JEE

RAPPORT DE PROJET: MODULE ETUDIANT

Spring Boot

Réalisé par : Ilyas El Amrani & Mohamed El Jaouhari
Encadré par : Mme. Ibtissam Bakkouri

Sommaire

A- INTRODUCTION & Présentation d'idée de projet

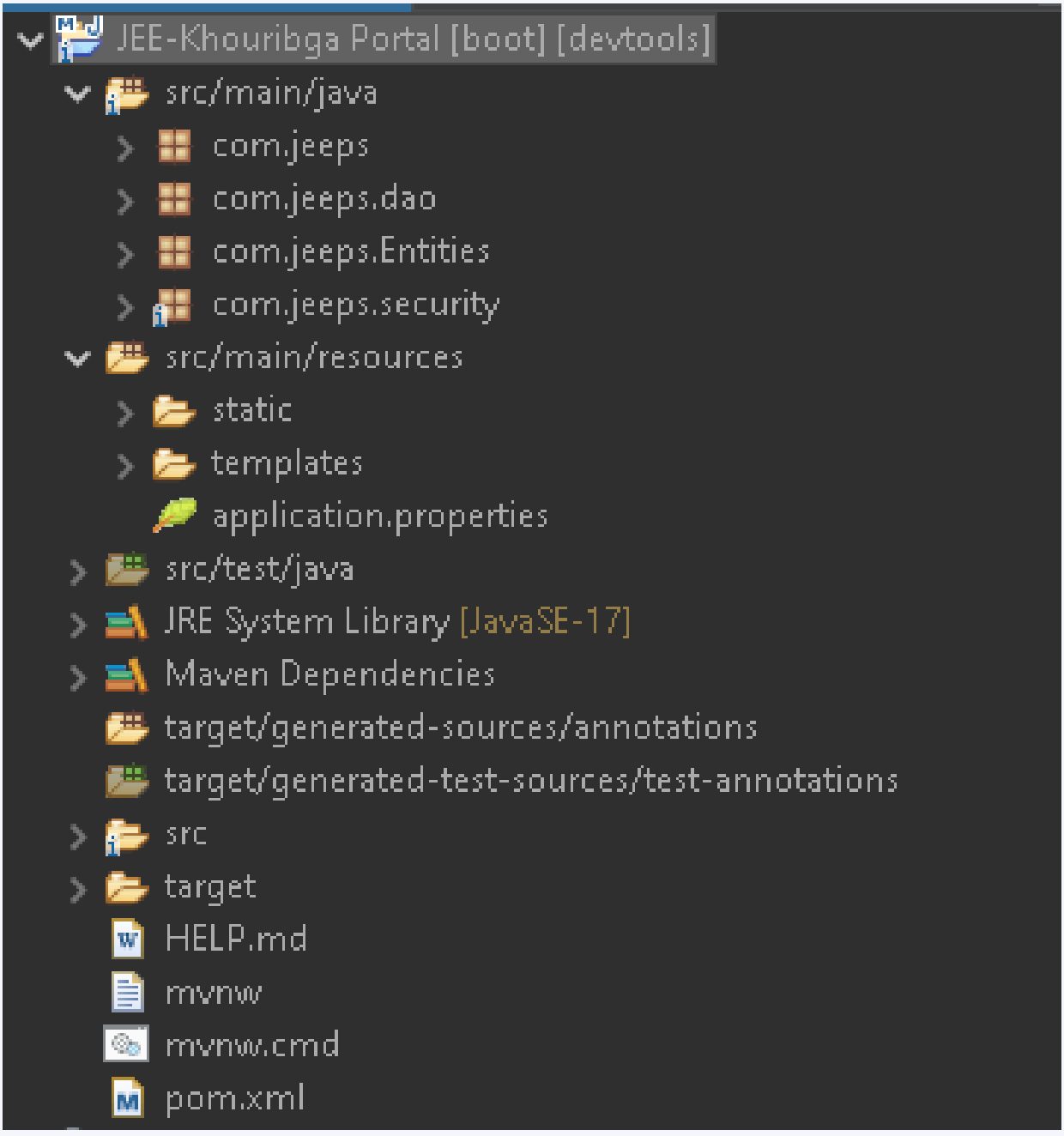
Dans ce projet, nous allons concevoir et développer une application web à l'aide de la plateform JEE, notre application sera capable de présenter un site web informatif aux utilisateurs public ou inscits sur les études à la ville de Khouribga à l'aide de présentation des informations des écoles de différents niveaux, formations et spécialités, ainsi que permettre aux utilisateurs de consulter des articles relatifs à la vie étudiants à la ville de Khouribga. D'avantage, notre application sera capable de permettre aux utilisateurs inscrits d'écrire et publier des articles sur le site avec toute les fonctionnalités réquises, ainsi qu'un espace d'administration capable de gérer la totalité du site web, tout en maintenant la sécurité et le style du site web, sa dynamicité et intéractivité grace aux commentaires, et son légitimalité grace aux conditions légales et controle et validation du contenu.

Pour réalisé cela, nous allons appliquées nos connaissances et expériences en développement WEB, en JEE et de programmation orientée objet, en se basant sur les technologies suivantes :

- Spring Boot 3.0.0
- JAVA 17
- Base de données MySQL
- Spring Security 6.0
- Spring WEB
- Serveur d'application TOMCAT par défaut
- HTML5 , CSS & Javascript
- Thymeleaf

A- Introduction & Présentation d'idée de projet

Pour le faire, nous allons utiliser l'environement de développement d'Eclipse, gestion de dépendance avec Maven et Spring Tools Suite 4, à l'aide de Spring Initializr, nous créons notre projet en sélectionnant Maven, JAVA 17 pour la configuration, et comme dépandances Spring WEB, Spring JPA, Spring Security, MySQL J Connector et Dev Tools.
Notre projet aura la structure suivante en terme de packages :



B- Modèle des données et entités:

Comme mentionné dans l'introduction, notre application doit être capable de supporter l'accès aux données, y compris leur l'ajout, modification, suppression et affichage, les entités à manipuler sont donc:

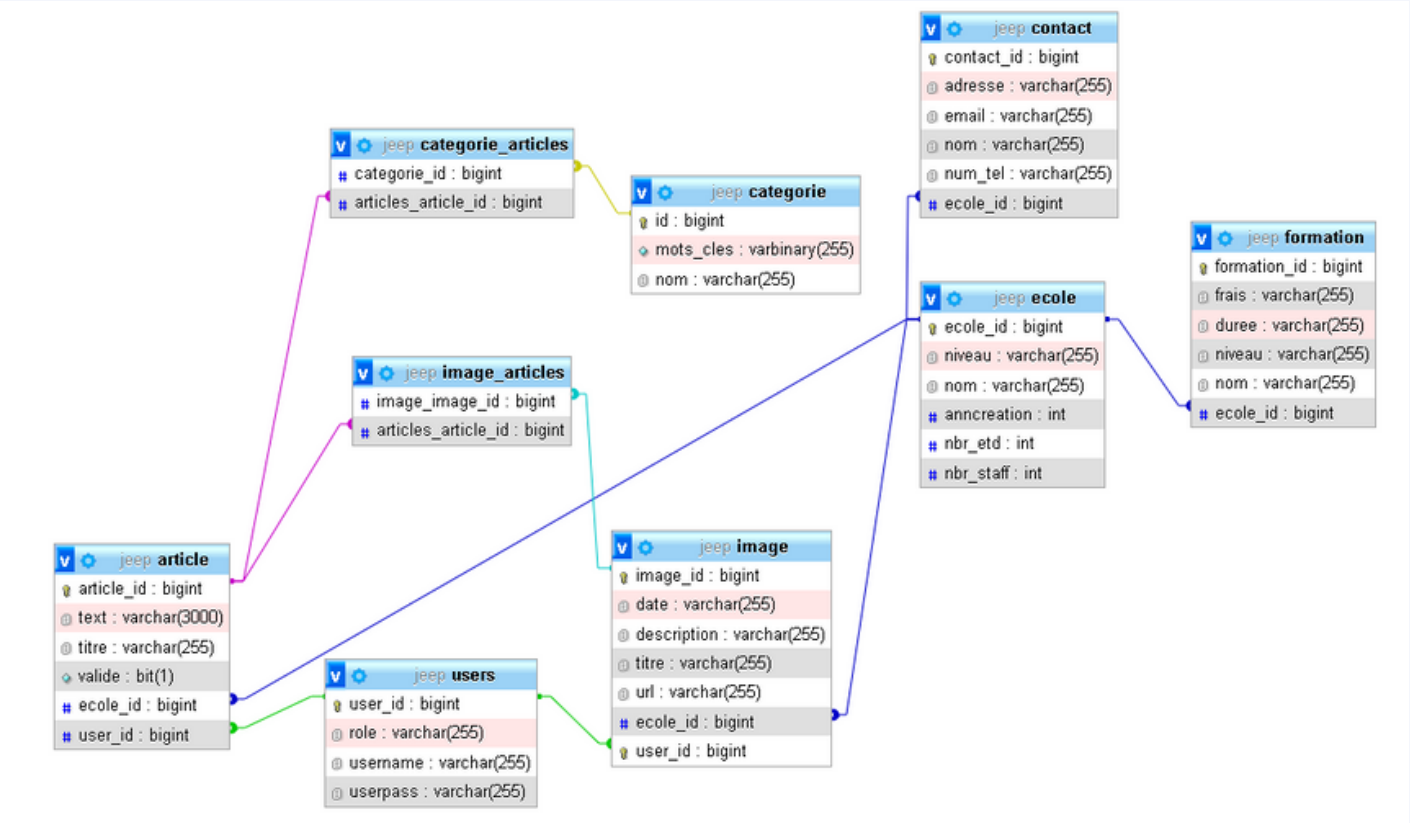
- École : Comprend des information sur chaque école.
- User : Comprend les informations de l'utilisateur.
- Formation : Comprend les information d'une formation d'une école
- Contact : Comprend les information d'un contact (numéro de téléphone, email ..)
- Image : Représente une image par son titre, description et URL
- Article : Permet de représenter un article avec son titre, text, mots-clés(utiles pour la recherche) et validité et le lie à un utilisateur (auteur), une école ou une ou plusieurs categories.
- Catégorie : permet de regrouper l'ensemble des données sur une catégorie d'articles.

D'ailleurs, nous allons utiliser un bean "status" pour permettre de représenter le status de l'opération demandé sur les données.
L'ensemble des données seront au niveau du package intern "com.jeeps.Entities".
Nos données seront stocké sur une base de données MySQL, donc au niveau des propriétés de l'application "Application properties", nous allons spécifier l'accès aux données avec les déclarations suivantes:

B- Modèle des données et entités:

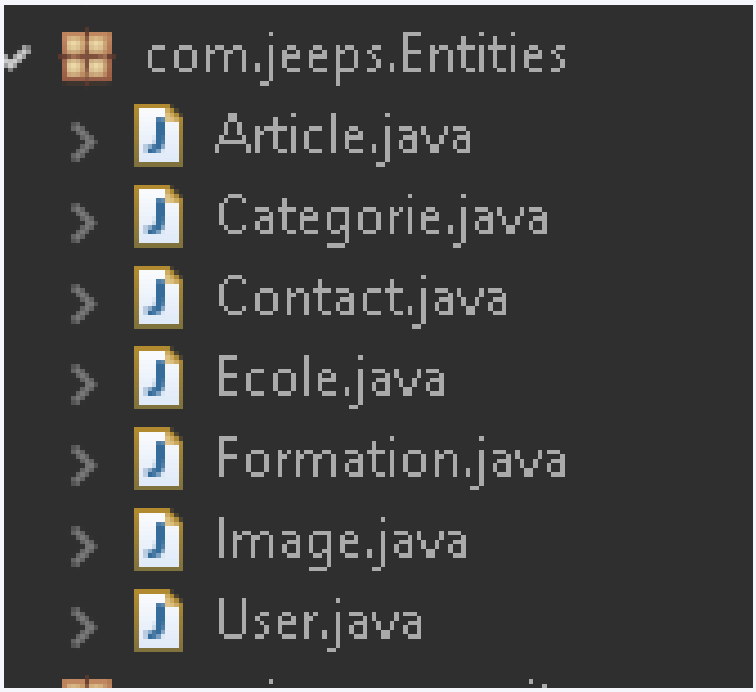
```
spring.datasource.url=jdbc:mysql://localhost:3306/JEEP?useSSL=false
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.jpa.hibernate.ddl-auto = update
```

Cela permet de spécifier l'url de notre BD et les données d'authentification, ainsi que le driver à utiliser et la méthode d'accès update, nous détaillons le code des entités ci-dessous, notre diagramme de données est le suivant:



Nos données sont représantées par des entités à l'aide de l'API JPA de Jakarta, la nouvelle version de Java, utiliser par Spring JPA, nous allons regrouper toutes les entités dans un package "com.jeeps.entities" comme suit :

B- Modèle des données et entités:



Nous représenterons chaque entité en suite, en capturant juste ces attributs, le reste est toujours les getters/setters et les constructeurs.

L'entité Article :

```java
package com.jeeps.Entities;

import java.io.Serializable;

@SuppressWarnings("serial")
@Entity
@Table(name="Article")
public class Article implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private long article_id;
    private String Titre;
    @Column(length = 3000)
    private String Text;
    private boolean valide;

    @ManyToOne
    @JoinColumn(name="user_id")
    private User user;

    @ManyToOne
    @JoinColumn(name="ecole_id")
    private Ecole ecole;

    @ManyToMany(mappedBy="articles")
    private List<Image> image;

    @ManyToMany(mappedBy="articles")
    private List<Categorie> categorie;
```

B- Modèle des données et entités:

L'entité Categorie :

```java
package com.jeeps.Entities;

import java.io.Serializable;

@SuppressWarnings("serial")
@Entity
@Table(name="Categorie")
public class Categorie implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private long id;
    private String nom;

    @ManyToMany
    private List<Article> articles;
```

L'entité Contact :

```java
package com.jeeps.Entities;

import java.io.Serializable;

@SuppressWarnings("serial")
@Entity
@Table(name="Contact")
public class Contact implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private long contact_id;
    private String nom;
    private String numTel;
    private String adresse;
    private String email;

    @ManyToOne
    @JoinColumn(name="ecole_id")
    private Ecole ecole;
```

B- Modèle des données et entités:

L'entité École :

```java
@SuppressWarnings("serial")
@Entity
@Table(name="Ecole")
public class Ecole implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private long ecole_id;
    private String Nom;
    private String Niveau;
    private int anncreation;
    private int nbrEtd;
    private int nbrStaff;

    @OneToMany(mappedBy="ecole")
    private List<Image> images;

    @OneToMany(mappedBy="ecole")
    private List<Article> articles;

    @OneToMany(mappedBy="ecole")
    private List<Formation> formations;

    @OneToMany(mappedBy="ecole")
    private List<Contact> contacts;
```

L'entité Formation :

```java
@SuppressWarnings("serial")
@Entity
@Table(name="Formation")
public class Formation implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private long formation_id;
    private String nom;
    private String duree;
    private String niveau;
    private String Frais;

    @ManyToOne
    @JoinColumn(name="ecole_id")
    private Ecole ecole;
```

B- Modèle des données et entités:

L'entité Image :

```java
@SuppressWarnings("serial")
@Entity
@Table(name="Image")
public class Image implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private long image_id;
    private String Titre;
    private String Description;
    private String URL;
    private String Date ;

    @ManyToMany
    private List<Article> articles;

    @OneToOne
    @JoinColumn(name="user_id")
    private User user;

    @ManyToOne
    @JoinColumn(name="ecole_id")
    private Ecole ecole;

    @OneToOne(mappedBy="profileimg")
    private User profile;
```

L'entité User :

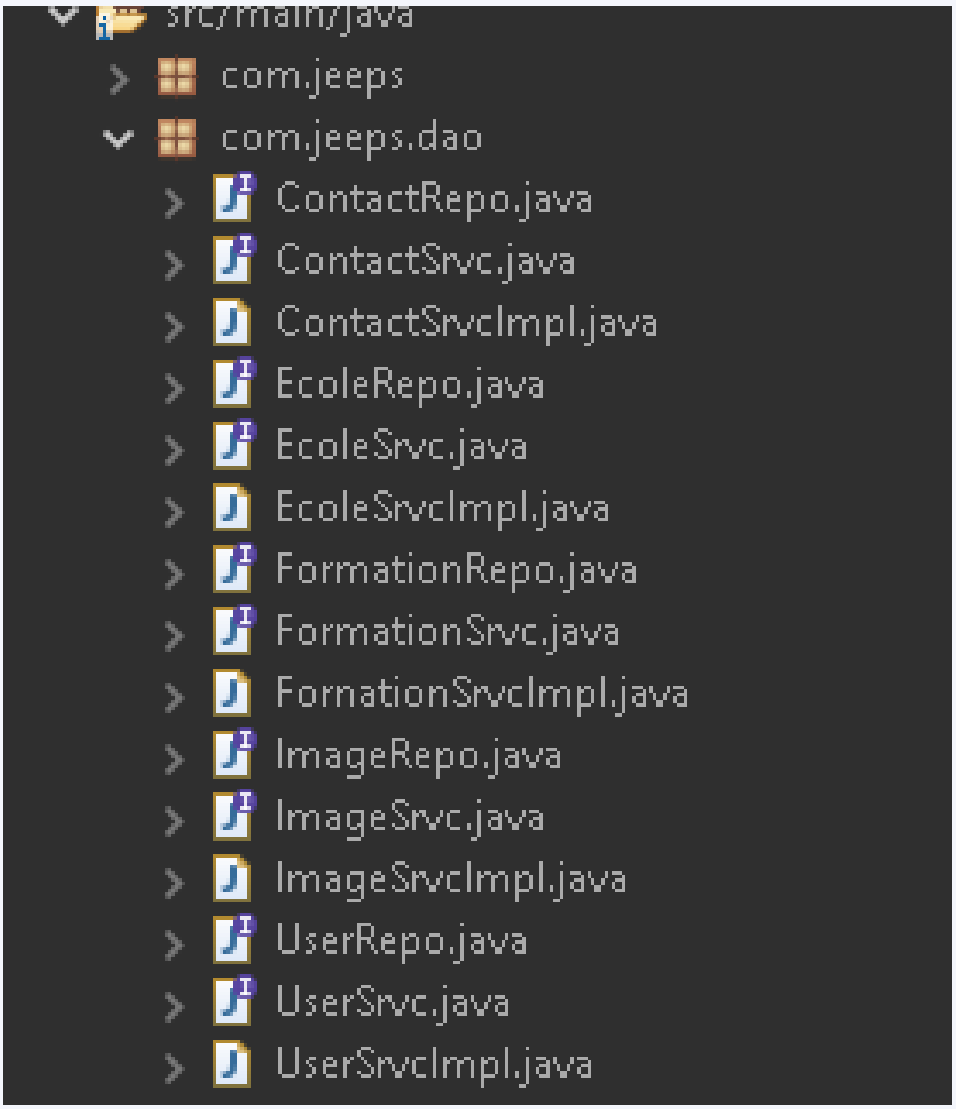
```java
@SuppressWarnings("serial")
@Entity
@Table(name="Users")
public class User implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private long user_id;
    private String username;
    private String userpass;
    private String role;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name="user_id", referencedColumnName="image_id")
    private Image profileimg;

    @OneToMany(mappedBy="user",cascade = CascadeType.ALL)
    private List<Article> articles;
```

B- Modèle des données et entités:

Pour pouvoir manipuler les données, pour chaque entité, nous avons besoin de déclarer une intérface qui hérite de l'interface JpaRepository en spécifiant la classe entité à manipuler et la classe de la clé primaire de cette classe, par la suite il faut créer une classe de services de manipulations (création, ajout , suppression, recherche ...) qui doit implémenter une interface décrivant ces méthodes, Nous notons l'interface héritant de JpaRepository pour l'entité XXX comme XXXRepo, l'interface qui décrit les méthodes par XXXSrvcs et la classe qui l'implémente par XXXSrvcImpl.
Toutes ces classes et intérfaces seront créées pour la totalité des entités manipulables dans le sous-package "com.jeeps.dao", qui a la structure suivante :



B- Modèle des données et entités:

Prenons l'exemple de l'entité École :

EcoleRepo :

```java
package com.jeeps.dao;

import org.springframework.data.jpa.repository.JpaRepository;


public interface EcoleRepo extends JpaRepository<Ecole,Long>{
}
```

EcoleSrvcs :

```java
package com.jeeps.dao;

import com.jeeps.Entities.Ecole;

public interface EcoleSrvc {
    public boolean ajouterEcole(Ecole e);
    public boolean supprimerEcole(long id);
    public Ecole chercherEcole(long id);
    boolean modifierEcole(long id, Ecole e);
}
```

EcoleSrvcsImpl : Vue générale :

```java
@Service
@Transactional
public class EcoleSrvcImpl implements EcoleSrvc{
    @Autowired
    private EcoleRepo ER;


    public boolean ajouterEcole(Ecole e) {

    public boolean supprimerEcole(long id) {

    public Ecole chercherEcole(long id) {

    public boolean modifierEcole(long id ,Ecole e) {


    public List<Ecole> toutEcole(){

    public List<Ecole> toutEcoleNom(String nom){

    public List<Ecole> toutEcoleNomNiveau(String nom,String niveau){

}
```

B- Modèle des données et entités:

La fonction ajouterEcole :

```
@Override
public boolean ajouterEcole(Ecole e) {
    Ecole result = ER.save(e);
    if(result.equals(e)){
        return true;
    }
    return false;
}
```

La fonction supprimerEcole :

```
@Override
public boolean supprimerEcole(long id) {
    try{
        if(ER.findById(id).get().getClass()==Ecole.class) {
            ER.deleteById(id);
            return true;
        }
        return false;
    }
    catch(Exception e){
        return false;
    }
}
```

La fonction modifierEcole :

```
@Override
public boolean modifierEcole(long id ,Ecole e) {
    Ecole dbE = chercherEcole(id);
    if(dbE!=null) {
        if(Objects.nonNull(e.getNiveau()) && !"".equalsIgnoreCase(e.getNiveau())){
            dbE.setNiveau(e.getNiveau());
        }
        if(Objects.nonNull(e.getNom()) && !"".equalsIgnoreCase(e.getNom())){
            dbE.setNom(e.getNom());
        }
        if(Objects.nonNull(e.getAnncreation())) {
            dbE.setAnncreation(e.getAnncreation());
        }
        if(Objects.nonNull(e.getNbrEtd())) {
            dbE.setNbrEtd(e.getNbrEtd());
        }
        if(Objects.nonNull(e.getNbrStaff())) {
            dbE.setNbrStaff(e.getNbrStaff());
        }
        ER.save(dbE);
        return true;
    }
    return false;
}
```

B- Modèle des données et entités:

La fonction toutEcole :

```
public List<Ecole> toutEcole(){
    return ER.findAll();
}
```

Le reste des méthodes sont similaires, en générale elle servent de l'instance de EcoleRepo pour exécuter les actions sur la b ase de données selon des conditions spécifiques, ces fonctions seront appelées par les différentes méthodes des contrôlleurs.

C- Configurations & Sécurité :

Pour bien géré l'accès à notre site web, nous avons implémenter les interfaces de Spring Security 6.0, cette dernière version qui viens de ce lancer a changer beaucoup de méthodes et de classes du framework, mais en générale, les concepts sont les mêmes, pour implémenter la sécurité de notre application nous devons tout d'abord définir deux classes qui implémentes respectivement les interfaces UserDetailsService et UserDetails, ceux sont, respectivement la classe UserDetailsSrvc et myUserDetails.

La classe UserDetailsSrvc: C'est la classe qui offre une méthode permettant de charger un utilisateur de son username, pour pouvoir par la suite l'autorisé et l'authentifier

```
@Component
public class UserDetailsSrvc implements UserDetailsService{
    @Autowired
    private UserRepo UR;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Optional<User> user = UR.findByUsername(username);
        return user.map(myUserDetails::new)
                .orElseThrow(() -> new UsernameNotFoundException("User not found " + username));
    }
}
```

C- Configurations & Sécurité :

La classe myUserDetails : permet de représenter un utilisateur suivant les spécifications réquises par la framework Spring Security, en exposant des méthode bien précises à implémenter lors de l'implémentation de l'interface UserDetails.

```java
@SuppressWarnings("serial")
public class myUserDetails implements UserDetails{
    private String username;
    private String userpass;
    private List<GrantedAuthority> authorities;

    public myUserDetails(User user) {
        username=user.getUsername();
        userpass=user.getUserpass();
        authorities = new ArrayList<GrantedAuthority>();
        SimpleGrantedAuthority auth = new SimpleGrantedAuthority(user.getRole());
        authorities.add(auth);
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return authorities;
    }

    @Override
    public String getPassword() {
        return userpass;
    }

    @Override
    public String getUsername() {
        return username;
    }

    public boolean isAccountNonExpired() {

    public boolean isAccountNonLocked() {

    public boolean isCredentialsNonExpired() {

    public boolean isEnabled() {
}
```

Les quatres derniers méthodes retournent tous simplement "true", comme tous les utilisateurs du site sont activées, n'expirent pas et ne se block pas.

C- Configurations & Sécurité :

Pour finaliser la configuration de la sécurité de l'application, il faut ajouter une dernière classe de configuration qui comprend quatres Beans: c'est la classe securityConfig, elle est de la forme générale suivante :

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class securityConfig {
    UserDetailsService userDetailsService() {

    PasswordEncoder passwordEncoder() {

    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

    AuthenticationProvider authenticationProvider(){
}
```

La méthode userDetailsService retourne tous simplement une instance de la classe UserDetailsSrvc qui implémente UserDetailsService, de même la méthode passwordEncoder retourne une instance du la classe BCryptPasswordEncoder qui perment de coder les mots de passe lors de leur récupération pour les comparés avec les mot de passé cryptés enregistrés sur la base de données dans la table "User":

```
public class securityConfig {
    @Bean
    UserDetailsService userDetailsService() {
        return new UserDetailsSrvc();
    }

    @Bean
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
```

C- Configurations & Sécurité :

La méthode securityFilterChain permet de définir les droit d'accés , l'autorisation, aux différents end points de notre application, nous voulons laisser accès à tous les utilisateurs aux points d'accès "/","/js/**/" etc..., pour le reste des points d'accès le visiteur doit s'authentifier, sauf pour le login, et l'inscription, qui require qu'il soit non authentifier "anonyme".
On spécifie aussi le point d'accès à la page de connexion "/login", s'il y a succès les utilisateurs passent au point d'accès "/" qui réfere à l'accueil.

```
@Bean
SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http.authorizeHttpRequests((requests) ->
    requests.requestMatchers("/","/public/**","/css/**","/js/**","/fonts/**"
            ,"/vendor/**","/webfonts/**","/image/**").permitAll()
            )
    .authorizeHttpRequests((requests) ->
    requests.requestMatchers("/login/**","/signup/**").anonymous()
            )
        .authorizeHttpRequests((requests) ->
            requests.anyRequest().authenticated()
        )
        .formLogin((form) -> form
                .loginPage("/login")
                .defaultSuccessUrl("/")
                .permitAll()
            );
    return http.build();
}
```

La méthode authentificationProvider regroupe le tout :

```
@Bean
AuthenticationProvider authenticationProvider(){
    DaoAuthenticationProvider authenticationProvider=new DaoAuthenticationProvider();
    authenticationProvider.setUserDetailsService(userDetailsService());
    authenticationProvider.setPasswordEncoder(passwordEncoder());
    return authenticationProvider;
}
```

C- Configurations & Sécurité :

Le fichier application properties, qui spécifie la configuration de l'accès à la base de données, le port et la configuration de Thymeleaf pour récupérer les pages HTML , il est de la forme suivante :

```
spring.datasource.url=jdbc:mysql://localhost:3306/JEEP?useSSL=false
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.jpa.hibernate.ddl-auto = update
server.port=9070
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
```

Pour désactiver l'exposition des données au publique, qui est par défaut activée , on utilise la classe de configuration suivante:

```
package com.jeeps;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SpringRestConfiguration implements RepositoryRestConfigurer {
    @Override
    public void configureRepositoryRestConfiguration(RepositoryRestConfiguration config, CorsRegistry cors) {
        config.disableDefaultExposure();
    }
}
```

Elle désactive le point d'accès aux données de type API REST par défaut.
On configure d'autre spécificités dans les classes concernées par le contrôle par la suite.

D- Contrôleurs principaux:

D - x) Controleurs des articles :

Ce controleur nous permet de gérer l'ensemble des fonctionnalité qu'une personne soit public/User/Admin peut effectuer comme opération sur les articles de notre application. on a 2 type d'article : école et catégorie. les 2 sont similaires dans certaines méthode et commune dans d'autres. **On commence par les parties similaires.**

1) getALLArticles :

Cette méthode nous permet d'afficher l'ensemble des articles liées à une école dans l'id de l'école ainsi que le type d'utilisateur sont mentionnées dans l'URL comme suivant :

```
@GetMapping("/{entity}/article/{id_school}")
public String getALLArticles(@PathVariable long id_school, @PathVariable String entity , Model model  , HttpSession session) {

    if ((String) session.getAttribute("cle")!=null && (List<Article>) session.getAttribute("articles")!=null &
            (Ecole) session.getAttribute("ecole")!=null) {

        String cle = (String) session.getAttribute("cle");
        session.removeAttribute("cle");
        List<Article> articles = (List<Article>) session.getAttribute("articles");
        Ecole ecole = (Ecole) session.getAttribute("ecole");
        model.addAttribute("cle", cle);
        model.addAttribute("ecole",ecole);
        model.addAttribute("articles",articles);
        return "article_liste_"+entity;

    }

    Ecole ecole = ecole_service.chercherEcole(id_school);
    List<Article> articles = article_service.allArticles(ecole);
    String cle = new String();

    model.addAttribute("cle", cle);
    model.addAttribute("ecole",ecole);
    model.addAttribute("articles",articles);
    return "article_liste_"+entity;
}
```

- La première partie où on mentionne la condition " if " est dédiée à une fonctionnalité de recherche que nous parlerons sur aprés.
- Le retour de notre fonction est une page d'affichage des listes d'articles. à savoir, chaque type d'utilisateur à une tel page dédiée à lui uniquement pour des raisons de fonctionnalités que vous pourrez consulter dans la partie vue du rapport.

2) getArticle :

Cette méthode nous permet d'afficher le contenu d'un article bien précis que nous sélectionnons à partir de la page des listes d'article gérée par la méthode avant. Le code est comme suivant :

```
@GetMapping("{entity}/article/{id_school}/{article_id}")
public String getArticle(@PathVariable long id_school, @PathVariable long article_id, @PathVariable String entity
        ,Model model) {
    String username = FileUtil.findUser();
    User user = user_service.findUser(username);
    Ecole ecole = ecole_service.chercherEcole(id_school);
    Article article_précis = article_service.getArticle(article_id);
    model.addAttribute("ecole",ecole);
    model.addAttribute("article",article_précis);
    model.addAttribute("user", user);
    return "article";
}
```

- Cette méthode doit produire à notre page les éléments suivants :
  - l'objet d'utilisateur courant : on le trouve à partir de la fonction "findUser()" que nous discuterons aprés.
  - l'objet de l'article en question que nous trouvons à partir de la méthode "getArticle()" qui nous produit l'article par son identifiant. ce dernier est passé dans l'URL.
  - l'objet de l'école liée à cette article pour pouvoir afficher quelques informations liées à elle dans la page tel que le nom, nombre de staff, son image, etc.
- Ces elements sont passées notre objet "model". à la fin, on retourne la page concernée par l'affichage des articles qui est la page "article".

3) AddArticle :

Cette méthode nous aide à nous faire envoyer fais la page concernée par l'ajout des articles des écoles. cette méthode est comme suivant :

```
@GetMapping("{entity}/article/{id_school}/newArticle")
public String addArticle(@PathVariable long id_school, @PathVariable String entity ,Model model) {

        String username = FileUtil.findUser();
        User user = user_service.findUser(username);
        Ecole ecole = ecole_service.chercherEcole(id_school);
        model.addAttribute("ecole",ecole);
        model.addAttribute("user", user);
        return "newArticle";

}
```

Comme cette méthode l'indique, on envoie à notre page l'objet utilisateur et ecole (cherché pour pouvoir passer l'identifiant de l'ecole dans le l'URL de la prochaine méthode).

4) saveArticle :

Notre méthode s'éxecute directement aprés la validation de notre formulaire. elle a comme but de recevoir tous les paramètres liée au formulaire et puis appeler une fonction dans notre "service" qui permet de stocker le nouveau article.

```
@PostMapping("{entity}/article/{id_school}/validForm")
public String saveArticle( Model model, @PathVariable long id_school, @PathVariable String entity, @RequestParam("user") long user_id,
        @RequestParam("image") MultipartFile images , @RequestParam("title") String title , @RequestParam("message") String message ,
        @RequestParam("cle") String cle) {
    try {
        article_service.ajouterArticleEcole(id_school, title,images,message,user_id,cle);
    } catch (IOException e) {
        e.printStackTrace();
    }
    return "redirect:/"+entity+"/article/"+id_school;
}
```

à la fin de la méthode, nous somme dirigée vers le contrôleur déjà expliquée qui permet d'afficher tous les articles de notre école sélectionner.

Notre méthode du service concernée est comme suivant :

```
@Override
public boolean ajouterArticleEcole(long ecole_id,String title,MultipartFile img, String message, long user_id,String cle) throws IOException {

        LocalDate currentDate = LocalDate.now();
        DateTimeFormatter dateFormat = DateTimeFormatter.ofPattern("yyyy-MM-dd");
        String dateString = currentDate.format(dateFormat);
        User user = ur.findById(user_id).get();
        Ecole ecole = er.findById(ecole_id).get();

        Article article = new Article();
        article.setDate(dateString);
        article.setText(message);
        article.setTitre(title);
        article.setEcole(ecole);
        article.setUser(user);
        article.setCle(cle);
         try {
            article.setImg(Base64.getEncoder().encodeToString(img.getBytes()));
        } catch (IOException e) {
            e.printStackTrace();
        }
         Article result = ar.save(article);
        if (result.equals(article)) {
            return true;
        }
        else {
            return false;
        }

}
```

Une petite explication sur notre méthode de service :

En effet, cette méthode enregistre la date sous la forme (YYYY-MM-DD) du moment de création de l'article, ensuite elle met les informations concernant l'article comme l'identitifant d'école, d'utilisateur, etc. à la fin, on enregistre nos article à l'aide de "repository" des articles.

Ce qui est à noter est que les images liée aux articles sont enregistrée sous une forme encodée (format blob, base64) comme suivant dans notre base :

img
[BLOB - 54.3 KiB]
[BLOB - 65.3 KiB]

4) modifierArticle :

Cette méthode nous permet de nous envoyer vers le formulaire de modification d'un article école. Elle cherche l'utilisateur courant ainsi que l'article voulu à fin d'insérer ces informations dans le formulaire puis nous envoie vers la page de modification est "modifierArticle".

```
@GetMapping("/{entity}/article/{id_school}/{article_id}/modify")
public String modifierArticle(@PathVariable long id_school, @PathVariable long article_id, @PathVariable String entity,Model model) {
    String username = FileUtil.findUser();
    User user = user_service.findUser(username);
    Article article_ = article_service.getArticle(article_id);
    //model.addAttribute("entity",entity);
    model.addAttribute("article_",article_);
    model.addAttribute("user", user);
    return "modifierArticle";

}
```

5) saveModifierArticle :

Cette méthode s'éxecute directement aprés notre méthode précédente. celle-là permet de recevoir tous les paramètre du formulaire, puis les insérer dans une méthode service qui est similaire à celle d'ajouter école à fin d'enregistrer les nouvelles modifications. à la fin, nous serons envoyée vers la page de la liste des articles aprés etre envoyer au contrôleur concernée déjà expliqué comme l'indique le retour de notre méthode.

```
@PostMapping("/{entity}/article/{id_school}/{article_id}/validModify")
public String saveModifierArticle(@ModelAttribute("article_") Article article_,@PathVariable long id_school, @PathVariable long article_id ,
        @PathVariable String entity,Model model ,@RequestParam("image") MultipartFile images , @RequestParam("title") String title ,
        @RequestParam("message") String message, @RequestParam("user") long user_id, @RequestParam("cle") String cle
        ) {

    article_service.modifierArticle(article_,id_school, images, title, message,user_id,cle);

    return "redirect:/"+entity+"/article/"+String.valueOf(id_school);
}
```

```
@Override
public boolean modifierArticle(Article article_,long id_school,MultipartFile img,String title,String message,long user_id,String cle)  {
    LocalDate currentDate = LocalDate.now();
    DateTimeFormatter dateFormat = DateTimeFormatter.ofPattern("yyyy-MM-dd");
    String dateString = currentDate.format(dateFormat);
    User user = ur.findById(user_id).get();
    article_.setTitre(title);
    article_.setText(message);
    article_.setDate(dateString);
    article_.setUser(user);
    article_.setCle(cle);
    Ecole ecole = er.findById(id_school).get();
    article_.setEcole(ecole);
    try {
        article_.setImg(Base64.getEncoder().encodeToString(img.getBytes()));
    } catch (IOException e) {
        e.printStackTrace();
    }
    Article result = ar.save(article_);
    if (Objects.nonNull(result)) {
        return true;
    }
    else {
        return false;
    }
```

6) suppArticle:

Cette méthode est dédiée à la suppression des articles de type école. Le code est comme suivant :

```
@GetMapping("/{entity}/article/{id_school}/{article_id}/delete")
public String suppArticle(@PathVariable long id_school, @PathVariable long article_id, @PathVariable String entity ,Model model) {
    Article article_supp = article_service.getArticle(article_id);
    boolean result = article_service.supprimerArticle(article_supp);
    if (id_school !=0) {
        if (result == true) {
            return "redirect:/"+entity+"/article/"+String.valueOf(id_school);
        } else {
            return "redirect:/"+entity+"/article/"+String.valueOf(id_school)+"/"+String.valueOf(article_id);
        }
    }
        else {
            if (entity.equalsIgnoreCase("user")) {
            return "redirect:/user/list/article";
        } else {
            return "redirect:/admin/list/articles";
        }
        }
}
```

Cette méthode a besoin comme entrée l'identifiant de l'article. La suppression est faite à l'aide de la méthode service "supprimerArticle" qui utilise la méthode delete de la "repository" ArticleRepo. Le condition de l'identitifiant de school est utile pour la partie portail admin/User à fin de nosu envoyer vers ces portails si on lance la suppression à partir d'eux au lieux des pages de liste d'article. Notre méthode service est défini comme suivant :

```
@Override
public boolean supprimerArticle(Article a) {
    try {
        ar.delete(a);
        return true;
    }catch(Exception e) {
        System.out.println("can't delete article");
        return false;
    }
}
```

Contrôleur des articles Catégories:

Les Articles de type Catégorie ont des méthodes similaires avec meme principes et buts. On passera maintenant à des méthodes communes utilisées pour les 2 types.

Contrôleur des articles Communs :

1) Search :

On a 3 méthodes de recherche des articles à partir d'un mot clé ou d'une série de mot séparée par des ",". ces méthodes sont basée sur une méthode de service sous la forme suivant :

Pour la première partie de la méthode, on divise l'entrée d'utilisateur en plusieurs mots par " , " , puis nous récupérons tous les articles disponibles. Ensuite, nous itérons sur les articles récupérées et voir si les mots clé (property de la classe article) sont disponibles (ou une) dans les mots clés de l'utilisateur.
à la fin, nous aurons une liste "articles_temp" contenant l'ensemble des articles vérifiant la disponibilités des mots clé.

```
@Override
public List<Article> AllArticlesByclé(String cle,long id,String related,User user) {

    try {
        List<Article> articles = new ArrayList<Article>();
        List<Article> articles_temp = new ArrayList<Article>();
        if (cle.equalsIgnoreCase("")) {
            return null;
        }
        String[] strArray = cle.trim().toLowerCase().split(",");
        for (String str:strArray) {

            List<Article> articles_ = ar.findAll();

            for (Article a : articles_) {
                String[] array_clés = a.getCle().trim().toLowerCase().split(",");
                boolean hasCle = Arrays.stream(array_clés)
                         .map(str_ -> str_.split(","))
                         .flatMap(Arrays::stream)
                         .anyMatch(word -> word.equals(str));

                if (hasCle) {
                        articles_temp.add(a);
                    }
                }
            }
```

Pour la deuxième partie de la méthode, la première condition est pour vérifier si l'utilisateur est est de type "user", pour ce cas, on choisit que les articles qui sont liée à ce utilisateur. cette recherche nous servit dans l'espace des articles propres à l'utilisateur lui-meme seulement.

```
if (user!=null && related.equalsIgnoreCase("user") )
    {

        for (Article art_:articles_temp)
            {

                if (art_.getUser().getUsername().equalsIgnoreCase(user.getUsername()))
                {
                    articles.add(art_);

                }
            }
    }
```

Pour la 3ème partie, On vérifie si l'utilisateur est un Admin. Si c'est le cas, on prend tous les articles sans aucune exception. cela est utile pour les articles dans l'espace Administrateur. Si ce n'est pas le cas, on passe à des contrôleur de public qui prennent comme indicateur dans les conditions les types d'article (école/categories) et vérifie l'identifiant de catégorie/école pour choisir les articles convenables comme le code suivant montre :

```
else if(related.equalsIgnoreCase("admin"))
        {
            articles.addAll(articles_temp);
        }
    else
        {
            for (Article art:articles_temp)
                {
                    if (related.equals("categorie"))
                        {
                            if (art.getCategorie().getId()==id)
                                {
                                articles.add(art);
                                }
                        }
                    else
                        {
                            if (art.getEcole().getEcole_id()==id)
                                {
                                    articles.add(art);
                                }
                        }
                }
        }
```

Enfin, on convertie notre liste en une set et puis convertir cette dernière en une liste à fin de supprimer des doublons d'articles si c'est le cas.

```
    Set<Article> setWithoutDuplicates = new HashSet<>(articles);
    List<Article> articles__ = new ArrayList<>(setWithoutDuplicates);
    return articles__;
    }
 catch(Exception e) {
    return null;
 }
```

On retourne la liste d'articles que nous cherchons, sinon on retourne la une liste vide qui permet d'affiche à l'utilisateur aucun article.

Nos contrôleurs de recherche sont les controleurs suivants :

- Controleur pour le réseau public - Article Ecole :

```
@PostMapping("/{entity}/article/{id_school}/search")
public String searchArticleEcole(@RequestParam("cle") String cle,@PathVariable long id_school,
        @PathVariable String entity,Model model , HttpSession session) {

    String username = FileUtil.findUser();
    User user = user_service.findUser(username);
    model.addAttribute("user", user);

    Ecole ecole = ecole_service.chercherEcole(id_school);
    List<Article> articles = article_service.AllArticlesByclé(cle,id_school,"school",null);

    session.setAttribute("cle", cle);
    session.setAttribute("articles", articles);
    session.setAttribute("ecole", ecole);

    return "redirect:/" + entity + "/article/" + id_school;
}
```

- Contrôleur pour le réseau Public- Article Catégorie :

```
@PostMapping("/{entity}/vie/{id_category}/search")
public String searchArticleCategory(@RequestParam("cle") String cle,
        @PathVariable long id_category,
        @PathVariable String entity,Model model , HttpSession session) {

    Optional<Categorie> category = categorie_service.CategorieByID(id_category);
    List<Article> articles = article_service.AllArticlesByclé(cle,id_category,"categorie",null);

    session.setAttribute("cle", cle);
    session.setAttribute("articles", articles);
    session.setAttribute("object", category.get());

    return "redirect:/" + entity + "/vie/" + id_category;
```

- Contrôleur pour le Portail User - Admin :

```
@PostMapping("/{entity}/list/articles/search")
public String searchArticleAdmin(@RequestParam("cle") String cle, @PathVariable String entity,Model model , HttpSession session) {
    String username = FileUtil.findUser();
    User user = user_service.findUser(username);
    model.addAttribute("user", user);
    List<Article> articles = article_service.AllArticlesByclé(cle,0,entity,user);

    session.setAttribute("cle", cle);
    session.setAttribute("articles", articles);
    model.addAttribute("articles", articles);
    if (entity.equalsIgnoreCase("admin")) {
    return "redirect:/"+entity+"/list/articles";
    } else if (entity.equalsIgnoreCase("user")) {
        return "redirect:/"+entity+"/list/article";
    }
    return null;
}
```

2) Accédé au portail d'administration Admin - Article User :

On a 2 portail de gestion d'articles, un utilisé par l'admin où il peut gérer tout type d'article et l'autre par user où il peut gérer seulement ses propres Article Ecole.
L'accées au portail d'admin n'est accessible que pour un utilisateur de type admin et la méthode responsable est la suivante :

```
//Entity Listing
@SuppressWarnings("unchecked")
@GetMapping("/admin/list/{entity}")
public String adminNavigationListing(@PathVariable String entity,Model model,HttpSession session) {
    if ((String) session.getAttribute("cle")!=null && (List<Article>) session.getAttribute("articles")!=null) {
        String cle = (String) session.getAttribute("cle");
        List<Article> articles = (List<Article>) session.getAttribute("articles");
        session.removeAttribute("cle");
        session.removeAttribute("articles");
        model.addAttribute("cle", cle);
        model.addAttribute("articles",articles);
        return "listArticle_portAdmin";
    }
    if(entity.equalsIgnoreCase("ecoles")) {
        model.addAttribute("ecoles", EcoleSrvcs.toutEcole());
        return "listEcoles.html";
    }
    if(entity.equalsIgnoreCase("users")) {
        model.addAttribute("users", UserSrvcs.toutUser());
        return "listUsers.html";
    }
    if(entity.equalsIgnoreCase("articles")) {
        model.addAttribute("articles", article_service.allArticlesDirect());
        return "listArticle_portAdmin.html";
    }
    return "admin.html";
}
```

Comme d'écrit techniquement, si "entity = articles", aloes c'est le portail administrateur qui sera lancée. si l'utilisateur n'est pas de type Admin, **Spring Security** l'arretera d'accéder à la page.

Pour le portail utilisateur, la méthode disponible pour cela est similaire au ancienne méthodes déjà vue, le code de cette méthode :

```
@GetMapping("/user/list/article")
public String getArticles(Model model,HttpSession session) {
    if ((String) session.getAttribute("cle")!=null && (List<Article>) session.getAttribute("articles")!=null ) {
        String cle = (String) session.getAttribute("cle");
        session.removeAttribute("cle");
        List<Article> articles = (List<Article>) session.getAttribute("articles");
        model.addAttribute("cle", cle);
        model.addAttribute("articles",articles);
        return "listArticle_portUser";
    }
    //get the user here
    String username = FileUtil.findUser();
    User user = user_service.findUser(username);
    List<Article> articles = article_service.findArticlesByUser(username);
    String cle ="";
    session.setAttribute("cle", cle);
    model.addAttribute("articles", articles);
    model.addAttribute("user", user);
    return "listArticle_portUser";
}
```

3) Modification dans le portail Administrateur - User :

La modification les portails soit Administrateur / User est faite à l'aide de 2 méthodes similaires à ceux d'avant. Pour différencier entre le type d'article , on utilise "vieport" dans l'URL pour les demandes concernant les articles catégories, et "articleport" dans l'URL pour les demandes concernant les articles écoles. Les 2 méthodes sont comme suivant :

```java
@GetMapping("/{entity}/{article}/{id}/{article_id}/modifyAdmin")
public String getmodifierArticle(@PathVariable long id, @PathVariable long article_id,
        @PathVariable String entity,Model model,@PathVariable String article) {
    String username = FileUtil.findUser();
    User user = user_service.findUser(username);
    if (article.equalsIgnoreCase("articleport")) {
    Article article_ = article_service.getArticle(article_id);
    model.addAttribute("article_",article_);
    model.addAttribute("user", user);
    } else if (article.equalsIgnoreCase("vieport")) {
        Article article_ = article_service.getArticle(article_id);
        model.addAttribute("article_",article_);
        model.addAttribute("user", user);
    }
    return "modifierArticleAdminUser";

}

@PostMapping("/{entity}/{article}/{id}/{article_id}/validModifyAdmin")
public String getmodifierArticle(@ModelAttribute("article_") Article article_,
        @PathVariable long id,
        @PathVariable long article_id , @PathVariable String entity,Model model ,
        @RequestParam("image") MultipartFile images ,@RequestParam("title") String title ,
        @RequestParam("message") String message, @RequestParam("user") long user_id,
        @PathVariable String article,@RequestParam("cle") String cle
        ) {

    if (article.equalsIgnoreCase("articleport")) {
    article_service.modifierArticle(article_,id, images, title, message,user_id,cle);
    } else if (article.equalsIgnoreCase("vieport")) {
        categorie_service.modifierArticle(article_, id , images, title, message,user_id,cle);
    }
    if (entity.equalsIgnoreCase("user")) {
        return "redirect:/user/list/article";
    } else {
        return "redirect:/admin/list/articles";
    }
}
```

la première méthode nous dirige vers la page de modification et le deuxième vers le contrôleur qui mène a portail soit d'administrateur soit User.

D - x) Controleurs de Contact:

Ce contrôleur est crée spécifiquement pour la page Contact à fin de mettre en position la possibilité de nous contacter. ce contrôleur reçoit les données de formulaire et utilise la bibliothèque "**javax.mail**" à coté d'une configuration du fichier **application.properties** et une configuration sur le compte **Google** qui recoit les emails. La configuration du fichier applciation.properties nécessaire :

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=<Login User to SMTP server>
spring.mail.password=<Login password to SMTP server>
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Pour le contrôleur, voici le code correspondant :

```java
@Controller
public class ContactController {

    @RequestMapping(value = "/sendEmail", method = RequestMethod.POST)
    public String contact(@RequestParam("name") String name,
                          @RequestParam("email") String email,
                          @RequestParam("website") String website,
                          @RequestParam("address") String address,
                          @RequestParam("subject") String subject,
                          @RequestParam("message") String message,
                          Model model) {

        String to = "eljaouharimohamed.2001@gmail.com"; // Email address to send the message to

        // Set up properties for the mail session
        Properties props = new Properties();
        props.put("mail.smtp.host", "smtp.gmail.com");
        props.put("mail.smtp.port", "587");
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");

        // Set up the mail session
        Session session = Session.getInstance(props, new Authenticator() {
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication("user_name", "password");
            }
        });

        try {
            // Create a new message
            Message mailMessage = new MimeMessage(session);
            mailMessage.setFrom(new InternetAddress(email));
            mailMessage.setRecipient(Message.RecipientType.TO, new InternetAddress(to));
            mailMessage.setSubject(subject);

            // Set the message body
            String messageBody = "Name: " + name + "\n" +
                                 "Email: " + email + "\n" +
                                 "Website: " + website + "\n" +
                                 "Address: " + address + "\n\n" +
                                 "Message:\n" + message;
            mailMessage.setText(messageBody);

            // Send the message
            Transport.send(mailMessage);

            // Add a success message to the model
            model.addAttribute("successMessage", "Your message has been sent!");

        } catch (MessagingException e) {
            // Add an error message to the model
            model.addAttribute("errorMessage", "There was an error sending your message. Please try again later.");
        }

        // Return the view name
        return "redirect:/public/Contact";
    }
```

D - x) Controleurs de Catégorie :

Notre contrôleur de catégorie permet comme actions à l'Admin seulement d'ajouter des catégories, les supprimer et les modifiers.

1) Méthodes d'ajout :

Comme avant, on a toujours 2 méthodes d'ajout :

```
@GetMapping("/admin/vieadmin/newCat")
public String addCategorie(Model model) {
        String username = FileUtil.findUser();
        User user = user_service.findUser(username);
        model.addAttribute("user", user);
        return "newCategorie";
    }


@PostMapping("/admin/vieadmin/validnewcat")
public String ConfirmaddCategorie(Model model,@RequestParam("image") MultipartFile images ,
        @RequestParam("title") String title,@RequestParam("user") long user_id) throws IOException {
    categorie_service.ajouterCategory(title, images,user_id);
    return "redirect:/admin/list/vieadmin";
}

@GetMapping("/admin/{category_id}/deleteCat")
public String deleteCat(@PathVariable long category_id) {
    categorie_service.deleteCat(categorie_service.CategorieByID(category_id).get());

    return "redirect:/admin/list/vieadmin";

}
```

Pour la méthode de service utiliser pour l'ajout, son code est similaire au méthode service d'ajout d'article.

2) Méthodes de modification :

Comme avant, on a toujours 2 méthodes de modification :

```
@GetMapping("/admin/vieadmin/modifyCat/{category_id}")
public String modifierCategory(@PathVariable long category_id,Model model) {
    String username = FileUtil.findUser();
    User user = user_service.findUser(username);
    model.addAttribute("user", user);
    Categorie cat = categorie_service.CategorieByID(category_id).get();
    model.addAttribute("categorie",cat);
    return "modifierCategory";
}


@PostMapping("/admin/vieadmin/Confirmmodify")
public String ConfirmmodifierCategory(Model model,@ModelAttribute("categorie") Categorie cat
        ,@RequestParam("image") MultipartFile images , @RequestParam("title") String title ,
        @RequestParam("user") long user_id, @RequestParam("id") long id_categorie
        ) throws IOException {

    categorie_service.modifierCat(title, images,user_id,id_categorie);

    return "redirect:/admin/list/vieadmin";

}
```

G- Conclusion et Remerciement

Dans ce projet, nous avons employé nos connaissances en développement web, en technologie JEE, et profiter des facilitées présenté par la framework Spring, y compris Spring Boot, Spring Security et Spring Web dans leurs derniers versions (pour le mois d'Avril 2023). à fin de réaliser une application compacte, dynamique et interactive pour faciliter l'accès et le partage de l'information en terms d'éducation.

Nous tenons à remercier madame Ibtissam Bakkouri pour son accompagnement et son encadrement tout au long du projet.

G- Conclusion et Remerciement

Dans ce projet, nous avons employé nos connaissances en développement web, en technologie JEE, et profiter des facilitées présenté par la framework Spring, y compris Spring Boot, Spring Security et Spring Web dans leurs derniers versions (pour le mois d'Avril 2023). à fin de réaliser une application compacte, dynamique et interactive pour faciliter l'accès et le partage de l'information en terms d'éducation.

Nous tenons à remercier madame Ibtissam Bakkouri pour son accompagnement et son encadrement tout au long du projet.

G- Conclusion et Remerciement

Dans ce projet, nous avons employé nos connaissances en développement web, en technologie JEE, et profiter des facilitées présenté par la framework Spring, y compris Spring Boot, Spring Security et Spring Web dans leurs derniers versions (pour le mois d'Avril 2023). à fin de réaliser une application compacte, dynamique et interactive pour faciliter l'accès et le partage de l'information en terms d'éducation.

Nous tenons à remercier madame Ibtissam Bakkouri pour son accompagnement et son encadrement tout au long du projet.