elasticsearch.
- Indexation Périodique : Un processus planifié réindexe périodiquement les données dans Elasticsearch pour s'assurer que toutes les modifications dans MongoDB sont reflétées dans l'index.

Gestion des Erreurs

Durant l'indexation, le système doit être capable de gérer les erreurs potentielles, telles que les échecs de connexion à Elasticsearch ou les erreurs de format de données. Une stratégie robuste de gestion des erreurs inclut :

- Réessayer en Cas d'Échec : En cas d'échec temporaire, le système peut réessayer l'opération d'indexation un nombre défini de fois.
- Logging : Tous les échecs d'indexation sont enregistrés, permettant une analyse ultérieure pour résoudre les problèmes.

5. Conclusion

Le succès de ce projet repose sur une compréhension approfondie des besoins utilisateurs, qui a guidé la conception et le développement de chaque composant du système. En mettant l'accent sur une expérience utilisateur intuitive et des résultats de recherche précis et pertinents, nous avons créé un outil qui non seulement répond aux attentes des utilisateurs mais les dépasse car la structure du système, pensée pour être à la fois robuste et flexible, garantit non seulement la performance et la fiabilité de l'application mais offre également une plateforme évolutif pour intégrer des fonctionnalités supplémentaires à l'avenir.

Ce projet démontre l'impact positif de l'ingénierie logicielle et des systèmes d'information modernes dans la résolution de problématiques complexes et la

création de valeur pour les utilisateurs. Il ouvre également la voie à de nouvelles explorations et innovations dans le domaine de la géolocalisation et de la mise en relation, offrant un vaste champ de possibilités pour les développements futurs. Notre engagement continu dans la recherche, le développement et l'amélioration de nos solutions est essentiel pour rester à l'avant-garde de la technologie et répondre aux défis émergents dans ce domaine dynamique.

6. Annexes

- Code source : Extraits significatifs du code développé.
- Captures d'écran : Illustrations de l'interface utilisateur et des exemples de résultats de recherche.

https://github.com/nkaurelien-ionis-stm/projet_mongo_db

The screenshot shows the GitHub interface for the repository 'projet_mongo_db' by user 'nkaurelien-ionis-stm'. The repository is public and has 1 branch and 0 tags. The main branch is selected. The repository contains 4 commits and has 0 stars and 0 forks. The file list shows a directory structure with folders like .idea, node_modules, resources, src, and webapp, and files like .env.exemple, .gitignore, README.md, package.json, requirements.txt, and yarn.lock. The README file is selected and its content is displayed at the bottom. The right sidebar shows the repository's metadata, including the README, Activity, Custom properties, 0 stars, 1 watching, 0 forks, and a link to report the repository. The Releases section shows no releases published, and the Packages section shows no packages published. The Languages section shows Python at 100.0%.

File/Folder	Commit Message	Time Ago
.idea	Ajoute de la recherche	last week
node_modules	WIP	3 hours ago
resources	WIP	3 hours ago
src	WIP	3 hours ago
webapp	Ajoute de la recherche	last week
.env.exemple	WIP	3 hours ago
.gitignore	Ajoute de la recherche	last week
README.md	WIP	5 hours ago
package.json	WIP	3 hours ago
requirements.txt	WIP	5 hours ago
yarn.lock	Ajoute de la recherche	last week

Recherche Géographique dans Elasticsearch

Latitude

24,89

-

+

Longitude

21,90

-

+

Rayon (km)

10000

-

+

Rechercher

Resulat de la recherche:

Nombre d'utilisateurs trouvées sur 10000km: 5 / 5

	Nom	Username	Adresss	Latitude	Longitude
0	Leanne Graham	Bret	Kulas Light, 92998-3874 Gwenborough	-37.3159	81.1496
1	Ervin Howell	Antonette	Victor Plains, 90566-7771 Wisokyburgh	-43.9509	-34.4618
2	Chelsey Dietrich	Kamren	Skiles Walks, 33263 Roscoeview	-31.8129	62.5342
3	Kurtis Weissnat	Elwyn.Skiles	Rex Trail, 58804-1099 Howemouth	24.8918	21.8984
4	Clementina DuBuque	Moriah.Stanton	Kattie Turnpike, 31428-2261 Lebsackbury	-38.2386	57.2232



Tous les utilisateurs

10 / 10

Nom	Username	Adresss	Latitude
Leanne Graham	Bret	Kulas Light, 92998-3874 Gwe...	-37.3159
Ervin Howell	Antonette	Victor Plains, 90566-7771 Wi...	-43.9509
Clementine Bauch	Samantha	Douglas Extension, 59590-41...	-68.6102
Patricia Lebsack	Karianne	Hoeger Mall, 53919-4257 So...	29.4572
Chelsey Dietrich	Kamren	Skiles Walks, 33263 Roscoeview...	-31.8129
Mrs. Dennis Schulist	Leopoldo_Corkery	Norberto Crossing, 23505-13...	-71.4197
Kurtis Weissnat	Elwyn.Skiles	Rex Trail, 58804-1099 Howe...	24.8918
Nicholas Runolfsdottir V	Maxime_Nienow	Ellsworth Summit, 45169 Ali...	-14.3990
Glenna Reichert	Delphine	Dayna Park, 76495-3109 Bart...	24.6463
Clementina DuBuque	Moriah.Stanton	Kattie Turnpike, 31428-2261 ...	-38.2386

```
[
  {
    "_index": "users",
    "_id": "CYgxm44BYzzxx51UPpQ9",
    "_score": 1.0,
    "_source": {
      "id": 1,
      "name": "John Doe",
      "username": "johndoe",
      "birthDate": "1973-01-22",
      "address": "123 Main Street, 12345-6789 Anytown\nApt. 4",
      "geo_point_2d": {
        "lat": "42.1234",
        "lon": "-71.2345"
      }
    }
  },
  {
    "_index": "users",
    "_id": "Cogxm44BYzzxx51UPpQ9",
    "_score": 1.0,
    "_source": {
      "id": 2,
      "name": "Jane Smith",
      "username": "janesmith",
      "birthDate": "1983-02-22",
      "address": "456 Oak Street, 12345-6789 Anytown\nSuite 200",
      "geo_point_2d": {
        "lat": "42.3456",
        "lon": "-71.6789"
      }
    }
  }
],
```

```

NosqlMongoDb > resources > data > {} users.json > ...
You, 1 hour ago | 1 author (You)
You, 5 hours ago • WIP
1  [
2    {
3      "id": 1,
4      "firstname": "John",
5      "lastname": "Doe",
6      "email": "johndoe@example.com",
7      "birthDate": "1973-01-22",
8      "login": {
9        "uuid": "1a0eed01-9430-4d68-901f-c0d4c1c3bf22",
10       "username": "johndoe",
11       "password": "jsonplaceholder.org",
12       "md5": "c1328472c5794a25723600f71c1b4586",
13       "sha1": "35544a31cc19bd6520af116554873167117f4d94",
14       "registered": "2023-01-10T10:03:20.022Z"
15     },
16     "address": {
17       "street": "123 Main Street",
18       "suite": "Apt. 4",
19       "city": "Anytown",
20       "zipcode": "12345-6789",
21       "geo": {
22         "lat": "42.1234",
23         "lng": "-71.2345"
24       }
25     },
26     "phone": "(555) 555-1234",
27     "website": "www.johndoe.com",
28     "company": {
29       "name": "ABC Company",
30       "catchPhrase": "Innovative solutions for all your needs",
31       "bs": "Marketing"
32     }
33   },

```

```

{
  "name": "NosqlMongoDb",
  "version": "1.0.0",
  "main": "index.js",
  "repository": "https://github.com/nkaurelien-ionis-stm/projet_mongo_db.git",
  "author": "Aurelien Nkumbe <nkaurelien@gmail.com>",
  "license": "MIT",
  > Debug
  "scripts": {
    "deploy": "npm run importdb; npm run createsearchindex",
    "start": "streamlit run src/streamlitappuser.py",
    "start2": "streamlit run src/streamlitappsncf.py",
    "importdb": "python src/utils/import_mongodb.py",
    "createsearchindex": "python src/utils/create_search_index.py"
  }
}

```

```
NosqlMongoDb > $ .env.example
You, 3 hours ago | 1 author (You)
1 ES_SERVER_NAME=3.72.35.55
2 ES_SERVER_PORT=9200
3
4 MONGO_DB_DATABASE=projet_mongo_db
5 MONGO_DB_URL = "mongodb+srv://${MONGO_DB_USER}:${MONGO_DB_PASSWORD}@cluster0.xifg
6
```

```
NosqlMongoDb > src > streamlitappuser.py > ...
1 import streamlit as st
2 import pandas as pd
3 from st_aggrid import AgGrid
4 from utils.es import Search
5
6 es = Search()
7
8 es.index_name = "users"
9
10 columns = ["Nom", "Username", "Adresss", "Latitude", "Longitude"]
11
12 start_lat=24.8918
13 start_lng=21.8984
14
15 @st.cache_data
16 def get_data() -> pd.DataFrame:
17     (users, total) = es.get_all_data()
18     print(users)
19     data = []
20
21     df = pd.DataFrame(data, columns=columns)
22
23     if users:
24         for hit in users:
25             source = hit["_source"]
26             data.append([source["name"], source["username"], source["address"], source["geo_point_2d"]])
27             source["geo_point_2d"] = source["geo_point_2d"]
28         if data:
29             df = pd.DataFrame(data, columns=columns)
30
31     return (df, total)
32
33 You, 3 hours ago * WIP
34
35 # Titre de l'application
36 st.title('Recherche Géographique')
37
38 # Formulaire de recherche géographique
39 with st.form(key='search_geo'):
40     lat = st.number_input('Latitude', value=start_lat) # Valeur par défaut pour Kurtis Elwyn
41     lon = st.number_input('Longitude', value=start_lng) # Valeur par défaut pour Kurtis Elwyn
42     rayon = st.number_input('Rayon (km)', value=10000)
43     submit_button = st.form_submit_button(label='Rechercher')
```

```

NosqlMongoDb > resources > mappings > {} users.json > ...
You, 3 hours ago | 1 author (You)
1  {}
2  "mappings": {
3      "properties": {
4          "id": {"type": "long"},
5          "name": {"type": "text"},
6          "username": {"type": "text"},
7          "email": {"type": "text"},
8          "address": {"type": "text"},
9          "geo_point_2d": {"type": "geo_point"}
10     }
11 }
12 {}
You, 5 hours ago • WIP

```

```

NosqlMongoDb > src > utils > mapping.py > ...
2
3 def transform_data(document, index_name):
4
5     transformed_document = document # This is just a placeholder line
6
7     if index_name == 'gares' :
8         transformed_document = {
9             "id": document["code_uic"],
10            "libelle": document["libelle"],
11            "commune": document["commune"],
12            "departemen": document["departemen"],
13            "geo_point_2d": {"lat": document["geo_point_2d"]["lat"], "lon": document["geo_point_2d"]["lon"]}
14        }
15
16     elif index_name == 'users' :
17         transformed_document = {
18             "id": document["id"],
19             "name": "{} {}".format(document["firstname"], document["lastname"]),
20             "username": document["login"]["username"],
21             "birthDate": document["birthDate"],
22             "address": "{} {} {}".format(document["address"]["street"], document["address"]["zipcode"], doc
23             "geo_point_2d": {"lat": document["address"]["geo"]["lat"], "lon": document["address"]["geo"]["lng"]}
24         }
25
26     return transformed_document
27

```

```

NosqlMongoDb > src > utils > import_mongodb.py > ...
 2  from pymongo.mongo_client import MongoClient
 3  from pymongo.server_api import ServerApi
 4  import os
 5  import json
 6
 7  from dotenv import load_dotenv
 8
 9  load_dotenv()
10
11  db_name=os.environ["MONGO_DB_DATABASE"]
12  mongo_db_server_url = os.environ["MONGO_DB_URL"]
13
14  # Create a new client and connect to the server
15  client = MongoClient(mongo_db_server_url, server_api=ServerApi('1'))
16  db = client[db_name]
17
18  #----- MAIN -----
19
20
21  directory = 'resources/data' # Update this path to the directory containing your
22
23  # List all JSON files in the directory
24  files = [f for f in os.listdir(directory) if f.endswith('.json')]
25
26  # Import each JSON file into MongoDB
27  for file in files:
28      # Extracting the filename without extension to use as collection name
29      collection_name = os.path.splitext(file)[0]
30
31      # Access the collection
32      collection = db[collection_name]
33
34      # Drop the collection if it exists
35      collection.drop()
36
37      # Open and load the JSON file
38      with open(os.path.join(directory, file), 'r') as json_file:
39          data = json.load(json_file)
40
41      # Assuming each file contains an array of documents
42      if isinstance(data, list):
43          collection.insert_many(data)
44      else: # If the file contains a single document
45          collection.insert_one(data)
46

```

```

NosqlMongoDb > src > utils > create_search_index.py > ...
27     You, 3 hours ago • WIP
28     # Import each JSON file into MongoDB
29     for data_file in data_files:
30         # Extracting the filename without extension to use as collection name
31         index_name = os.path.splitext(data_file)[0]
32
33         # Check if the index already exists
34         if es.indices.exists(index=index_name):
35             # Delete the index if it exists
36             es.indices.delete(index=index_name)
37
38         mapping_file_path = os.path.join(mapping_directory, index_name + '.json')
39
40         if os.path.exists(mapping_file_path):
41             with open(mapping_file_path, 'r') as mapping_file:
42                 mapping = json.load(mapping_file)
43                 # Create the index with the mapping
44                 # es.indices.create(index=index_name, body=mapping, ignore=400) # ignore
45                 es.indices.create(index=index_name, body=mapping)
46
47         else:
48             print(f"No mapping file found for index {index_name}. Creating index without
49                 # Create the index without specific mapping
50                 es.indices.create(index=index_name)
51
52
53         # Load and transform data, then insert into Elasticsearch
54         with open(os.path.join(data_directory, data_file), 'r') as f:
55             data = json.load(f)
56
57             # Assume data is a list of documents. If not, adjust accordingly.
58             transformed_data = [transform_data(doc, index_name) for doc in data]
59
60             # Use the bulk API to insert transformed documents
61             actions = [
62                 {
63                     "_index": index_name,
64                     "_source": doc,
65                 }
66                 for doc in transformed_data
67             ]
68
69             helpers.bulk(es, actions)
70
71     print("Data transformation and insertion complete.")

```


NosqlMongoDb > ⓘ README.md > # INSTALL

```
1
2  # INSTALL      You, 6 days ago • first commit
3  Install python virtual env
4
5  ```console
6  pip install --upgrade pip
7  sudo apt install python3-virtualenv
8  pip install --upgrade virtualenv
9  ```
10
11  # Create virtual Env
12  Install python lib
13
14  ```shell
15  virtualenv -p python3 venv
16  chmod +x venv/bin/activate
17  ```
18  Install python lib
19  ```shell
20  pip install -r requirements.txt
21  ```
22
23  # RUN
24  Activate python venv
25  ```shell
26  source ./venv/bin/activate
27  ```
28
29  ### Import dataset to mongoDb
30  ```shell
31  python src/utils/import_mongodb.py
32  ```
33
34  ### Import dataset to Elasticseacrh
35  ```shell
36  python src/utils/create_search_index.py
37  ```
38
39  # Run app
40  ```shell
41  yarn start
42  # or
43  streamlit run src/streamlitappuser.py
44  ```
45
```