

PORJECT DÉVELOPPEMENT À BASE DE COMPOSANTS

FELAH MOHAMED AMINE
GUIDARA FATMA
ABID AYCHA

Développement à Base de Composants

**Dans ce project on a crée deux microservices
d'écriture et de lecture des bouquins
avec le framework **spring boot** 2.2. 1**



- Le premier microservice permettrait de lire dans un catalogue
- Le deuxième microservice permettrait d'écrire dans un catalogue

- Le premier microservice de lecture utiliserait la base de donnée **elasticsearch**



elasticsearch

- Le deuxième microservice d'écriture utiliserait la base de donnée **mongodb**



- La synchronisation des deux bases de données se ferait avec **talend**



- Spring jouerait le rôle de l'apigateway et permettrait de orchestrait tous le flux de chargement de données

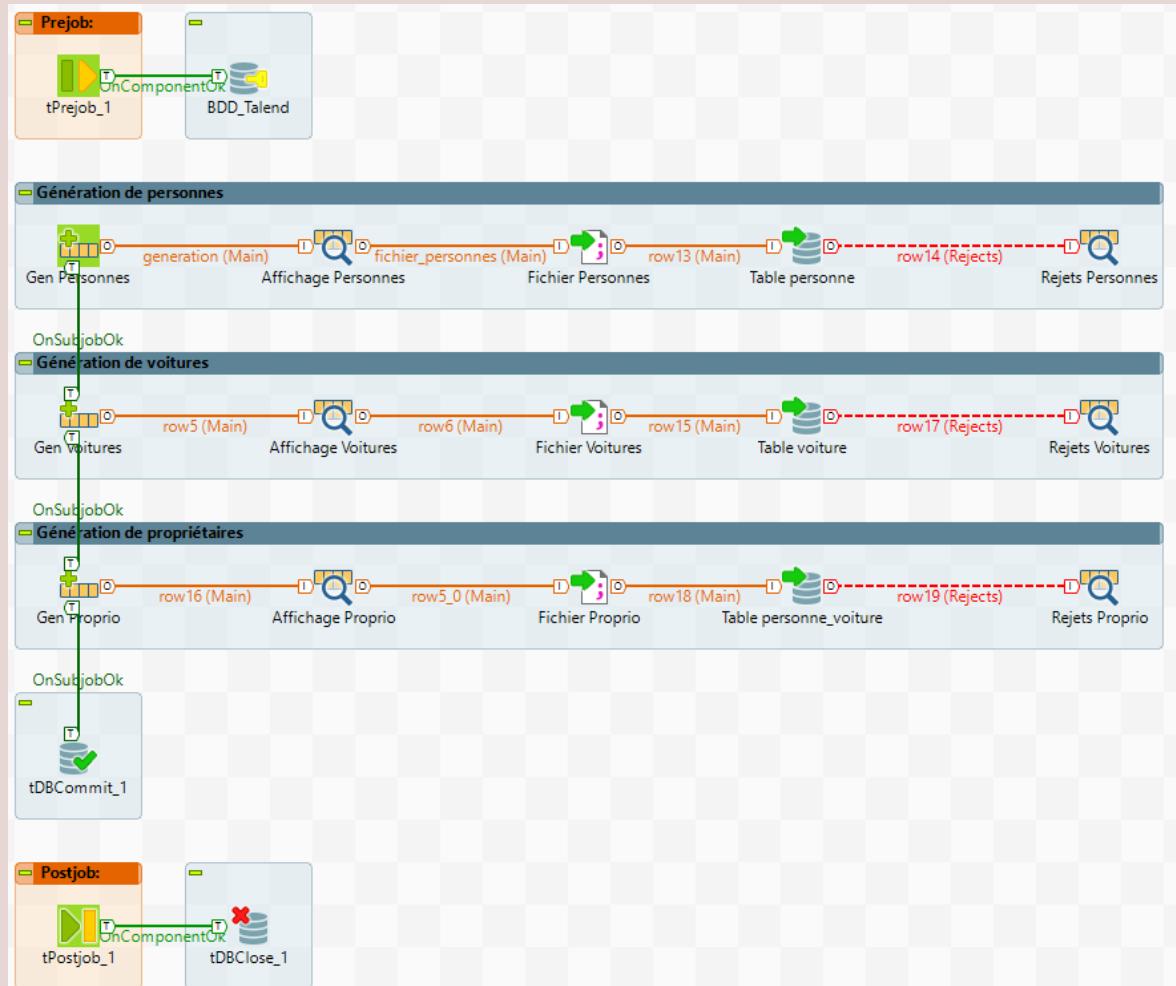


- Elasticsearch est utilisé pour le query ou bien la lecture puisque il est très rapide, il est également une plate-forme de recherche en temps quasi réel, ce qui signifie que la latence entre le moment où un document est indexé et celui où il devient consultable est très courte, généralement une seconde. Par conséquent, Elasticsearch est bien adapté aux cas d'utilisation urgents.
- Mongodb est utilisé pour la commande ou bien l'écriture puisqu'il est flexible on peut garder les données en forme **JSON** contrairement au base de donnée relationnelle sous forme de table ainsi il est expressive, scalabe et facile à repliquer

```
{  
  "personId": "1",  
  "fname": "amin",  
  "lname": "felah"  
}
```

example d'un object JSON

- Talend et un service qui permet de synchroniser les bases de données est d'ajouter les données manquant d'une base a partir de l'autre ceci est automatisé grace a des jobs qui sont asynchrones, reactive grace a des snapshots ce qui accentue l'aspect d'event sourcing qui declance un event appartir de l'autre



- Les Microservices et les services sont tous enregistrer dans des conteneurs **docker** qui enregistrer les images de notre solutions



The screenshot shows the Docker desktop application interface. The left sidebar has options: Containers / Apps, Images (selected), Volumes, and Dev Environments (PREVIEW). The main panel title is "Images on disk" with "3 images" and "Total size: 2.49 GB". It shows a table of local repositories:

NAME	TAG	IMAGE ID	CREATED	SIZE	
elastic-boot-master_project-t...	latest	d79cdf3b25ca	1 day ago	1.26 GB	
elasticsearch	IN USE	7.16.3	3a5e93284781	21 days ago	610.98 MB
mongo	IN USE	latest	ee13a1eacac9	21 days ago	696.14 MB

At the bottom, it says "Not connected" and lists benefits: "Store and backup your images remotely", "Collaborate with your team", "Unlock vulnerability scanning for greater security", and "Connect for free". There is also a "Sign in" button.

- Pour manager docker on utiliserait le fichier **dockerFile** qui permet de build et d'exporter notre project ainsi que **docker-composer** pour demarer tous les services qu'on a besoin avec tous leurs dependances

```
1 FROM maven:3-jdk-11
2
3 ADD . /cqrss
4 WORKDIR /cqrss
5
6 RUN ls -l
7
8 RUN mvn clean install -U
9
10 CMD mvn spring-boot:run
```

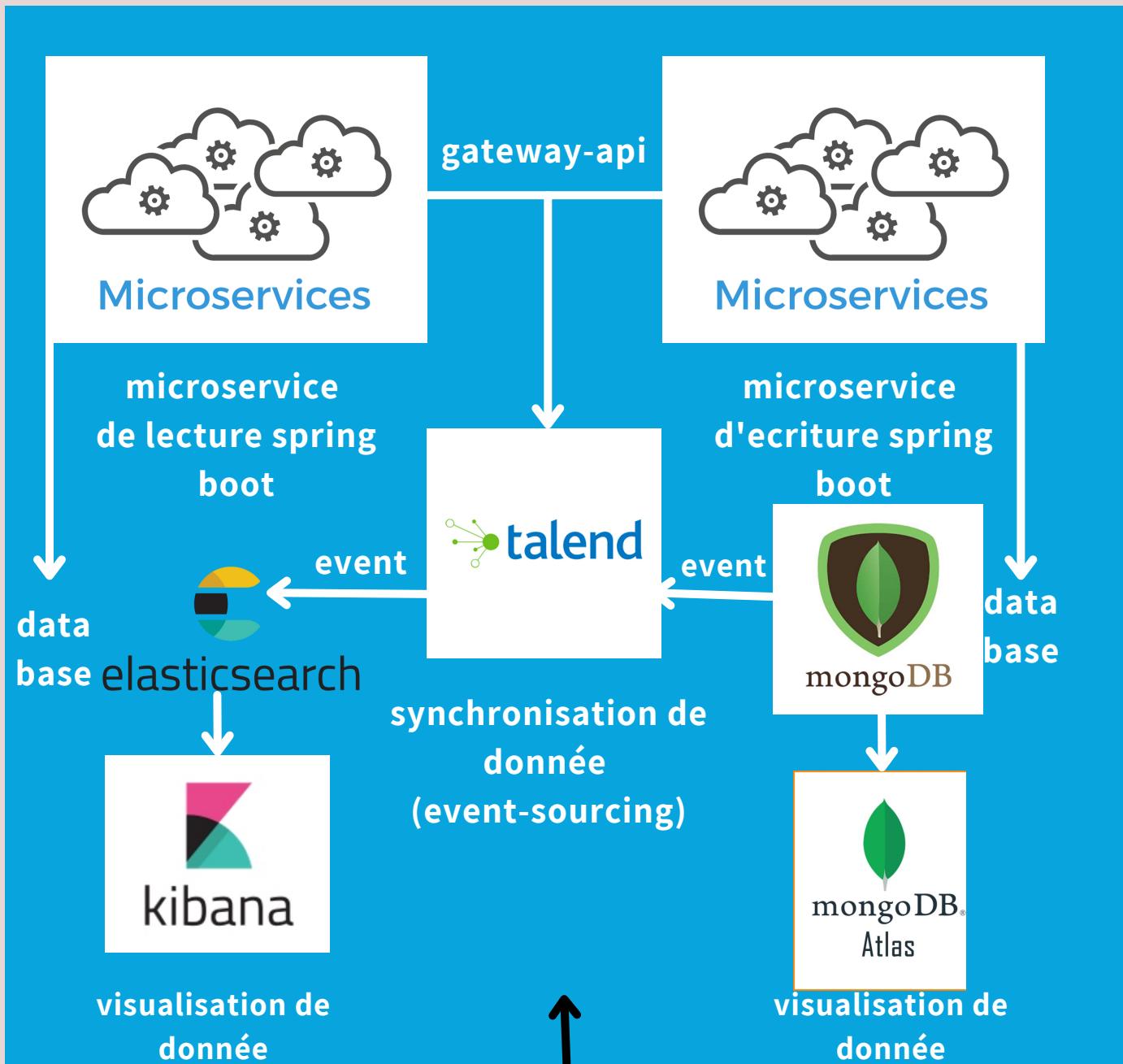
DockerFile

```
mongo:
  image: mongo
  restart: on-failure
  ports:
    - 27017:27017
  environment:
    MONGO_INITDB_ROOT_USERNAME: root
    MONGO_INITDB_ROOT_PASSWORD: example
    MONGO_INITDB_DATABASE: user
    MONGO_DB_USERNAME: root
    MONGO_DB_PASSWORD: example
  networks:
    - my-cqrss-network

elasticsearch:
  image: elasticsearch:7.16.3
  container_name: elasticsearch
  environment:
    - node.name=elasticsearch
    - xpack.security.enabled=false
    - discovery.type=single-node
    - cluster.name=docker-cluster
  ulimits:
    memlock:
      soft: -1
      hard: -1
    nofile:
      soft: 65536
      hard: 65536
    cap_add:
      - IPC_LOCK
  volumes:
    - elasticsearch-data:/usr/share/elasticsearch/data
  ports:
    - 9200:9200
  networks:
    - my-cqrss-network
```

Docker-compose

- L'architecture du project



Conteneaire docker

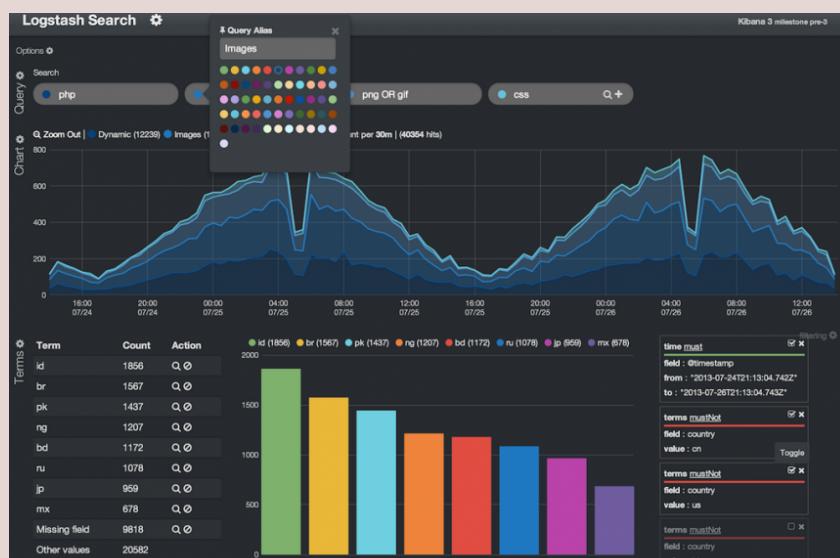
MongoDB ATLAS

pour la visualisation de notre database mongo

The screenshot shows the MongoDB Atlas interface. On the left, there's a sidebar with sections for DEPLOYMENT (Databases), DATA SERVICES (Triggers, Data API, SECURITY), and SECURITY (Database Access, Network Access, Advanced). The main area is titled 'Cluster0' and has tabs for Overview, Real Time, Metrics, Collections (which is selected), Search, Profiler, Performance Advisor, Online Archive, and Cmd Line Tools. Below the tabs, it says 'DATABASES: 1 COLLECTIONS: 1'. A 'myFirstDatabase' section contains a 'person' collection. The 'myFirstDatabase.person' page displays storage size (36KB), total documents (4), and index sizes (36KB). It includes tabs for Find, Indexes, Schema Anti-Patterns, Aggregation, and Search Indexes. A query result table shows one document: {_id: "1", fname: "amin", lname: "felah"}. There are buttons for 'FILTER', 'OPTIONS', 'Apply', and 'Reset'.

Kibana

pour la visualisation de notre database elasticsearch

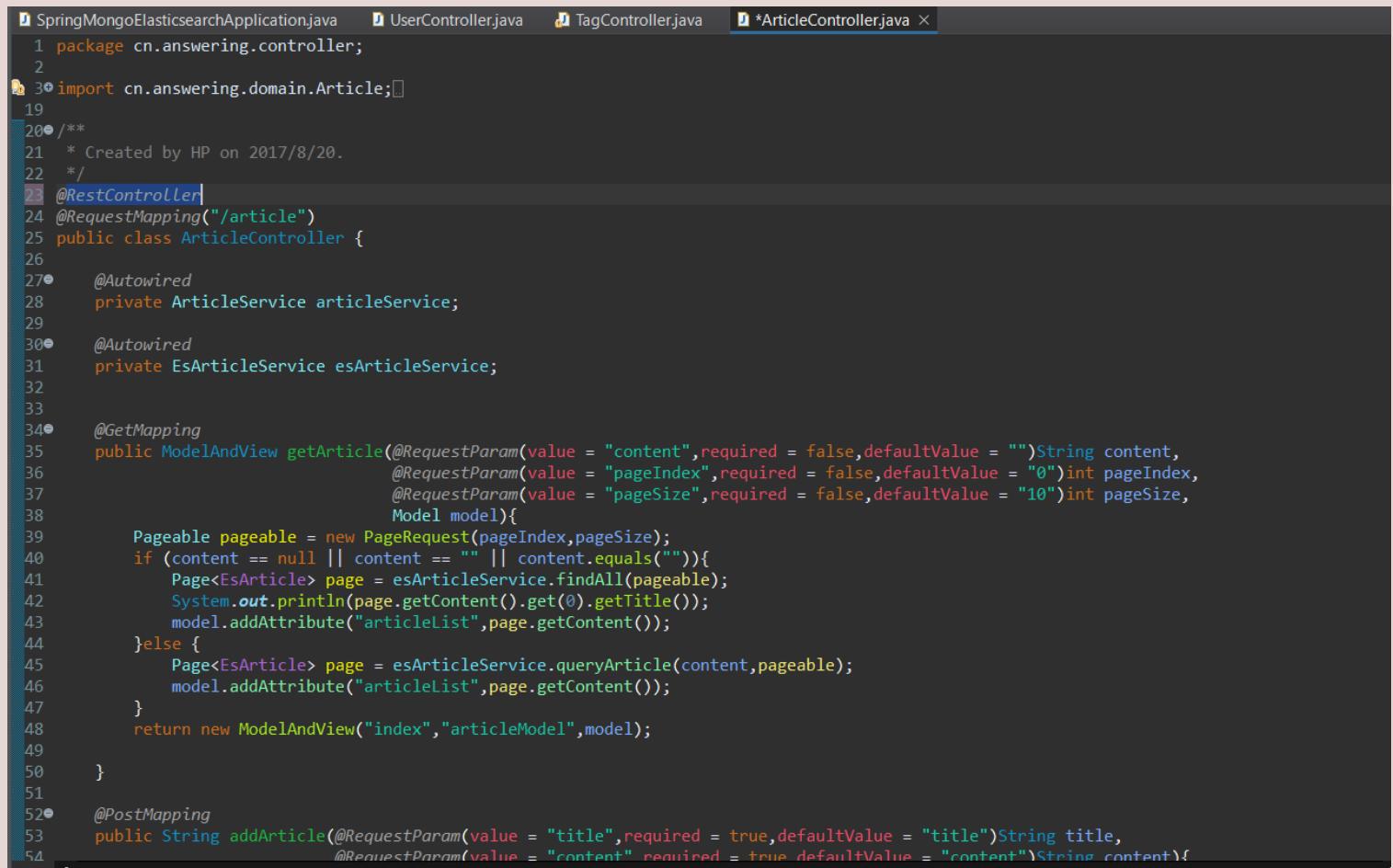


localhost:5601

L'orchestration de l'api-gateway par springboot

get (localhost:8080/user/id)	avoir l'utilisateur et les informations d'apres son id creer un utilisateur (lecture microservice)
post(localhost:8080/user)	creer un utilisateur (ecriture microservice)
get (localhost:8080/article")	avoir un article soit par son nom soit par sa description creer un utilisateur (lecture microservice)
post (localhost:8080/article")	ajouter un article (ecriture microservice)
update (localhost:8080/article")	modifier un article (ecriture microservice)
delete (localhost:8080/article")	supprimer un article (ecriture microservice)
get (localhost:8080)	aller a la page d'accueil l'interface graphique de notre api creer par thymleaf

L'orchestration de l'api-gateway par springboot est assurer par le controller est leur url de notre webservices notre controller est batit sur rest qui est rapide



The screenshot shows a Java code editor with several tabs at the top: SpringMongoElasticsearchApplication.java, UserController.java, TagController.java, and ArticleController.java (which is currently selected). The ArticleController.java code is displayed below:

```
1 package cn.answering.controller;
2
3 import cn.answering.domain.Article;
4
5 /**
6  * Created by HP on 2017/8/20.
7  */
8 @RestController
9 @RequestMapping("/article")
10 public class ArticleController {
11
12     @Autowired
13     private ArticleService articleService;
14
15     @Autowired
16     private EsArticleService esArticleService;
17
18     @GetMapping
19     public ModelAndView getArticle(@RequestParam(value = "content", required = false, defaultValue = "") String content,
20                                     @RequestParam(value = "pageIndex", required = false, defaultValue = "0") int pageIndex,
21                                     @RequestParam(value = "pageSize", required = false, defaultValue = "10") int pageSize,
22                                     Model model) {
23         Pageable pageable = new PageRequest(pageIndex, pageSize);
24         if (content == null || content == "" || content.equals("")) {
25             Page<EsArticle> page = esArticleService.findAll(pageable);
26             System.out.println(page.getContent().get(0).getTitle());
27             model.addAttribute("articleList", page.getContent());
28         } else {
29             Page<EsArticle> page = esArticleService.queryArticle(content, pageable);
30             model.addAttribute("articleList", page.getContent());
31         }
32         return new ModelAndView("index", "articleModel", model);
33     }
34
35     @PostMapping
36     public String addArticle(@RequestParam(value = "title", required = true, defaultValue = "title") String title,
37                             @RequestParam(value = "content", required = true, defaultValue = "content") String content) {
38 }
```

Le modèle de écriture de mongo db

```
@Document
public class Article {
    @Id
    private String id;
    private String title;
    private String content;
    private User author;
    private List<Tag> tags;
    private int readCount;
    private int likeIt;
    protected Article(){}
    public Article(String title, String content, User author){
        this.title = title;
        this.content = content;
        this.author = author;
    }
    public Article(String title, String content){
        this.title = title;
        this.content = content;
    }
}
```

Le modèle de lecture de elastic search

```
@Document(indexName = "article", type = "article")
public class EsArticle implements Serializable{
    private static final long serialVersionUID = -3442422877407279457L;

    @Id
    private String id;
    @Field
    private String articleId;
    private String title;
    private String content;
    private User author;
    private List<Tag> tags;
    @Field(index = FieldIndex.not_analyzed)
    private int readCount;
    @Field(index = FieldIndex.not_analyzed)
    private int likeIt;
    protected EsArticle(){}
    public EsArticle(String title, String content){
        this.content = content;
    }
}
```

Le repository de mongo db

```
1 package cn.answering.repository;
2
3 import cn.answering.domain.Article;
4 import org.springframework.data.mongodb.repository.MongoRepository;
5 import org.springframework.data.repository.CrudRepository;
6 import org.springframework.stereotype.Repository;
7
8 |
9 @Repository
10 public interface ArticleRepository extends MongoRepository<Article, String>{
11 }
12
13
```

Le repository de elastic search

```
1 package cn.answering.repository.es;
2
3 import cn.answering.domain.Tag;
4 import cn.answering.domain.User;
5 import cn.answering.domain.es.EsArticle;
6 import org.springframework.data.domain.Page;
7 import org.springframework.data.domain.Pageable;
8 import org.springframework.data.elasticsearch.repository.ElasticsearchRepository;
9 import org.springframework.data.repository.CrudRepository;
10 import org.springframework.stereotype.Repository;
11
12 import java.util.List;
13
14 @Repository
15 public interface EsArticleRepository extends ElasticsearchRepository<EsArticle, String> {
16
17     Page<EsArticle> findDistinctByTitleContainingOrContentContainingOrAuthorContainingOrTagContaining(String title, String content, String author, String tag, Pageable pageable);
18
19     Page<EsArticle> findDistinctByTitleContainingOrContentContaining(String title, String content, Pageable pageable);
20
21     void deleteEsArticleByArticleId(String id);
22
23
24 }
25
```

Les services de lecture

```
@Override  
public Page<EsArticle> queryArticle(String content, Pageable pageable) {  
    return esArticleRepository.findDistinctByTitleContainingOrContentContainin  
}  
  
@Override  
public Page<EsArticle> queryArticle(String title, String content, User author  
    return esArticleRepository.findDistinctByTitleContainingOrContentContainin  
}
```

Les services d'écriture

```
@Autowired  
private ArticleRepository articleRepository;  
  
@Autowired  
private EsArticleRepository esArticleRepository;  
  
@Transactional  
@Override  
public Article saveOrUpdateArticle(Article article) {  
    Article returnValue = articleRepository.save(article);  
    esArticleRepository.save(new EsArticle(article));  
    return returnValue;  
}  
  
@Transactional  
@Override  
public void deleteArticle(String id) {  
    articleRepository.delete(id);  
    esArticleRepository.deleteEsArticleByArticleId(id);  
}
```

L'interface de l'utilisateur créer avec thymleaf



```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head th:replace="fragments/header :: header">
    <meta charset="UTF-8"/>
    <title>Titleeeee</title>
</head>
<body>
<div class="container">
    <div th:replace="fragments/nav :: nav"></div>

    <div class="page-container">
        <div><button class="btn btn-primary" data-toggle="modal" data-target="#myModal">Add to </button></div>
        <div th:if="${articleModel}">
            <ul class="list-group" th:each="article : ${articleModel.articleList}">
                <li class="list-group-item"><span th:text="${article.title}"></span></li>
            </ul>
        </div>
    </div>
    <div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-labelledby="myModalLabel" aria-hidden="true">
        <div class="modal-dialog">
            <div class="modal-content">
                <div class="modal-header">
                    <h4 class="modal-title" id="myModalLabel">Add article </h4>
                </div>
                <div class="modal-body">
                    <form action="article" th:action="article" method="post">
                        <div>
                            <label>title :</label>
                            <input name="title" type="text"/>
                        </div>
                        <div>
                            <label>Content: </label>
                            <textarea name="content" rows="30" cols="60"/>
                        </div>
                        <div>
                            <input type="submit" value="submit"/>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>
```

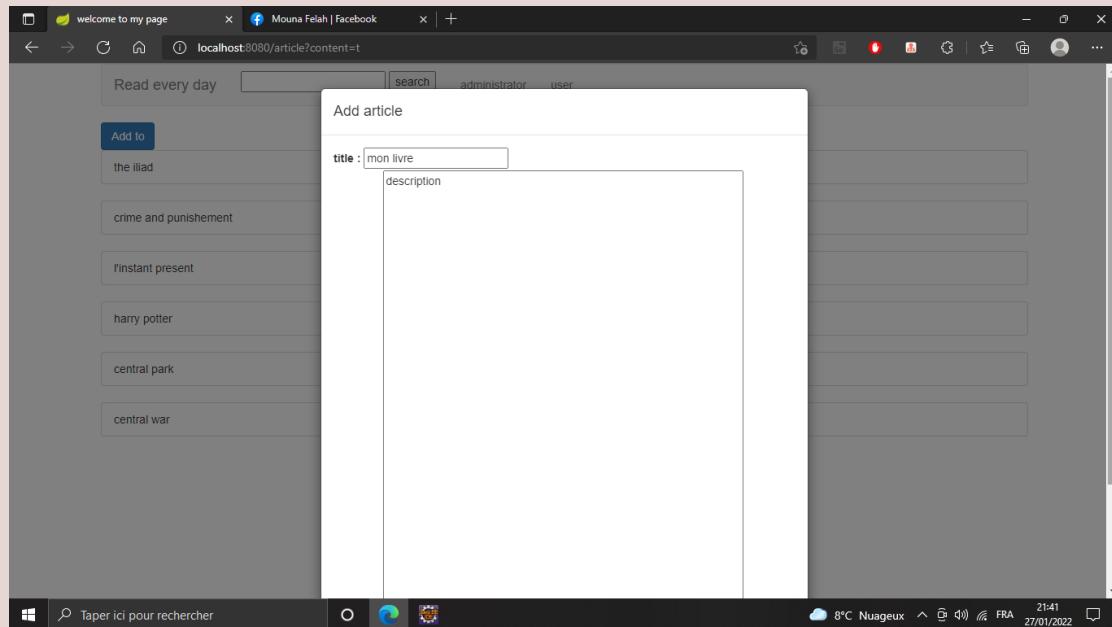
Ce qui crée l'interface

A screenshot of a web browser window. The address bar shows "localhost:8080/article?content=t". The page content includes a search bar with placeholder "Read every day" and buttons "search", "administrator", and "user". A list of books is displayed in a table:

Add to
the iliad
crime and punishment
l'instant présent
harry potter
central park
central war

Welcome to this website

A screenshot of a web browser window. The address bar shows "localhost:8080/article?content=t". The page content includes a search bar with placeholder "Read every day" and buttons "search", "administrator", and "user". A modal dialog box titled "Add article" is open, containing a "title" input field with the value "mon livre" and a "description" text area.



Pour les test on utiliserait Les Test unitaires produits par spring boot

```
package cn.answering;

import org.junit.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
public class SpringMongoElasticsearchApplicationTests {

    @Test
    public void contextLoads() {
    }
}
```