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'ecriture et de lecture des bouquins avec le framework spring boot 2.2. 1

- Le premier microservice permettrait de lire dans un catalogue

- Le deuxieme microservice permettrait d'ecrire dans un catalogue

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

elasticsearch

- Le deuxieme microservice d'ecriture utiliserait la base de donnée **mongodb**

- La synchornisation des base deux donnés se ferait avec **talend**

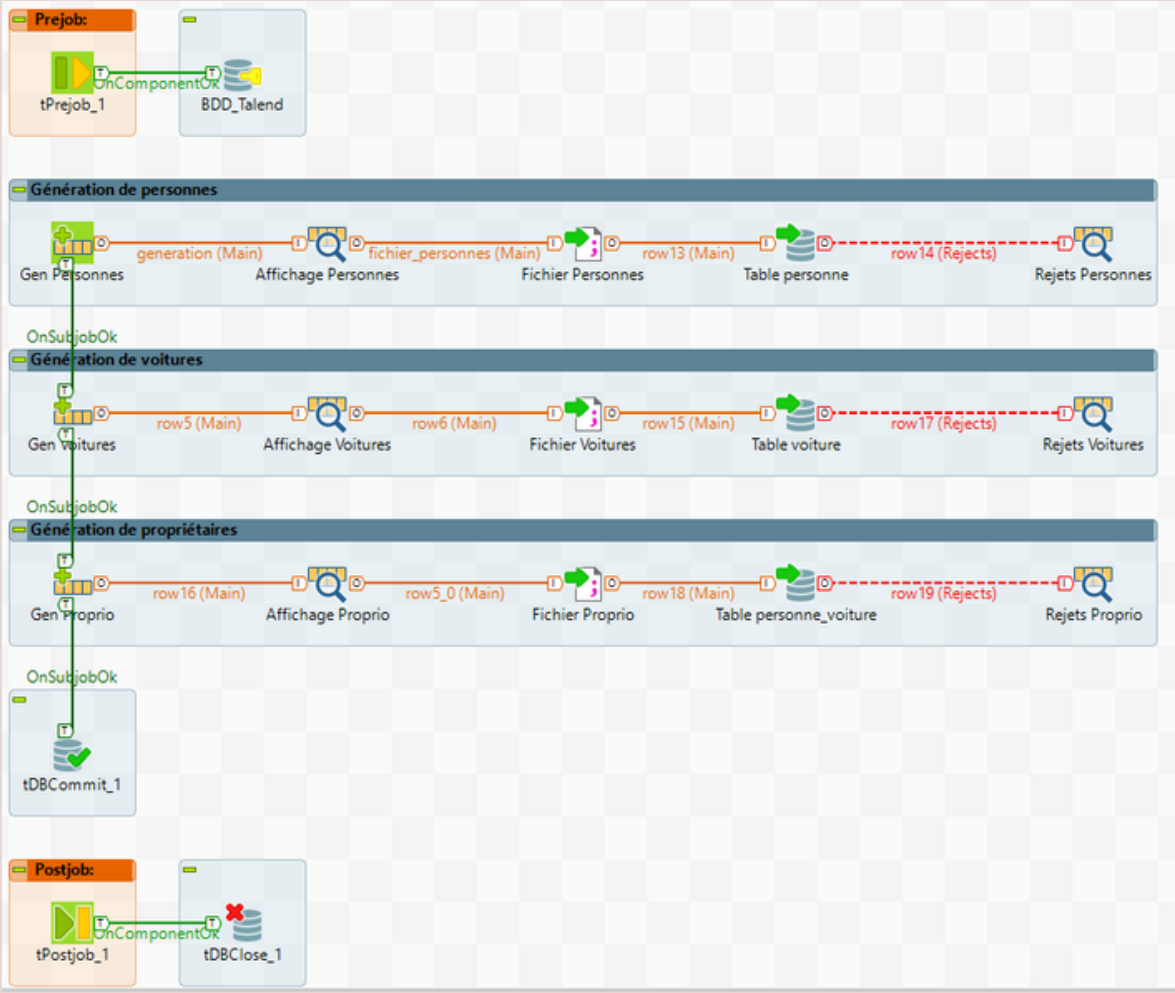
- Spring joueurait le role de l'apigateway et permetrait de orchestrait tous le flux de chargement de données

- Elasticsearch est utilisé pour le query ou bien la lecture puisque il est trés 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.

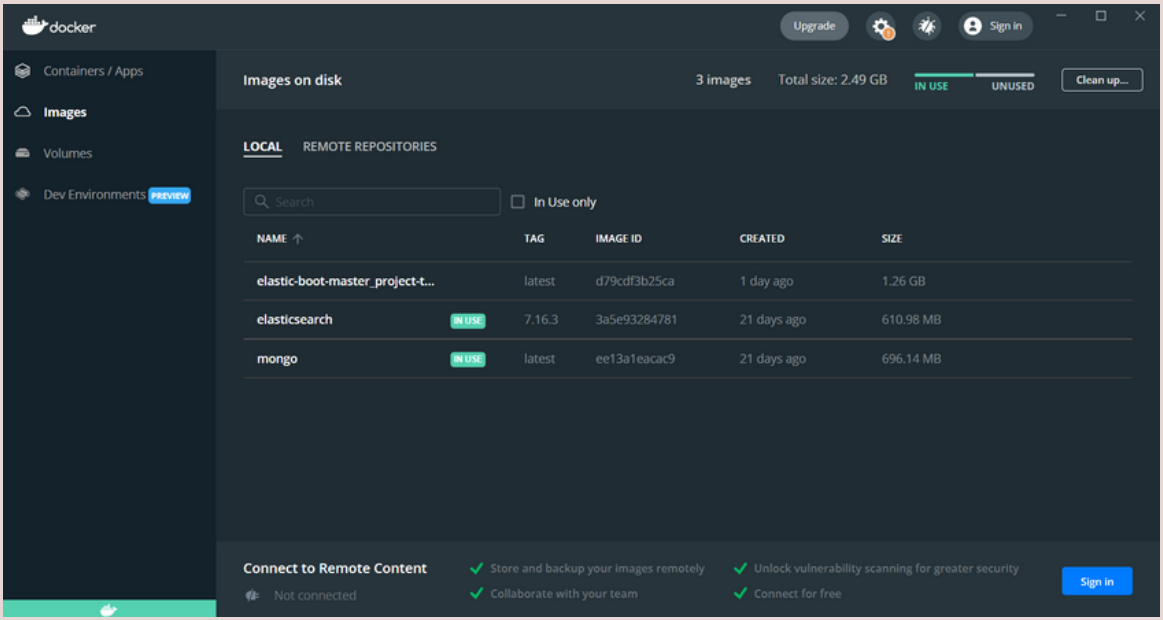
- Mongodb est utilisé pour la commande ou bien l'ecriture puisqu'il est flexible on peut garder les données en forme **JSON** contrairement au base de donnée relationnelle sous forme de table ainsi il est expressive,scalabe et facile a repliquer

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 contenaires docker qui enregistrer les images de notre solutions



- 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

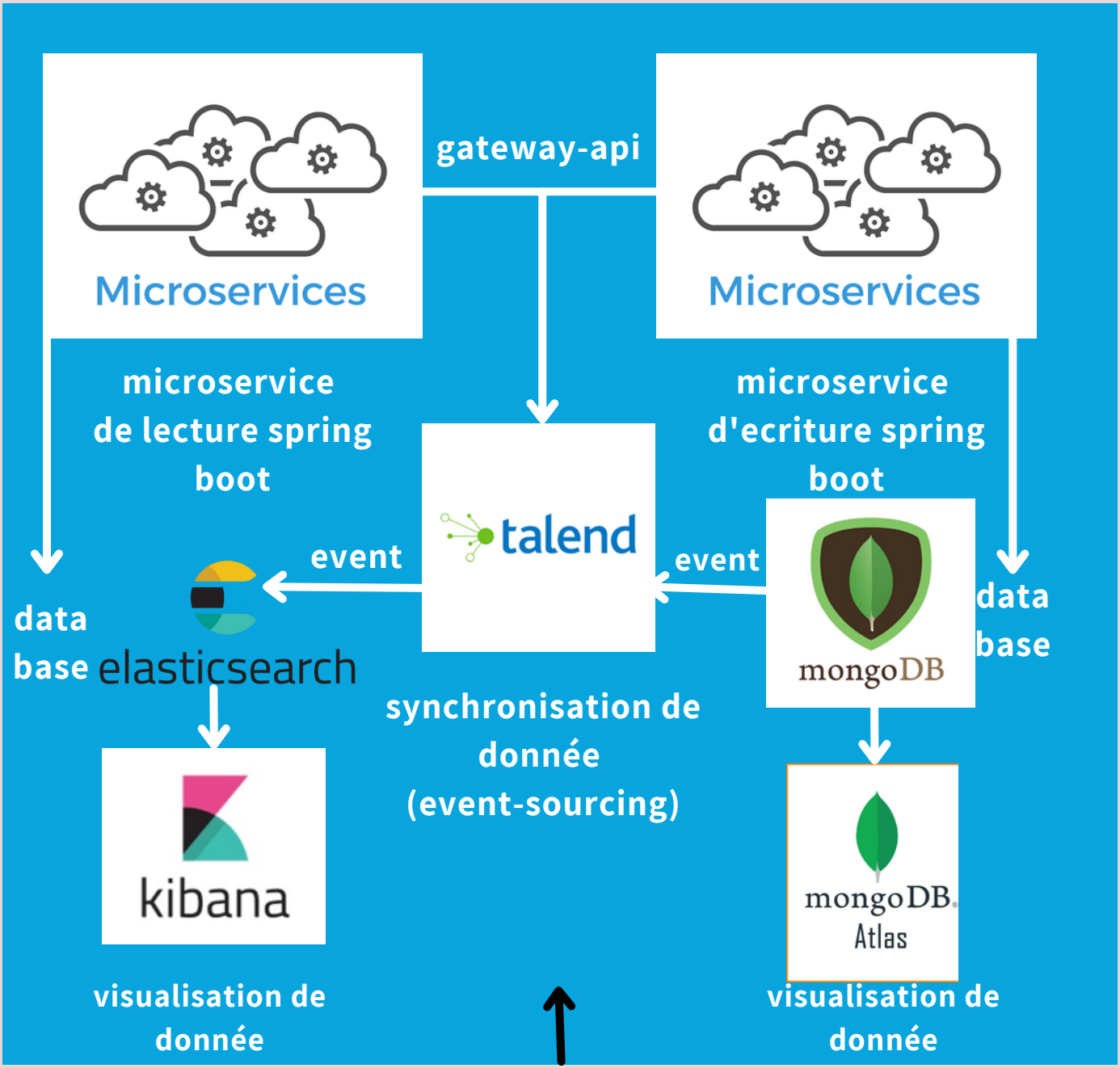
```
FROM maven:3-jdk-11

ADD . /cqrs
WORKDIR /cqrs

RUN ls -l

RUN mvn clean install -U

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-cqrs-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-cqrs-network
```

Docker-compose

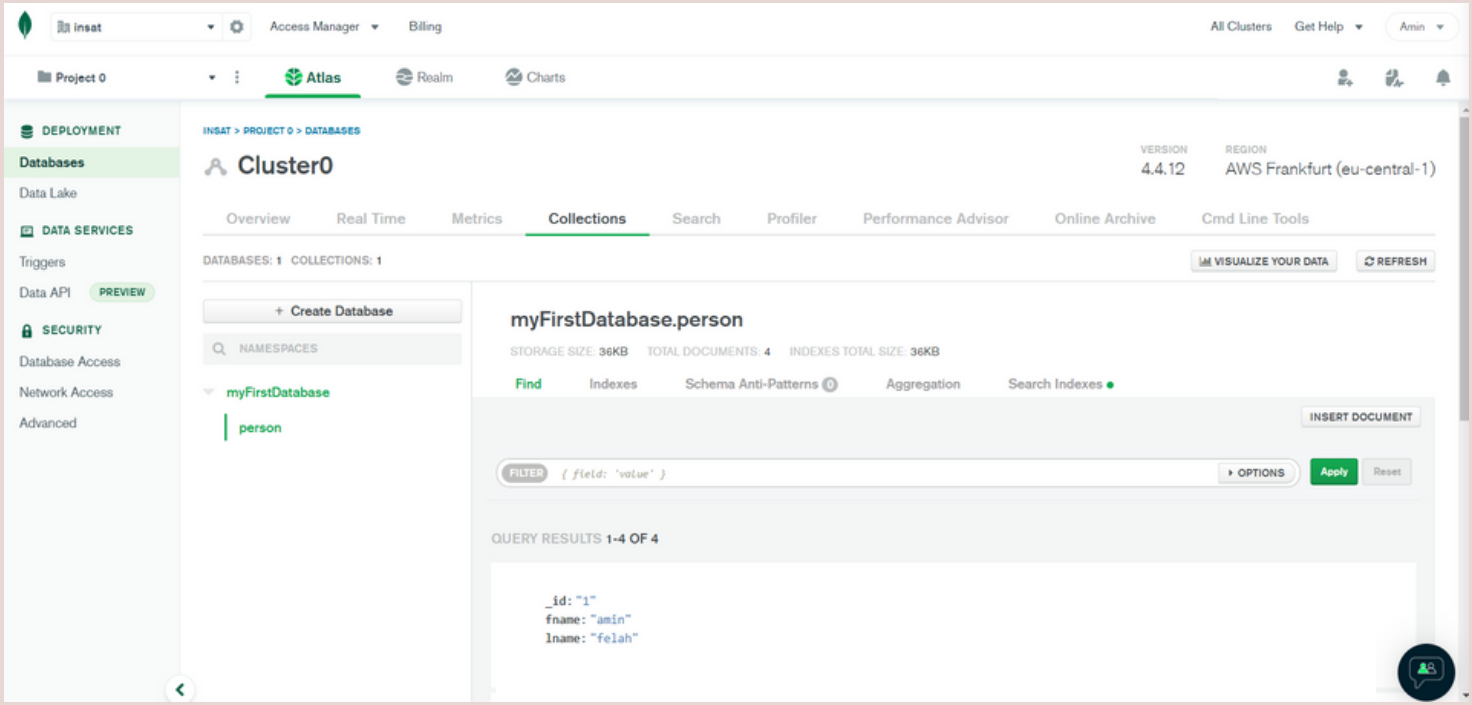
- L'architecture du project



Contenaire docker

MongoDB ATLAS

pour la visualisation de notre database mongo



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

SpringMongoElasticsearchApplication.java | UserController.java | TagController.java | *ArticleController.java

```java
package cn.answering.controller;

import cn.answering.domain.Article;

/**
 * Created by HP on 2017/8/20.
 */
@RestController
@RequestMapping("/article")
public class ArticleController {

    @Autowired
    private ArticleService articleService;

    @Autowired
    private EsArticleService esArticleService;


    @GetMapping
    public ModelAndView getArticle(@RequestParam(value = "content",required = false,defaultValue = "")String content,
                                   @RequestParam(value = "pageIndex",required = false,defaultValue = "0")int pageIndex,
                                   @RequestParam(value = "pageSize",required = false,defaultValue = "10")int pageSize,
                                   Model model){
        Pageable pageable = new PageRequest(pageIndex,pageSize);
        if (content == null || content == "" || content.equals("")){
            Page<EsArticle> page = esArticleService.findAll(pageable);
            System.out.println(page.getContent().get(0).getTitle());
            model.addAttribute("articleList",page.getContent());
        }else {
            Page<EsArticle> page = esArticleService.queryArticle(content,pageable);
            model.addAttribute("articleList",page.getContent());
        }
        return new ModelAndView("index","articleModel",model);

    }

    @PostMapping
    public String addArticle(@RequestParam(value = "title",required = true,defaultValue = "title")String title,
                             @RequestParam(value = "content",required = true,defaultValue = "content")String content){
```

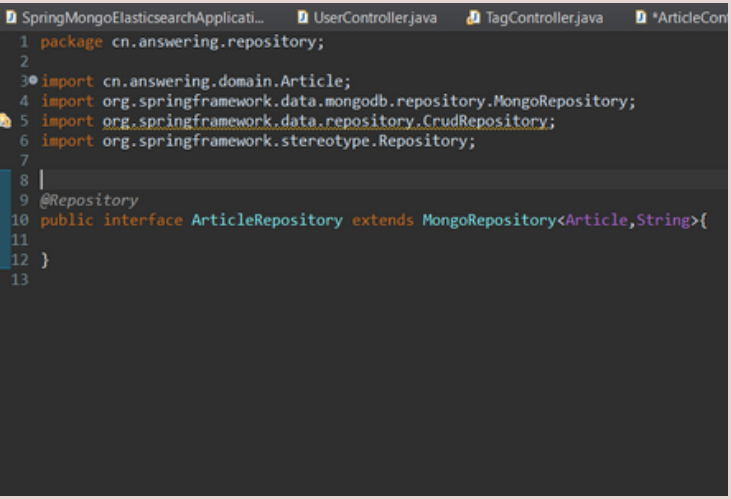
Le modele de ecriture de mongo db

```java
@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;
    }
```

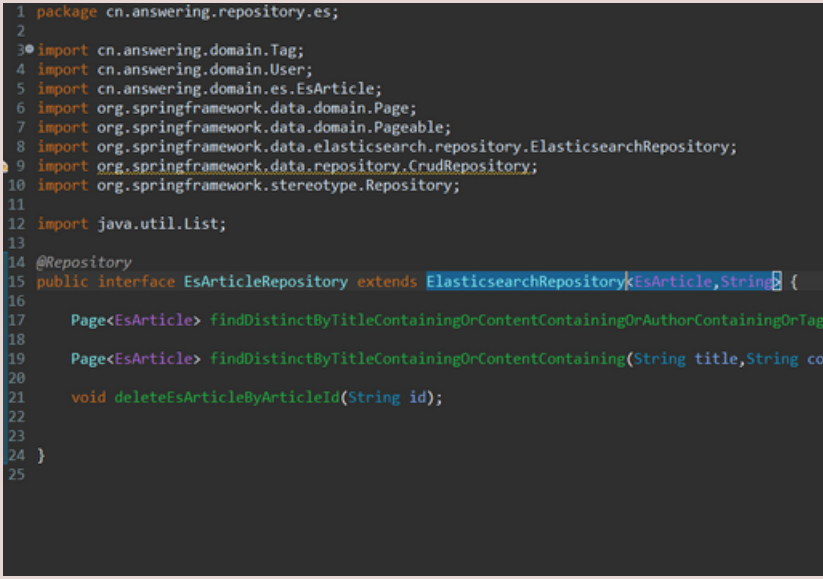
Le modele de lecture de elastic search

```java
@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



Le repository de elastic search



Les services de lecture

```java
@Override
public Page<EsArticle> queryArticle(String content, Pageable pageable) {
    return esArticleRepository.findDistinctByTitleContainingOrContentContaini
}

@Override
public Page<EsArticle> queryArticle(String title, String content, User author
    return esArticleRepository.findDistinctByTitleContainingOrContentContaini
}
```

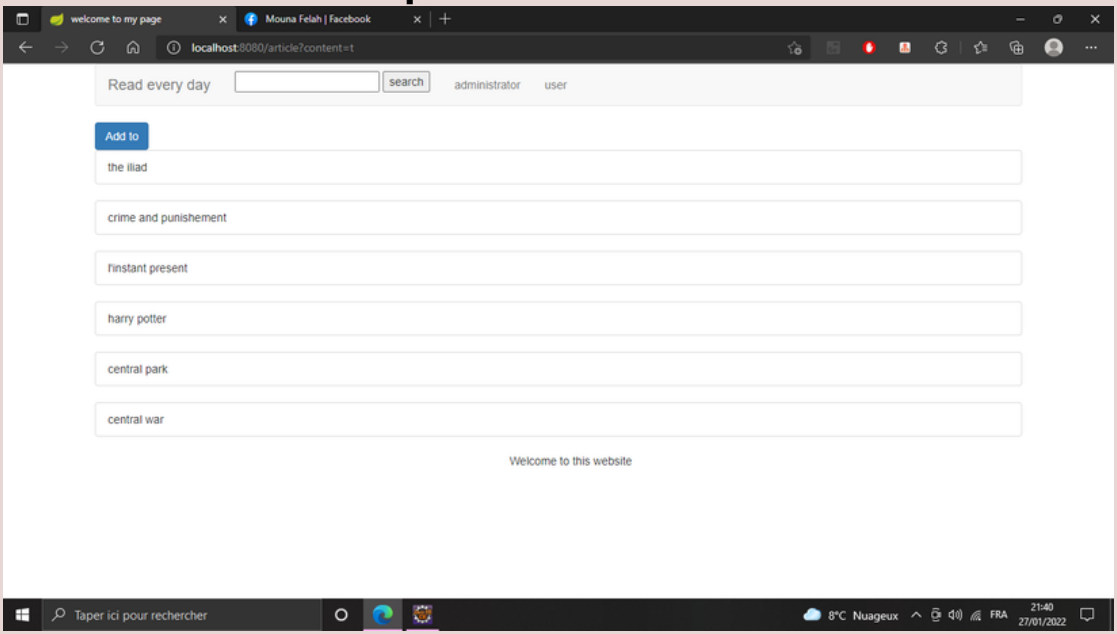
Les services d'ecriture

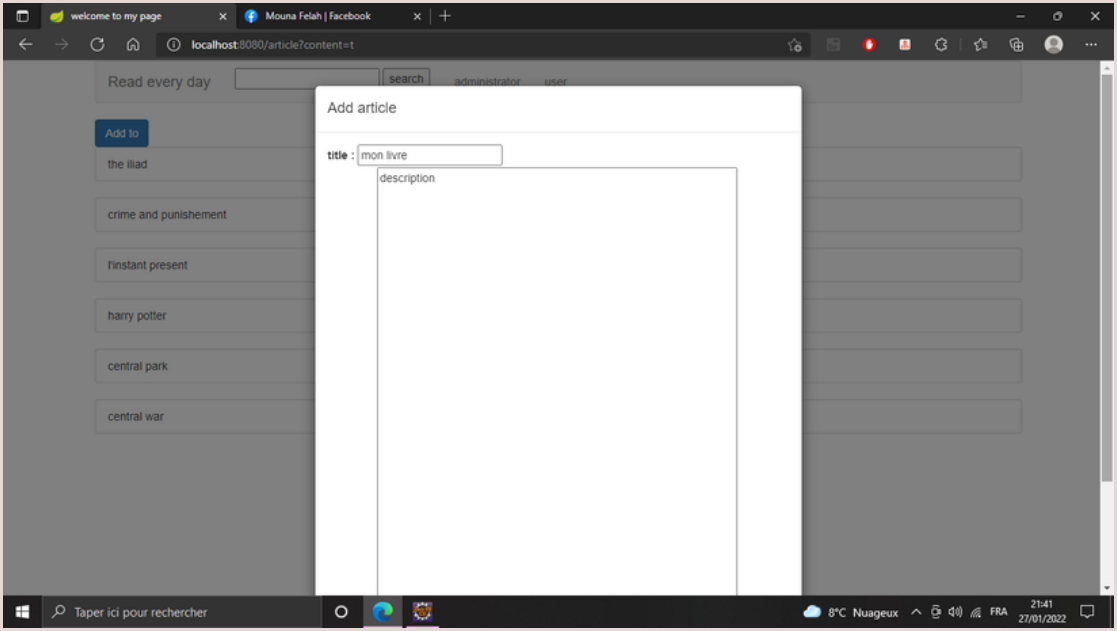
```java
@Autowired
private ArticleRepository articleRepository;

@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);
}
```

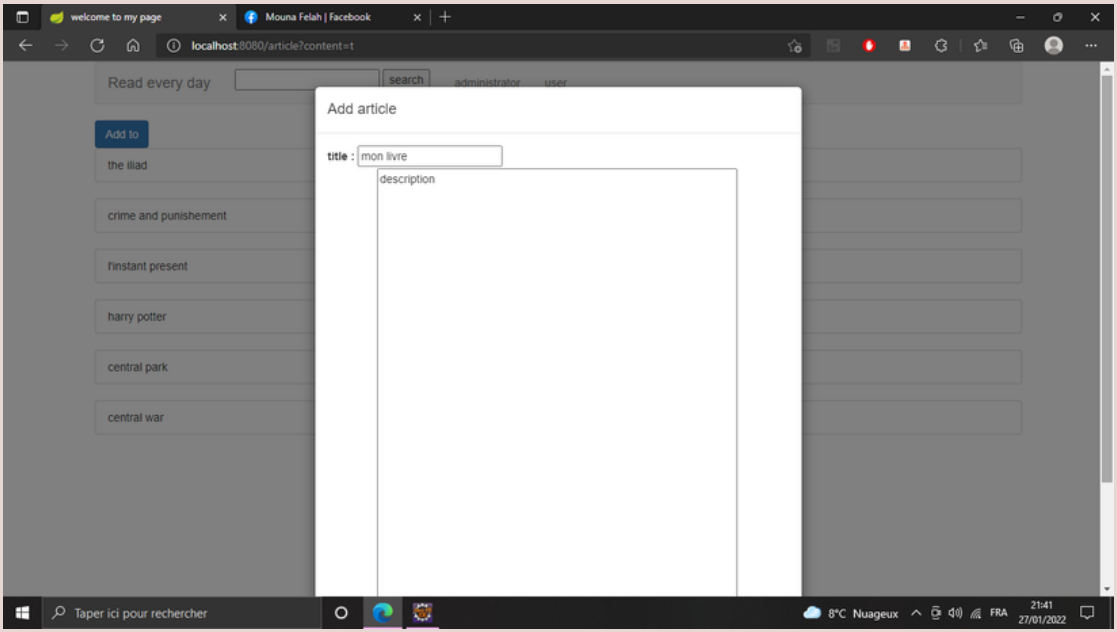
L'interface de l'utilisateur creer avec thymleaf


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

Ce qui crée l'interface







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

```java
package cn.answering;

import org.junit.Test;

@RunWith(SpringRunner.class)
@SpringBootTest
public class SpringMongoElasticsearchApplicationTests {

    @Test
    public void contextLoads() {
    }

}
```