Node et Elastic TP-07

Configuration «minimale» de ElasticSearch

Au cours du TP-02 vous avez installé un service ElasticSearch.

Vérifiez si Elastic est disponible avec Curl :

curl -X GET "http://localhost:9200/"

On va utiliser Curl pour créer notre index « local_users »

curl -X PUT "http://localhost:9200/local_users/"

On vérifie la création :

curl -X GET "http://localhost:9200/local_users"

Voilà notre index est prêt. Pour la suite de ce cours on n'utilisera plus curl on passera par le client Esclient de node.

Mais au cas où ! La commande curl pour supprimer l'index.

curl -X DELETE "http://localhost:9200/local_users"

Pour en savoir plus sur Elasticsearch et Curl :
https://riptutorial.com/fr/elasticsearch/topic/3703/commandes-curl

Retour dans le serveur

Dans le TP précédent on à utiliser node comme serveur de données… Ce qui n'est pas son rôle, on va maintenant le configurer pour son rôle de middle.

Recevoir des demandes du front, les valider / transformer, échanger avec l'API qui a la donnée, et faire le chemin dans l'autre sens.

Au début, on a créé un seul fichier dans service pour gérer users.

On va maintenant l'éclater en 3.

users-repository qui va gérer les échanges avec la base de données

users-handler qui va s'occuper des validation et transformation avant l'envoi au repository.

users qui s'occupe uniquement du routage. Pour plus de clarté on pourrait le renommer users-routage et partager un objet usersRouter plutôt que router.

Il faudra aussi ajouter le client elasticsearch Esclient.

Procédons par étape :

On installe le module client EsClient :

npm install @elastic/elasticsearch

On va maintenant créer le service es-client.js :

```
import elasticsearch from '@elastic/elasticsearch';

const esClient = new elasticsearch.Client({
  node: 'http://localhost:9200',
  log: 'info',
  apiVersion: '7.5',
  maxSockets: 20,
});

export default esClient;
```

On a maintenant un client que l'on peut injecter dans d'autres services.

On va modifier le fichier de routage users.js

On le renomme en users-routage.js

Et on le modifie :

On va maintenant importer en plus de express le users-handler (on créera le fichier plus tard).

Import usersHandler from './users-handler';

On renomme notre router au passage

const usersRouter = express.Router();

Et on lui ajoute les 2 routes get et post :

usersRouter.get('/', usersHandler.getUsers);

usersRouter.post('/', usersHandler.create);

Et on modifie l'export :

export default usersRouter;

Comme on a modifié l'export on modifie l'import dans le server.js, ainsi que le routage de /users

On pourrait croire que c'est bon mais !

Maintenant on va devoir solliciter une API externe pour avoir la donnée, l'appel à ces 2 routes va donc être asynchrone.

Express va nous aider grâce à express-async-handler :

On l'installe :

npm install --save express-async-handler

On l'importe dans users-routage :

import asyncHandler from 'express-async-handler';

et on fait les appels dans l'asyncHandler.

usersRouter.get('/', asyncHandler(usersHandler.getUsers));

usersRouter.post('/', asyncHandler(usersHandler.create));

On passe maintenant au handler

On crée le fichier users-handler.js dans service.

Il va falloir gérer plusieurs choses :

- importer le users-repository (que l'on créera plus tard) ;

- créer les fonctions getUsers et create,

- créer une fonction pour vérifier si l'user existe déjà (firstName === firstName)

- gérer les appels asynchrones.

En avant !

On importe le repository

import usersRep from './users-repository';

pour gérer l'asynchrone on va utiliser async-await

on crée la fonction getUsers :

```
 async function getUsers(req, res) {
  try {
    const result = await usersRep.getAll();
    const finalArray = [];
    for (let obj of result.body.hits.hits) {
      finalArray.push(obj. source);
    }
    res.send(finalArray);
  } catch (e) {
    res.status(400).end();
  }
}
```

On fait nos appels dans un try-catch pour protéger notre code.

Lors de l'appel à getAll() on utilise await pour attendre la résolution avant de continuer.

Les résultats obtenus sont dans le tableau result.body.hits.hits et les objets users sont stockés dans _source. On parcourt le tableau pour récupérer les objets users et on les push dans le tableau de résultat. Que l'on retourne au front.

Maintenant on crée la fonction create(),

```
async function create(req, res) {
  res.set('Content-Type', 'application/json');
  try {
    const userBool = await userExist(req.body.firstName);
    if(userBool) {
      res.send({});
    } else {
      await usersRep.store(req.body);
      res.send({firstName=  'ok');
    }
  } catch (e) {
    res.status(400).end();
  }
}
```

On protège l'appel dans un try-catch, avant d'enregistrer le user on va vérifier s'il existe en appelant userExist() et en lui passant le firstName.
Si il existe on retourne un objet vide pour lui dire qu'il n'a pas été créé, sinon on fait la création et on renvoie l'objet au front pour lui dire qu'il a été créé.

Au tour de la fonction userExist() :

Elle va retourner un boolean pour dire si l'utilisateur existe.

```
async function userExist (firstName) {
  try {
    const result = await usersRep.getUser(firstName);
    return result.body.hits.total.value > 0 ? true : false;
  } catch (e) {
    console.log('error getting user', e);
    return false;
  }
}
```

S'il y a plus de 0 résultat qui correspondent à firstName === firstName elle renvoie true.

Reste à exporter nos fonctions :

```
export default {
  getUsers,
  create,
  userExist,
};
```

Maintenant on attaque users-repository !

Créer le fichier, puis on l'édite :

On commence par importer le client :

```
import esClient from './es-client';
```

On crée une variable index avec le nom de l'index sur le serveur

```
const index = 'local_users';
```

Et on va construire un gestionnaire d'erreur pour elasticSearch :

```
const handleElasticsearchError = (error) => {
  if (error.status === 404) {
    throw new Error('User Not Found', 404);
  }
  throw new Error(error.msg, error.status || 500);
};
```

Puis on crée nos fonctions en utilisant then-catch pour gérer les promesses :

```
const getAll = () => esClient.search({
  index,
}).then(response => response).catch((error) => {
  handleElasticsearchError(error);
});

const store = user => esClient.index({
  index,
  refresh: 'true',
  body: user,
}).then(response => response.status).catch((error) => {
  handleElasticsearchError(error);
});

const getUser = firstName => esClient.search({
  index,
  body: {"query": {
    "match":{
      "firstName": {
        "query": firstName
      }
    }
  }
  },
}).then(response => { response; })
  .catch((error) => {
    handleElasticsearchError(error);
});
```

Puis on exporte nos 3 fonctions.

```
export default {
  getUser,
  store,
  getAll,
};
```

Pour chaque fonction on utilise le then-catch pour renvoyer les erreurs Elasticsearch au handler.

Dans getUser() on va utiliser le Query Dsl de elasticsearch pour faire la recherche sur le firstName.

Voilà, On peut lancer npm start. La donnée est maintenant hébergée par une API tiers et Node s'occupe uniquement du transfert / validation de la données.

Testez !

Exercice : crée un flux pour supprimer les users.

Donc on reprend :

Dans le front :

On va ajouter dans le html de user-list un bouton sur chaque item user, qui va passer le firstName à une fonction onSuppress

Ajouter dans le ts de user list, la fonction onSuppress qui va demander une confirmation puis appeler en cas de confirmation une fonction suppressUser en lui passant le firstName.

Ajouter la fonction suppressUser qui va appeler le service http puis gérer le retour en rechargeant la liste s'il y a eu une suppression.

Crée un route dans le http.service pour envoyer la demande de suppression.

Dans le middle :

On ajouter une route delete dans le routage,

Dans le handler, Une fonction userDelete qui va vérifier si le user existe et le supprimer s'il existe.

Dans le repositroy, une fonction qui va demander la suppression à ElasticSearch.

Les infos nécessaire :

Dans la fonction suppressUser(firstName) dans le subscribe :

```
 if(result.status === 200) {
  this.ngOnInit();
}
```

On teste le code de retour, si c'est 200 on recharge les users en appelant ngOnInit.

Dans la fonction suppressUser, on va passer le firstName dans l'Url :

this.serverUrl+'users/'+firstName

Direction le serveur (ne pas oublier le build du front etc …)

Dans le routage, on spécifie l'url '/:id' qui va affecter le string après le users/ à req.params.id .
Dans le Handler on teste si le user existe en utilisant req.params.id qui contient le firstName du user.

S'il n'existe pas on utilise res.status(404).end() pour lui dire que le user n'est pas supprimé.

Sinon on retourne le result de l'appel à usersRep.remove(req.params.id) ;

Dans remove on va appeler esClient.deleteByQuery, et passer dans le body une query Dsl du même type que pour le getUser.

Correction plus bas

{{stagiaire.firstName}} ! Es-tu sûr que tout marche ?

Si oui pas besoin d’aller plus loin !

Sinon, il faut vérifier dans la console les échanges dans Network !

Error 404

La page que vous cherchez n'existe pas !

Bon ok !

Côté Client :

User-list.component.html :

```
    <button class="btn btn-sm btn-danger"
      (click)="onSuppress(user.firstName)">Supprimer</button>
```

User-list.component.ts :

```
  onSuppress(firstName) {
    if(confirm('Etes-vous sûr de la supprimer ?')) {
      this.suppressUser(firstName);
    } else {
      return null;
    }
  }

  suppressUser(firstName) {
    this.http.suppUser(firstName).subscribe((result)=>{
      if(result.status === 200) {
        this.ngOnInit();
      } else {
        alert('le User n\'existe pas !');
      }
    })
  }
```

http.service.ts :

```
  public suppUser(firstName): Observable<any> {
    return this.http.delete<any>(this.serverUrl+'users/'+firstName, {observe: 'response'});
  }
```

Côté Serveur :

users-routage.js :

```
usersRouter.delete('/:id', asyncHandler(usersHandler.userDelete));
```

users-handler.js :

```
async function userDelete(req, res) {
  try {
    const userBool = await userExist(req.params.id);
    if(!userBool) {
      res.status(404).end();
    } else {
      const result = await usersRep.remove(req.params.id);
      res.send(result);
    }
  } catch (e) {
```

```
    res.status(error.status || 400).end();
  }
}
```

users-repository.js :

```
const remove = firstName => esClient.deleteByQuery({
  index,
  refresh: 'true',
  body: {
    "query": {
      "match":{
        "firstName":{
          "query": firstName
        }
      }
    }
  },
}).then(response => response).catch((error) => {
  handleElasticsearchError(error);
});
```