Exercice DevOps

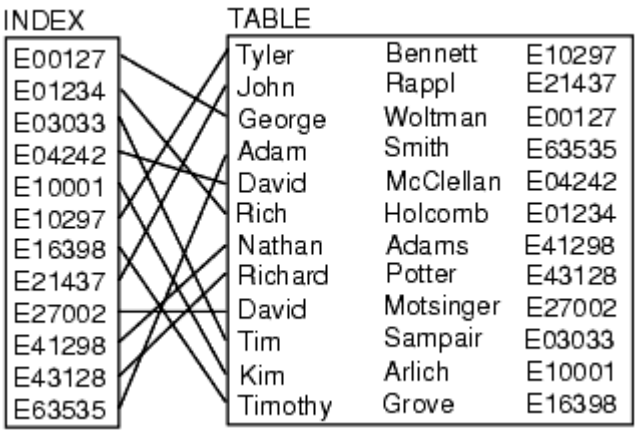
Mettre en place Docker et Docker Compose sur son ordinateur.

Qu'est ce que ElasticSearch ?

ElasticSearch est une base de données qui fournit un moyen flexible et puissant pour rechercher du texte grâce à des indexs inversés

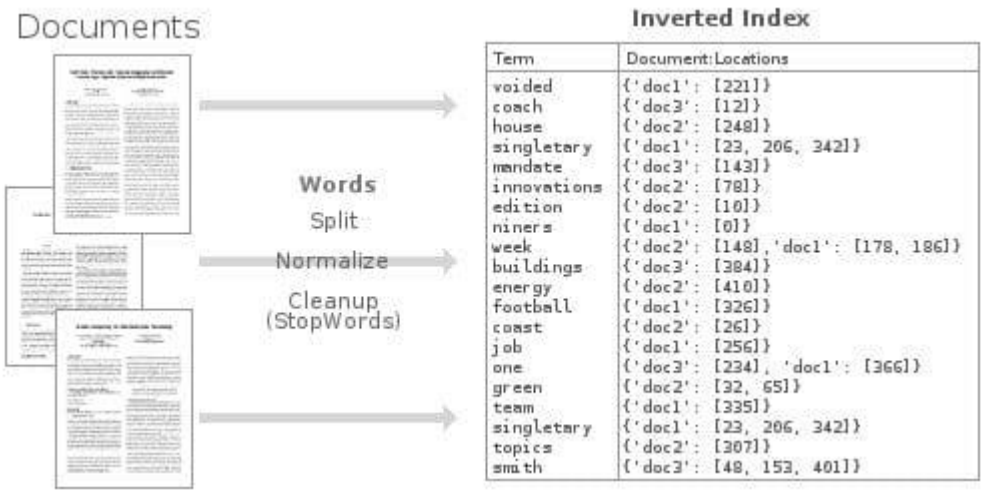
Un index est une structure de données (façon d'entreposer des données) qui permet de retrouver des informations très rapidement.

Une base de données SQL standard va, généralement, relier la ligne du tableau à l'index structuré en arbre B.



Si nous connaissons l'auteur ou le nom du livre, une base de données traditionnelle les retrouvera rapidement.

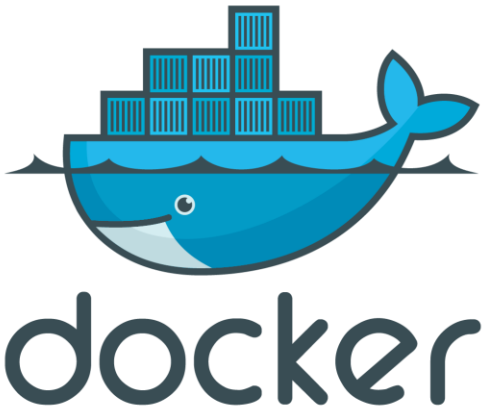
Un index inversé fonctionne différement. Les données sont découpées et chaque index est relié à l'ensemble des documents où l'information a été trouvée.



Si nous voulons trouver l'ensemble des documents contenant le mot football, il le fera rapidement.

2 - MISE EN PLACE DU PROJET

2.0 - Docker



2.1 - Installer Docker et Docker-Compose

Installer Docker - https://docs.docker.com/engine/installation/
Installer Docker Compose - https://docs.docker.com/compose/install/

2.2 – Répertoire de configuration du projet

Fabriquer un répertoire nommé `guttenberg_search` pour le projet.
Ce projet aura 2 sous-répertoires

- `/public` – Contiendra les répertoires pour l'application front en Vue.js
- `/server` – Contiendra le serveur en node.js

2.3 – Ajouter la configuration de Docker-Compose

Nous allons mettre en place le fichier `docker-compose.yml` pour chaque application

1. `gs-api` – Pour le conteneur Note.js, partie back de notre application
2. `gs-frontend` – Pour Nginx qui fournira les fichiers fronts (Vue.js)
3. `gs-search` – Pour Elasticsearch qui contiendra les données

```
version: '3'

services:
  api: # Node.js App
    container_name: gs-api
    build: .
    ports:
      - "3000:3000" # Port de l'API exposé
      - "9229:9229" # Port pour le débogage de Node (à désactiver en production)
    environment: # variables d'environnement
     - NODE_ENV=local
     - ES_HOST=elasticsearch
     - PORT=3000
    volumes: # repertoire des livres
      - ./books:/usr/src/app/books

  frontend: # Serveur Nginx pour l'app. front
    container_name: gs-frontend
    image: nginx
    volumes: # Répertoire accessible « répertoire public »
      - ./public:/usr/share/nginx/html
    ports:
      - "8080:80" # liaison port 8080 à 80

  elasticsearch: # Instance Elasticsearch
    container_name: gs-search
    image: docker.elastic.co/elasticsearch/elasticsearch:6.1.1
    volumes: # données d'ES dans un volume séparé "esdata"
      - esdata:/usr/share/elasticsearch/data
    environment:
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
      - discovery.type=single-node
    ports: # Expose ports Elasticsearch
```

```
      - "9300:9300"
      - "9200:9200"

volumes: # Définit le volume séparé des données ES
  esdata:
```

Ce fichier définit notre stack applicatif, il n'y a pas besoin d'installer ElasticSearch, Node ou Nginx sur notre système. Chaque conteneur redirige les ports du système hôte (`localhost`)

2.4 - Le Dockerfile

Nous allons utiliser une image déjà faite pour la partie Nginx/ElasticSearch mais, pour l'application back, Node.js, nous devons la mettre en place

Mettre en place un `Dockerfile` dans le répertoire où se trouvera l'application back en Node.

```
# Node v8.9.0 LTS
FROM node:carbon

# Configuration du répertoire de travail
WORKDIR /usr/src/app

# Copie « package.json » et « package-lock.json »
COPY package*.json ./

# Installe les dépendances de l'application node (npm est équivalent à apt/yum pour js)
RUN npm install

# copie le code source
COPY . .

# démarre l'application
CMD [ "npm", "start" ]
```

Cette configuration docker utilise l'image officielle de Node.js mais place le code source de notre application et ses dépendances dans le conteneur
Nous ajoutons un fichier `.dockerignore` pour éviter de copier des fichiers inutiles dans le docker

```
node_modules/
npm-debug.log
books/
public/
```

Remarquez que nous ne copions pas le répertoire `node_modules dans le conteneur` , car nous allons utiliser `npm install` qui fera l'installation des dépendances dans le conteneur

2.5 – Ajout des fichiers de base

Nous allons mettre en place les différents fichiers pour tester l'application.

Ajouter dans le repertoire, le fichier html : `public/index.html`

```
<html><body>Ceci est le conteneur de la partie front</body></html>
```

Ensuite, ajouter le fichier Node ; `server/app.js`

```
const Koa = require('koa')
const app = new Koa()

app.use(async (ctx, next) => {
  ctx.body = 'Bienvenue sur le conteneur partie back'
})

const port = process.env.PORT || 3000

app.listen(port, err => {
  if (err) console.error(err)
  console.log(`App Listening on Port ${port}`)
})
```

Ensuite, ajouter le fichier à notre application `package.json` : Il permet d'indiquer les dépendances du projet JavaScript/Node

```
  "name": "guttenberg-search",
  "version": "0.0.1",
  "description": "Code source pour l'application ElasticSearch",
  "scripts": {
    "start": "node --inspect=0.0.0.0:9229 server/app.js"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/triestpa/guttenberg-search.git"
```

```
  },
  "author": "patrick.triest@gmail.com (merci à lui)",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/triestpa/guttenberg-search/issues"
  },
  "homepage": "https://github.com/triestpa/guttenberg-search#readme",
  "dependencies": {
    "elasticsearch": "13.3.1",
    "joi": "13.0.1",
    "koa": "2.4.1",
    "koa-joi-validate": "0.5.1",
    "koa-router": "7.2.1"
  }
}
```

2.6 – Teston

Tout est en place, dans le repétoire du projet, lancer `docker-compose build` ; il va construire notre application Node

```
patrick@localmac ~/D/p/es-tut-test> docker-compose build
elasticsearch uses an image, skipping
Building api
Step 1/6 : FROM node:carbon
 ---> 2eeae8debf3d
Step 2/6 : WORKDIR /usr/src/app
 ---> Using cache
 ---> c8ce3a9e45b8
Step 3/6 : COPY package*.json ./
 ---> Using cache
 ---> 64b97113b889
Step 4/6 : RUN npm install
 ---> Using cache
 ---> e0499e99cd21
Step 5/6 : COPY . .
 ---> Using cache
 ---> c7268febfb81
Step 6/6 : CMD [ "npm", "start" ]
 ---> Using cache
 ---> f10df49e0233
Successfully built f10df49e0233
Successfully tagged estuttest_api:latest
frontend uses an image, skipping
```

Puis `docker-compose up` qui va lancer le stack applicatif

```
patrick@localmac ~/D/p/es-tut-test> docker-compose up
Starting gs-frontend ...
Starting gs-search ...
Starting gs-search ... done
Attaching to gs-frontend, gs-api, gs-search
gs-api           |
gs-api           | > guttenberg-search@0.0.1 start /usr/src/app
gs-api           | > node --inspect=0.0.0.0:9229 server/app.js
gs-api           |
gs-api           | Debugger listening on ws://0.0.0.0:9229/a7f90c71-a883-4460-8e82-519e0c206c
gs-api           | For help see https://nodejs.org/en/docs/inspector
gs-api           | App Listening on Port 3000
gs-search        | [2018-01-24T01:55:13,368][INFO ][o.e.n.Node               ] [] initializir
gs-search        | [2018-01-24T01:55:13,471][INFO ][o.e.e.NodeEnvironment    ] [qRLuv_V] usir
gs-search        | [2018-01-24T01:55:13,471][INFO ][o.e.e.NodeEnvironment    ] [qRLuv_V] heap
gs-search        | [2018-01-24T01:55:13,473][INFO ][o.e.n.Node               ] node name [qRL
gs-search        | [2018-01-24T01:55:13,473][INFO ][o.e.n.Node               ] version[6.1.1]
1/25.151-b12]
```

Cela peut prendre du temps car il y a un certain nombre de téléchargement.

Dans le navigateur, taper et tester : `localhost:8080` , il devrait afficher la page «Ceci est le conteneur de la partie front »

Et en allant sur `localhost:3000` , on devrait avoir : «`Bienvenue sur le conteneur partie back`»

Et sur `localhost:9200` on devrait voir ElasticSearch tourner et afficher quelque chose ressemblant à :

```
{
  "name" : "SLTcfpI",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "iId8e0ZeS_mgh9ALlWQ7-w",
  "version" : {
    "number" : "6.1.1",
    "build_hash" : "bd92e7f",
    "build_date" : "2017-12-17T20:23:25.338Z",
    "build_snapshot" : false,
    "lucene_version" : "7.1.0",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

Si toutes les URLS fonctionnent, parfait !

Sinon, vérifier étape par étape depuis le début

3 - CONNECTER A ELASTICSEARCH

Nous devons maintenant connecter node à ES

3.0 – Ajouter le module de connexion ES

Ajouter un nouveau fichier `server/connection.js`.

```
const elasticsearch = require('elasticsearch')

// Variable de connexion pour ce projet
const index = 'library'
const type = 'novel'
const port = 9200
const host = process.env.ES_HOST || 'localhost'
const client = new elasticsearch.Client({ host: { host, port } })

/** Test l'état de la connexion */
async function checkConnection () {
  let isConnected = false
  while (!isConnected) {
    console.log('Connexion à ES')
    try {
      const health = await client.cluster.health({})
      console.log(health)
      isConnected = true
    } catch (err) {
      console.log('Connexion ratée, nouvelle tentative...', err)
    }
  }
}

checkConnection()
```

Reconstuire le conteneur Node avec les changements avec la commande : `docker-compose build`. puis `docker-compose up -d` avec l'application tournant en daemon

Quand l'application démarre, lancer en ligne de commande dans le conteneur `docker exec gs-api "node" "server/connection.js"`

Nous devrions avoir :

```
{ cluster_name: 'docker-cluster',
  status: 'yellow',
  timed_out: false,
  number_of_nodes: 1,
  number_of_data_nodes: 1,
  active_primary_shards: 1,
  active_shards: 1,
  relocating_shards: 0,
  initializing_shards: 0,
  unassigned_shards: 1,
  delayed_unassigned_shards: 0,
  number_of_pending_tasks: 0,
  number_of_in_flight_fetch: 0,
  task_max_waiting_in_queue_millis: 0,
  active_shards_percent_as_number: 50 }
```

Nous pouvons donc désactiver la fonction checkConnection() que nous avons créé dans le précédent fichier

3.1 – Ajouter une fonction pour réinitialiser l'index

Dans `server/connection.js` ajouter après la fonction `checkConnection`, ce qui permet de réinitialiser facilement les indexs

```
/** Vide les indexs, recréé et map */
async function resetIndex () {
  if (await client.indices.exists({ index })) {
    await client.indices.delete({ index })
  }

  await client.indices.create({ index })
  await putBookMapping()
}
```

3.2 – Ajout du schema/mapping des livres

Nous allons ajouter le mapping vers les données.
Ajouter à la fin de `server/connection.js`.

```
/** Ajout du mapping de données ES */
async function putBookMapping () {
  const schema = {
    title: { type: 'keyword' },
    author: { type: 'keyword' },
    location: { type: 'integer' },
    text: { type: 'text' }
  }

  return client.indices.putMapping({ index, type, body: { properties: schema } })
}
```

Nous avons défini le mapping pour les index de livres `book`. Un `index` Elasticsearch est l'équivalent d'une table ou d'une collection MongoDB.

Ajouter un mapping, nous permet de spécifier les champs pour les documents stockés. ES est « schema-less », c'est à dire qu'il n'est pas nécessaire, techniquement, d'ajouter un mappage. Cependant, pour des questions de performances, l'ajout d'un mappage peut avoir un impact énorme sur le comportement et la vitesse de recherche.

Par exemple, nous affectons le type `keyword` aux champs "title" et "author", le type `text` au champ "text" et le type `integer` au champ « location ». Cela entraînera pour le moteur de recherche un traitement différent de ces champs. Lors d'une recherche, le moteur recherchera les correspondances potentielles dans le champ `text`, tandis que les champs `keyword` seront comparés en fonction de leur contenu intégral.

Exportez les propriétés et les fonctions exposées en bas du fichier, afin qu'elles puissent être accessibles par d'autres modules de notre application.

```
module.exports = {
  client, index, type, checkConnection, resetIndex
}
```

4 – CHARGER LES DONNÉES

Nous utiliserons les données provenant du Projets Gutenberg – une initiative en ligne dédiée à la fourniture gratuite de copies numériques de livres du domaine public.

4.1 - Télécharger les fichiers du livre

J'ai compressé 100 livres dans un fichier que vous pouvez télécharger ici : https://cdn.patricktriest.com/data/books.zip

Dézippé le fichier dans le dossier de votre application : `books/`

Si vous le souhaitez, vous pouvez utiliser les lignes de commande en utilisant **wget** et **unar**.

```
wget https://cdn.patricktriest.com/data/books.zip
unar books.zip
```

4.2 – Structure d'un livre

Si vous ouvrez l'un des fichiers, vous pouvez constater que ce fichier comporte ces éléments :

```
Author: Joseph Conrad

Release Date: February 1995 [EBook #219]
Last Updated: September 7, 2016

Language: English

Character set encoding: UTF-8
```

Puis :

`*** START OF THIS PROJECT GUTENBERG EBOOK HEART OF DARKNESS ***`

Et :

`*** END OF THIS PROJECT GUTENBERG EBOOK HEART OF DARKNESS ***`, suivi d'une version beaucoup plus détaillée de la licence du livre.

Dans les étapes suivantes, nous allons analyser les métadonnées du livre à partir de cet en-tête et extraire le contenu du livre entre les repères de position `*** START OF` et `***END OF`.

4.3 – Lire le dossier `books/`

Nous allons maintenant lire le contenu du livre pour en extraire ces données et les ajouter à ES. Tout ça de manière programmatique bien sûr. Nous allons définir un nouveau fichier Javascript `server/load_data.js` pour effectuer ces opérations.

Tout d'abord nous allons lister le contenu du dossier `books/`. Ajouter le contenu suivant à `server/load_data.js`.

```
const fs = require('fs')
const path = require('path')
const esConnection = require('./connection')

/** Clear ES index, parse and index all files from the books directory */
async function readAndInsertBooks () {
  try {
    // Clear previous ES index
    await esConnection.resetIndex()

    // Read books directory
    let files = fs.readdirSync('./books').filter(file => file.slice(-4) === '.txt')
    console.log(`Found ${files.length} Files`)

    // Read each book file, and index each paragraph in elasticsearch
    for (let file of files) {
      console.log(`Reading File - ${file}`)
      const filePath = path.join('./books', file)
      const { title, author, paragraphs } = parseBookFile(filePath)
      await insertBookData(title, author, paragraphs)
    }
  } catch (err) {
    console.error(err)
  }
}

readAndInsertBooks()
```

Après avoir collé le contenu du code ci-dessus, vous pouvez reconstruire votre image docker et relancer vos conteneurs avec les commandes vues plus haut :

`docker-compose build` et ensuite `docker-compose up -d.`

Cependant, il existe un raccourci qui vous permettra de gagner du temps :

`docker-compose up -d --build`

Cette commande va reconstruire votre image et votre conteneur avant de les relancer en mode deamon.

```
patrick@localmac ~/D/p/es-tut-test> docker-compose up -d --build
Building api
Step 1/6 : FROM node:carbon
 ---> 2eeae8debf3d
Step 2/6 : WORKDIR /usr/src/app
 ---> Using cache
 ---> c8ce3a9e45b8
Step 3/6 : COPY package*.json ./
 ---> Using cache
 ---> 64b97113b889
Step 4/6 : RUN npm install
 ---> Using cache
 ---> e0499e99cd21
Step 5/6 : COPY . .
 ---> 48bfe0d4fbcf
Step 6/6 : CMD [ "npm", "start" ]
 ---> Running in d84aef5010e6
Removing intermediate container d84aef5010e6
 ---> ee44d76f99e9
Successfully built ee44d76f99e9
Successfully tagged estuttest_api:latest
gs-frontend is up-to-date
gs-search is up-to-date
Recreating gs-api ... done
```

Executé la commande : `docker exec gs-api "node" "server/load_data.js"` afin de lancer le script `load_data.js` du conteneur. Vous devriez voir en sortie le statut Elasticsearch, suivie de `Found 100 Books`.

Ensuite une erreur apparaîtra car nous appelons une fonction (`parseBookFile`) qui n'a pas encore été définie.

```
patrick@localmac ~/D/p/es-tut-test> docker exec gs-api "node" "server/load_data.js"
Found 100 Files
Reading File - 10.txt
ReferenceError: parseBookFile is not defined
    at readAndInsertBooks (/usr/src/app/server/load_data.js:18:24)
    at <anonymous>
    at process._tickCallback (internal/process/next_tick.js:188:7)
patrick@localmac ~/D/p/es-tut-test>
```

4.4 – Lire le contenu des fichiers

Nous allons définir une nouvelle fonction dans le fichier `server/load_data.js`. Copié et collé le code suivant :

```
/** Read an individual book text file, and extract the title, author, and paragraphs */
function parseBookFile (filePath) {
  // Read text file
```

```
  const book = fs.readFileSync(filePath, 'utf8')

  // Find book title and author
  const title = book.match(/^Title:\s(.+)$/m)[1]
  const authorMatch = book.match(/^Author:\s(.+)$/m)
  const author = (!authorMatch || authorMatch[1].trim() === '') ? 'Unknown Author' : authorMatch[1]

  console.log(`Reading Book - ${title} By ${author}`)

  // Find Guttenberg metadata header and footer
  const startOfBookMatch = book.match(/^\*{3}\s*START OF (THIS|THE) PROJECT GUTENBERG EBOOK.+\*{3}$/m)
  const startOfBookIndex = startOfBookMatch.index + startOfBookMatch[0].length
  const endOfBookIndex = book.match(/^\*{3}\s*END OF (THIS|THE) PROJECT GUTENBERG EBOOK.+\*{3}$/m).index

  // Clean book text and split into array of paragraphs
  const paragraphs = book
    .slice(startOfBookIndex, endOfBookIndex) // Remove Guttenberg header and footer
    .split(/\n\s+\n/g) // Split each paragraph into it's own array entry
    .map(line => line.replace(/\r\n/g, ' ').trim()) // Remove paragraph line breaks and whitespace
    .map(line => line.replace(/_/g, '')) // Guttenberg uses "_" to signify italics. We'll remove it, since it makes the raw text look messy.
    .filter((line) => (line && line !== '')) // Remove empty lines

  console.log(`Parsed ${paragraphs.length} Paragraphs\n`)
  return { title, author, paragraphs }
}
```

Cette fonction effectue un traitement important :

1. Elle récupère le contenu du fichier
2. Elle utilise les expressions régulières pour analyser le titre et l'auteur du livre (plus d'info sur les expressions régulières).
3. Elle identifie le début et la fin du livre en faisant correspondre l'en-tête `*** START OF THIS PROJECT GUTENBERG EBOOK HEART OF DARKNESS ***` et le pied de page `*** END OF THIS PROJECT GUTENBERG EBOOK HEART OF DARKNESS ***` grâce à une expression régulière.
4. Elle extrait le contenu du livre.
5. Elle divise chaque paragraphe dans son propre tableau.
6. Elle nettoie le texte et supprime les lignes vides.

Comme valeur de retour, nous allons créer un objet contenant le titre du livre, son auteur et un tableau contenant les paragraphes du livre.

Executé les commandes :

```
docker-compose up -d --build
```

et

```
docker exec gs-api "node" "server/load_data.js"
```

Vous devriez voir la même sortie qu'auparavant mais avec trois lignes supplémentaires à la fin.

```
patrick@localmac ~/D/p/es-tut-test> docker exec gs-api "node" "server/load_data.js"
Found 100 Files
Reading File - 0-11-0.txt
Reading Book - Alice's Adventures in Wonderland By Lewis Carroll
Parsed 820 Paragraphs

ReferenceError: insertBookData is not defined
    at readAndInsertBooks (/usr/src/app/server/load_data.js:20:7)
    at <anonymous>
    at process._tickCallback (internal/process/next_tick.js:188:7)
patrick@localmac ~/D/p/es-tut-test>
```

C'est parfait ! Nous avons réussi à analyser le titre, l'auteur et le contenu du livre. Vous devriez voir encore une erreur, rassurez-vous, c'est normal nous n'avons pas encore terminé.

4.5 – Indexer les donnée dans ES

Pour finir, il nous reste plus qu'à indexer nos données dans Elasticsearch.

Ajouter une nouvelle fonction dans votre fichier `load_data.js`, on l'appellera `insertBookData()` :

```
/** Bulk index the book data in Elasticsearch */
async function insertBookData (title, author, paragraphs) {
  let bulkOps = [] // Array to store bulk operations

  // Add an index operation for each section in the book
  for (let i = 0; i < paragraphs.length; i++) {
    // Describe action
    bulkOps.push({ index: { _index: esConnection.index, _type: esConnection.type } })

    // Add document
```

```
    bulkOps.push({
      author,
      title,
      location: i,
      text: paragraphs[i]
    })

    if (i > 0 && i % 500 === 0) { // Do bulk insert in 500 paragraph batches
      await esConnection.client.bulk({ body: bulkOps })
      bulkOps = []
      console.log(`Indexed Paragraphs ${i - 499} - ${i}`)
    }
  }

  // Insert remainder of bulk ops array
  await esConnection.client.bulk({ body: bulkOps })
  console.log(`Indexed Paragraphs ${paragraphs.length - (bulkOps.length / 2)} -
${paragraphs.length}\n\n\n`)
}
```

Cette fonction vous permettra d'indexer chaque paragraphe du livre avec les métadonnées correspondantes.

Nous insérons les paragraphes en utilisant une opération de bloc pour des questions de performances car cette méthode est beaucoup plus rapide que l'indexation individuelle.

Il ne vous reste plus qu'a recontruire votre conteneur docker et à le relancer :

```
docker-compose up -d --build
```

et tester ce que l'on vient de faire, pour cela exécuté la commande suivante :

```
docker exec gs-api "node" "server/load_data.js"
```

Vous devriez maintenant voir une sortie complète de 100 livres analysés et insérés dans Elasticsearch. Cela pourrait prendre une minute ou deux.

```
Reading File - 28054-0.txt
Reading Book - The Brothers Karamazov By Fyodor Dostoyevsky
Parsed 5961 Paragraphs

Indexed Paragraphs 1 - 500
Indexed Paragraphs 501 - 1000
Indexed Paragraphs 1001 - 1500
Indexed Paragraphs 1501 - 2000
Indexed Paragraphs 2001 - 2500
Indexed Paragraphs 2501 - 3000
Indexed Paragraphs 3001 - 3500
Indexed Paragraphs 3501 - 4000
Indexed Paragraphs 4001 - 4500
Indexed Paragraphs 4501 - 5000
Indexed Paragraphs 5001 - 5500
Indexed Paragraphs 5501 - 5961



Reading File - 2814-0.txt
Reading Book - Dubliners By James Joyce
Parsed 1706 Paragraphs

Indexed Paragraphs 1 - 500
Indexed Paragraphs 501 - 1000
Indexed Paragraphs 1001 - 1500
Indexed Paragraphs 1501 - 1706



Reading File - 2852-0.txt
Reading Book - The Hound of the Baskervilles By A. Conan Doyle
Parsed 1484 Paragraphs

Indexed Paragraphs 1 - 500
Indexed Paragraphs 501 - 1000
Indexed Paragraphs 1001 - 1484



Reading File - 3207.txt
Reading Book - Leviathan By Thomas Hobbes
Parsed 2223 Paragraphs

Indexed Paragraphs 1 - 500
Indexed Paragraphs 501 - 1000
Indexed Paragraphs 1001 - 1500
Indexed Paragraphs 1501 - 2000
Indexed Paragraphs 2001 - 2223
```

5 - RECHERCHE

Maintenant qu'Elasticsearch contient une centaine de livres, essayons quelques requêtes de recherche.

5.0 – Requête HTTP

Commençons par tester l'api d'ES grâce à votre navigateur favoris. Taper l'URL suivante :

```
http://localhost:9200/library/_search?q=text:Java&pretty
```

Ici, nous effectuons une recherche en texte intégral simple pour trouver le mot "Java" dans notre bibliothèque de livres.

Vous devriez voir une réponse JSON semblable à la suivante.


```
{
  "took" : 11,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 13,
    "max_score" : 14.259304,
    "hits" : [
      {
        "_index" : "library",
        "_type" : "novel",
        "_id" : "p_GwFWEBaZvLlaAUdQgV",
        "_score" : 14.259304,
        "_source" : {
          "author" : "Charles Darwin",
          "title" : "On the Origin of Species",
          "location" : 1080,
          "text" : "Java, plants of, 375."
        }
      },
      {
        "_index" : "library",
        "_type" : "novel",
        "_id" : "wfKwFWEBaZvLlaAUkjfk",
        "_score" : 10.186235,
```

```
        "_source" : {
          "author" : "Edgar Allan Poe",
          "title" : "The Works of Edgar Allan Poe",
          "location" : 827,
          "text" : "After many years spent in foreign travel, I sailed in the year 18--
, from the port of Batavia, in the rich and populous island of Java, on a voyage to the
Archipelago of the Sunda islands. I went as passenger--having no other inducement than a
kind of nervous restlessness which haunted me as a fiend."
        }
      },
      ...
    ]
  }
}
```

Un peu de sécurité :

L'API d'ES est utiles en développement pour vérifier que nos données ont été correctement indexées. Cependant, il ne faut **JAMAIS** exposer cette API publiquement. En effet celle-ci est composée de fonctionnalités administratives (telles que l'ajout et la suppression directs de documents). C'est pourquoi nous allons écrire une API en NODE.js que nous exposerons publiquement.

5.1 – Le script de recherche

Essayons de créer maintenant notre propre API de recherche.

Créer un nouveau fichier pour notre API, nous l'appellerons : `server/search.js`. Voici son contenu :

```
const { client, index, type } = require('./connection')

module.exports = {
  /** Query ES index for the provided term */
  queryTerm (term, offset = 0) {
    const body = {
      from: offset,
      query: { match: {
        text: {
          query: term,
          operator: 'and',
          fuzziness: 'auto'
        } } },
      highlight: { fields: { text: {} } }
```

```
    }

    return client.search({ index, type, body })
  }
}
```

Notre module de recherche définit une fonction de recherche simple, qui effectuera une requête de correspondance en utilisant le terme saisi.

Voici les paramètres de notre requête :

- `from` : Permets la pagination des résultats.
- `query` : Permets de spécifier les termes de la recherche.
- `Operator` : Permets de modifier le comportement de la requête.
- `fuzziness` Permets d'ajuster la tolérance pour les fautes d'orthographe. Plus la valuer est élevée, plus la correction dans les résultats est élevée. Par exemple, `fuzziness: 1` permettrait de renvoyer `Patrick` avec comme terme de recherche `Patricc`
- `highlights` : Permets de surligner les termes de recherche dans les résultats grâce à une balise HTML.

N'hésitez pas à jouer avec les paramètres pour personnaliser vos résultats de recherches. Pour plus d'information consulter le DSL Elastic Full-Text Query.

6 - API

Après avoir créé notre module de recherche, il nous reste plus qu'à écrire notre API front.

6.0 - API serveur

Remplacer le contenu du fichier `server/app.js` par le contenu suivant :

```
const Koa = require('koa')
const Router = require('koa-router')
const joi = require('joi')
const validate = require('koa-joi-validate')
const search = require('./search')

const app = new Koa()
const router = new Router()

// Log each request to the console
app.use(async (ctx, next) => {
```

```
  const start = Date.now()
  await next()
  const ms = Date.now() - start
  console.log(`${ctx.method} ${ctx.url} - ${ms}`)
})

// Log percolated errors to the console
app.on('error', err => {
  console.error('Server Error', err)
})

// Set permissive CORS header
app.use(async (ctx, next) => {
  ctx.set('Access-Control-Allow-Origin', '*')
  return next()
})

// ADD ENDPOINTS HERE

const port = process.env.PORT || 3000

app
  .use(router.routes())
  .use(router.allowedMethods())
  .listen(port, err => {
    if (err) throw err
    console.log(`App Listening on Port ${port}`)
  })
```

Ce code importe différentes dépendances nécessaires au bon fonctionnement de notre projet, il configure aussi la gestion et le traitement des erreurs.

6.1 – Lier les requêtes avec notre point d'accès

Nous allons ajouter les routes nécessaires pour notre API pour pouvoir effectuer nos requêtes.

Insérer le code suivant après le commentaire : `// ADD ENDPOINTS HERE` situé dans le fichier `server/app.js`.

```
/**
 * GET /search
 * Search for a term in the library
```

```
 */
router.get('/search', async (ctx, next) => {
    const { term, offset } = ctx.request.query
    ctx.body = await search.queryTerm(term, offset)
  }
)
```

Il ne vous reste plus qu'à redémarrer votre conteneur docker avec la commande habituelle :

`docker-compose up -d -build`

Rendez-vous dans votre navigateur favoris et nous allons refaire la même requête que tous à l'heure mais en utilisant notre propre API :

`http://localhost:3000/search?term=java`

La réponse devrait être similaire à notre test précédent lors de l'utilisation de l'API d'ES.


```
{
    "took": 242,
    "timed_out": false,
    "_shards": {
        "total": 5,
        "successful": 5,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": 93,
        "max_score": 13.356944,
        "hits": [{
            "_index": "library",
            "_type": "novel",
            "_id": "eHYHJmEBpQg9B4622421",
            "_score": 13.356944,
            "_source": {
                "author": "Charles Darwin",
                "title": "On the Origin of Species",
                "location": 1080,
                "text": "Java, plants of, 375."
            },
            "highlight": {
```

```
                "text": ["<em>Java</em>, plants of, 375."]
            }
        }, {
            "_index": "library",
            "_type": "novel",
            "_id": "2HUHJmEBpQg9B462xdNg",
            "_score": 9.030668,
            "_source": {
                "author": "Unknown Author",
                "title": "The King James Bible",
                "location": 186,
                "text": "10:4 And the sons of Javan; Elishah, and Tarshish, Kittim, and
Dodanim."
            },
            "highlight": {
                "text": ["10:4 And the sons of <em>Javan</em>; Elishah, and Tarshish,
Kittim, and Dodanim."]
            }
        }
        ...
      ]
   }
}
```

6.2 – Validation des données

Vous pensiez avoir terminé ? Hé bien non, toujours dans un souci de sécurité, nous devons valider les données envoyées par l'utilisateur. En effet des données non valides ou manquantes génèreraient des erreurs de serveur.

Nous allons ajouter un middleware qui va intercepter la requête avant son traitement et qui va valider les données. Pour cela nous allons nous aider de la librairie Koa-Joi-Validate ajouter précédemment.

```
/**
 * GET /search
 * Search for a term in the library
 * Query Params -
 * term: string under 60 characters
 * offset: positive integer
 */
router.get('/search',
  validate({
    query: {
```

```
      term: joi.string().max(60).required(),
      offset: joi.number().integer().min(0).default(0)
    }
  }),
  async (ctx, next) => {
    const { term, offset } = ctx.request.query
    ctx.body = await search.queryTerm(term, offset)
  }
)
```

Maintenant, si vous redémarrez votre serveur et que vous vous rendez sur votre navigateur en indiquant une recherche vide :

`http://localhost:3000/search`

Vous devriez avoir en retour un code HTTP 400 et le message suivant dans vos logs :

```
Invalid URL Query - child "term" fails because ["term" is required]
```

Note : Pour afficher les logs de votre application Node, vous pouvez exécuter la commande suivante : `docker-compose logs -f api`.

7 - APPLICATION FRONT-END

Maintenant que notre route `/search` est en place, connectons une petite application web pour tester notre API.

7.0 – Application Vue.js

Pour la gestion du front, nous utiliserons le framework Vue.js.

Ajouter un nouveau fichier : `/public/app.js` qui contiendra notre application Vue.js

```
const vm = new Vue ({
  el: '#vue-instance',
  data () {
    return {
      baseUrl: 'http://localhost:3000', // API url
      searchTerm: 'Hello World', // Default search term
      searchDebounce: null, // Timeout for search bar debounce
      searchResults: [], // Displayed search results
      numHits: null, // Total search results found
      searchOffset: 0, // Search result pagination offset

      selectedParagraph: null, // Selected paragraph object
```

```
      bookOffset: 0, // Offset for book paragraphs being displayed
      paragraphs: [] // Paragraphs being displayed in book preview window
    }
  },
  async created () {
    this.searchResults = await this.search() // Search for default term
  },
  methods: {
    /** Debounce search input by 100 ms */
    onSearchInput () {
      clearTimeout(this.searchDebounce)
      this.searchDebounce = setTimeout(async () => {
        this.searchOffset = 0
        this.searchResults = await this.search()
      }, 100)
    },
    /** Call API to search for inputted term */
    async search () {
      const response = await axios.get(`${this.baseUrl}/search`, { params: { term:
this.searchTerm, offset: this.searchOffset } })
      this.numHits = response.data.hits.total
      return response.data.hits.hits
    },
    /** Get next page of search results */
    async nextResultsPage () {
      if (this.numHits > 10) {
        this.searchOffset += 10
        if (this.searchOffset + 10 > this.numHits) { this.searchOffset = this.numHits -
10}
        this.searchResults = await this.search()
        document.documentElement.scrollTop = 0
      }
    },
    /** Get previous page of search results */
    async prevResultsPage () {
      this.searchOffset -= 10
      if (this.searchOffset < 0) { this.searchOffset = 0 }
      this.searchResults = await this.search()
      document.documentElement.scrollTop = 0
    }
  }
})
```

Cette application est assez simple : Nous définissons des propriétés puis des méthodes qui nous permettent d'appeler notre API. Ensuite nous retournons les résultats paginés. Expliquer le fonctionnement de Vue.js ne fait pas partie de ce cours. Si vous souhaitez approfondir ce framework, je vous invite à consulter https://vuejs.org/v2/guide/.

7.1 - HTML

Remplacer le contenu du fichier suivant `/public/index.html` afin de charger notre application Vue.js contenant une interface de recherche basique.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Elastic Library</title>
  <meta name="description" content="Literary Classic Search Engine.">
  <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1,
user-scalable=no">
  <link href="https://cdnjs.cloudflare.com/ajax/libs/normalize/7.0.0/normalize.min.css"
rel="stylesheet" type="text/css" />
  <link href="https://cdn.muicss.com/mui-0.9.20/css/mui.min.css" rel="stylesheet"
type="text/css" />
  <link href="https://fonts.googleapis.com/css?family=EB+Garamond:400,700|Open+Sans"
rel="stylesheet">
  <link href="styles.css" rel="stylesheet" />
</head>
<body>
<div class="app-container" id="vue-instance">
    <!-- Search Bar Header -->
    <div class="mui-panel">
      <div class="mui-textfield">
        <input v-model="searchTerm" type="text" v-on:keyup="onSearchInput()">
        <label>Search</label>
      </div>
    </div>

    <!-- Search Metadata Card -->
    <div class="mui-panel">
      <div class="mui--text-headline">{{ numHits }} Hits</div>
      <div class="mui--text-subhead">Displaying Results {{ searchOffset }} - {{
searchOffset + 9 }}</div>
    </div>

    <!-- Top Pagination Card -->
    <div class="mui-panel pagination-panel">
```

```
        <button class="mui-btn mui-btn--flat" v-on:click="prevResultsPage()">Prev
Page</button>
        <button class="mui-btn mui-btn--flat" v-on:click="nextResultsPage()">Next
Page</button>
    </div>

    <!-- Search Results Card List -->
    <div class="search-results" ref="searchResults">
      <div class="mui-panel" v-for="hit in searchResults" v-
on:click="showBookModal(hit)">
        <div class="mui--text-title" v-html="hit.highlight.text[0]"></div>
        <div class="mui-divider"></div>
        <div class="mui--text-subhead">{{ hit._source.title }} - {{ hit._source.author
}}</div>
        <div class="mui--text-body2">Location {{ hit._source.location }}</div>
      </div>
    </div>

    <!-- Bottom Pagination Card -->
    <div class="mui-panel pagination-panel">
        <button class="mui-btn mui-btn--flat" v-on:click="prevResultsPage()">Prev
Page</button>
        <button class="mui-btn mui-btn--flat" v-on:click="nextResultsPage()">Next
Page</button>
    </div>

    <!-- INSERT BOOK MODAL HERE -->
</div>
<script src="https://cdn.muicss.com/mui-0.9.28/js/mui.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.3/vue.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/axios/0.17.0/axios.min.js"></script>
<script src="app.js"></script>
</body>
</html>
```

7.2 - CSS

On va ajouter un peu de style. Ajouter un nouveau fichier : `/public/styles.css`, qui contiendra un style personnalisé :

```
body { font-family: 'EB Garamond', serif; }

.mui-textfield > input, .mui-btn, .mui--text-subhead, .mui-panel > .mui--text-headline {
  font-family: 'Open Sans', sans-serif;
```

```
}

.all-caps { text-transform: uppercase; }
.app-container { padding: 16px; }
.search-results em { font-weight: bold; }
.book-modal > button { width: 100%; }
.search-results .mui-divider { margin: 14px 0; }

.search-results {
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
  justify-content: space-around;
}

.search-results > div {
  flex-basis: 45%;
  box-sizing: border-box;
  cursor: pointer;
}

@media (max-width: 600px) {
  .search-results > div { flex-basis: 100%; }
}

.paragraphs-container {
  max-width: 800px;
  margin: 0 auto;
  margin-bottom: 48px;
}

.paragraphs-container .mui--text-body1, .paragraphs-container .mui--text-body2 {
  font-size: 1.8rem;
  line-height: 35px;
}

.book-modal {
  width: 100%;
  height: 100%;
  padding: 40px 10%;
  box-sizing: border-box;
  margin: 0 auto;
  background-color: white;
```

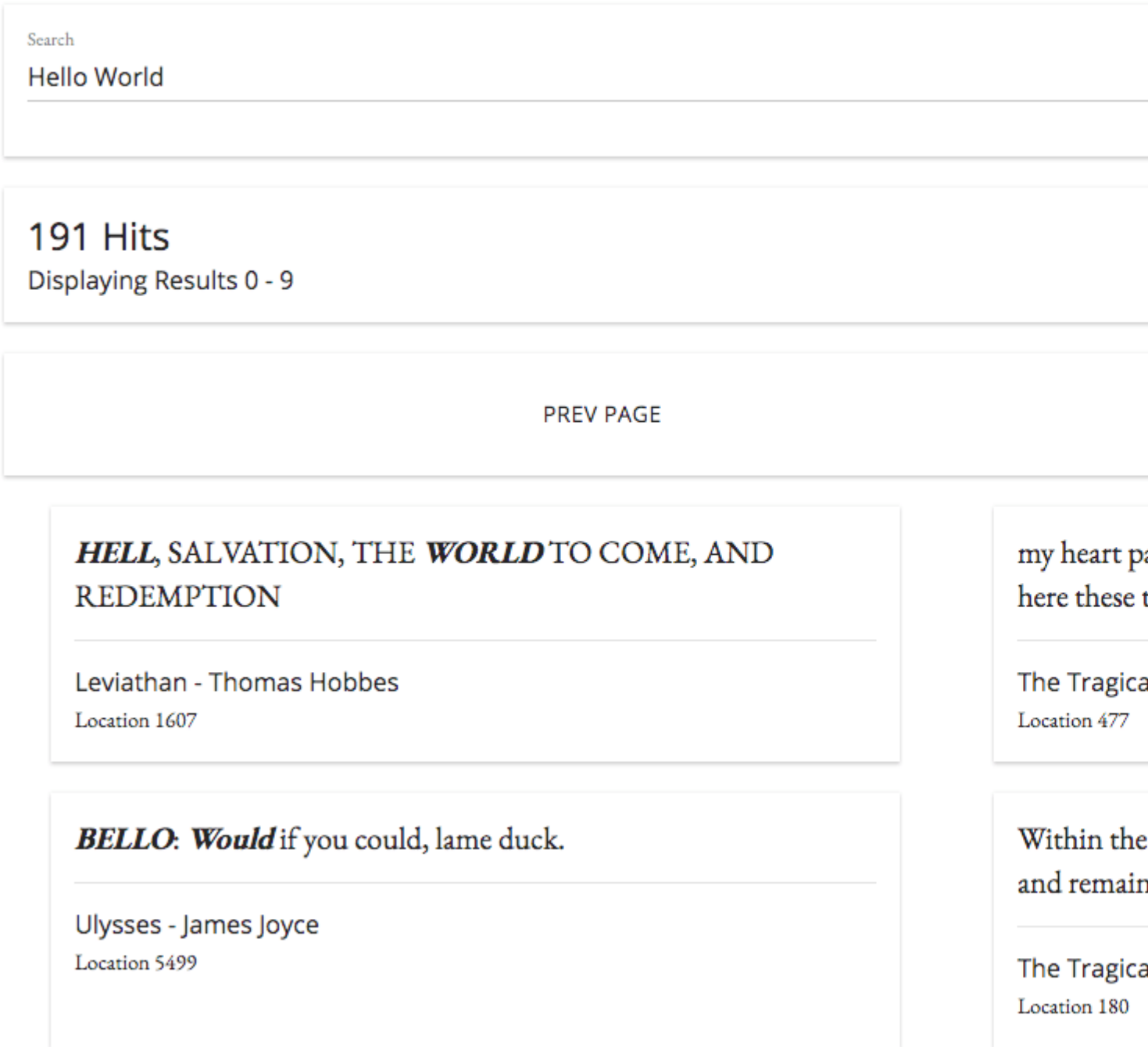
```
  overflow-y: scroll;
  position: fixed;
  top: 0;
  left: 0;
}

.pagination-panel {
  display: flex;
  justify-content: space-between;
}

.title-row {
  display: flex;
  justify-content: space-between;
  align-items: flex-end;
}

@media (max-width: 600px) {
  .title-row{
    flex-direction: column;
    text-align: center;
    align-items: center
  }
}

.locations-label {
  text-align: center;
  margin: 8px;
}

.modal-footer {
  position: fixed;
  bottom: 0;
  left: 0;
  width: 100%;
  display: flex;
  justify-content: space-around;
  background: white;
}
```

7.3 – Testons

Rendez-vous dans votre navigateur et taper l'URL suivante :

`localhost:8080`

Vous devriez voir une interface de recherche simple avec des résultats paginés. Essayez de taper dans la barre de recherche supérieure pour trouver des correspondances de différents termes.



Si vous essayez de cliquer sur un résultat, rien ne se passe - nous avons encore une fonctionnalité à ajouter à l'application.

8 – DETAILS D'UN LIVRE

Ce serait bien de pouvoir cliquer sur chaque résultat de recherche et d'en visualiser le contenu. Codons tous ça.

8.0 – Ajout de la requête ES

Tout d'abord, nous devons définir une requête simple pour obtenir une plage de paragraphes d'un livre.

Ajouter la fonction suivante dans le bloc `module.exports` situé dans le fichier `server/search.js`.

```
/** Get the specified range of paragraphs from a book */
getParagraphs (bookTitle, startLocation, endLocation) {
  const filter = [
    { term: { title: bookTitle } },
    { range: { location: { gte: startLocation, lte: endLocation } } }
  ]

  const body = {
    size: endLocation - startLocation,
    sort: { location: 'asc' },
    query: { bool: { filter } }
  }

  return client.search({ index, type, body })
}
```

Cette nouvelle fonction renverra un tableau ordonné de paragraphes entre les emplacements de début et de fin d'un livre.

8.1 – Ajout d'un point d'accé à L'API

Lions maintenant cette fonction à notre API

Ajouter le contenu suivant à notre fichier `server/app.js`, sous notre route `/search`.

```
/**
 * GET /paragraphs
 * Get a range of paragraphs from the specified book
 * Query Params -
 * bookTitle: string under 256 characters
 * start: positive integer
 * end: positive integer greater than start
 */
router.get('/paragraphs',
  validate({
    query: {
```

```
      bookTitle: joi.string().max(256).required(),
      start: joi.number().integer().min(0).default(0),
      end: joi.number().integer().greater(joi.ref('start')).default(10)
    }
  }),
  async (ctx, next) => {
    const { bookTitle, start, end } = ctx.request.query
    ctx.body = await search.getParagraphs(bookTitle, start, end)
  }
)
```

8.2 – Ajout de la partie UI

Maintenant que notre API est en place, ajoutons quelques fonctionnalités pour interroger et afficher les pages complètes du livre.

Ajouter les fonctions suivantes dans le fichier `/public/app.js`.

```
    /** Call the API to get current page of paragraphs */
    async getParagraphs (bookTitle, offset) {
      try {
        this.bookOffset = offset
        const start = this.bookOffset
        const end = this.bookOffset + 10
        const response = await axios.get(`${this.baseUrl}/paragraphs`, { params: {
bookTitle, start, end } })
        return response.data.hits.hits
      } catch (err) {
        console.error(err)
      }
    },
    /** Get next page (next 10 paragraphs) of selected book */
    async nextBookPage () {
      this.$refs.bookModal.scrollTop = 0
      this.paragraphs = await this.getParagraphs(this.selectedParagraph._source.title,
this.bookOffset + 10)
    },
    /** Get previous page (previous 10 paragraphs) of selected book */
    async prevBookPage () {
      this.$refs.bookModal.scrollTop = 0
      this.paragraphs = await this.getParagraphs(this.selectedParagraph._source.title,
this.bookOffset - 10)
    },
```

```
    /** Display paragraphs from selected book in modal window */
    async showBookModal (searchHit) {
      try {
        document.body.style.overflow = 'hidden'
        this.selectedParagraph = searchHit
        this.paragraphs = await this.getParagraphs(searchHit._source.title,
searchHit._source.location - 5)
      } catch (err) {
        console.error(err)
      }
    },
    /** Close the book detail modal */
    closeBookModal () {
      document.body.style.overflow = 'auto'
      this.selectedParagraph = null
    }
```

Ces cinq fonctions fournissent la logique de téléchargement et de pagination de pages (dix paragraphes chacune) dans un livre.

Il ne reste plus qu'à ajouter une interface utilisateur pour afficher les pages du livre. Inseré le code suivant sous la balise `<!-- INSERT BOOK MODAL HERE -->` situé dans le fichier `/public/index.html`.

```
    <!-- Book Paragraphs Modal Window -->
    <div v-if="selectedParagraph" ref="bookModal" class="book-modal">
      <div class="paragraphs-container">
        <!-- Book Section Metadata -->
        <div class="title-row">
          <div class="mui--text-display2 all-caps">{{ selectedParagraph._source.title
}}</div>
          <div class="mui--text-display1">{{ selectedParagraph._source.author }}</div>
        </div>
        <br>
        <div class="mui-divider"></div>
        <div class="mui--text-subhead locations-label">Locations {{ bookOffset - 5 }} to
{{ bookOffset + 5 }}</div>
        <div class="mui-divider"></div>
        <br>

        <!-- Book Paragraphs -->
        <div v-for="paragraph in paragraphs">
```