

Université Assane SECK de Ziguinchor



L'excellence, ma référence

UFR Sciences & Technologies
Département Informatique
Master 1 Informatique

**** RAPPORT DE PROJET ****

***GESTION DES INSCRIPTIONS
PEDAGOGIQUES***

[Application Java - Swing]

Module :

Programmation Avancée

Ziguinchor, Mars 2025

Mis en lumière Par :

Safiétou DIALLO

El Hadji Abdou DRAME

Sous la directive

Mr Khadim DRAME

Année Académique 2023 - 2024

Table des figures

| | |
|---|----|
| Figure 1:Diagramme entité association initiale..... | 3 |
| Figure 2:Arborescence du projet | 5 |
| Figure 3:Classe Role.java..... | 6 |
| Figure 4:Classe Utilisateur.java..... | 6 |
| Figure 5:Classe ResponsablePedagogique.java..... | 7 |
| Figure 6:Classe Etudiant.java..... | 7 |
| Figure 7: Classe Formation.java..... | 7 |
| Figure 8:Classe Groupe.java | 8 |
| Figure 9:Classe Enseignant.java | 8 |
| Figure 10:Classe UE.java | 9 |
| Figure 11:Classe Inscription.java | 9 |
| Figure 12:Méthode InscrireEtudiant dans InsriptionService.java..... | 10 |
| Figure 13:Méthode InscrireEtudiant dans InsriptionService.java (Suite)..... | 10 |
| Figure 14:Méthode accepterInscrition..... | 11 |
| Figure 15:Méthode refuserInscrition..... | 11 |
| Figure 16:Méthode repartirEtudiant..... | 12 |
| Figure 17:Méthode exportEtudiantsByGroupe | 12 |
| Figure 18:Interface d'inscription (Choix de la formation) | 13 |
| Figure 19:Interface d'inscription (Choix des UE optionnelles)..... | 13 |
| Figure 20:Interface d'inscription (Confirmation d'Inscription) | 14 |
| Figure 21:Interface du responsable pédagogique | 14 |
| Figure 22:Validation d'une inscription | 14 |
| Figure 23:Email reçu par l'étudiant après validation | 14 |
| Figure 24:Répartition des étudiants par groupe | 15 |
| Figure 25:Exportation des données en format PDF | 15 |
| Figure 26:Exemple : English-Student.pdf..... | 15 |
| Figure 27:Exportation des données en format CSV | 16 |
| Figure 28:Exportation des données en format csv d'un groupe d'étudiant..... | 16 |

Table des matières

| | |
|---|----|
| Table des figures..... | i |
| INTRODUCTION..... | 1 |
| I. Présentation du cahier des charges | 2 |
| I.1. Objectif du projet | 2 |
| I.2. Spécifications Techniques | 2 |
| II. Modélisation et structure de la base de données | 2 |
| II.1. Diagramme Entité Association..... | 2 |
| II.2. Relation entre les tables..... | 3 |
| III. Structure des POJO et conception logicielle | 5 |
| III.1. Arborescence du projet..... | 5 |
| III.2. Présentation des classes POJO | 6 |
| IV. Fonctionnalités réalisées | 9 |
| IV.1. Inscriptions pédagogiques | 10 |
| IV.2. Validation et Refus des inscriptions pédagogique..... | 10 |
| IV.3. Répartition automatique | 11 |
| IV.4. Exportation des données en format CSV et PDF | 12 |
| V. IHM (Interface Homme-Machine) | 13 |
| Conclusion..... | 17 |

INTRODUCTION

Ce rapport présente le développement d'une application de gestion des inscriptions pédagogiques destinée aux établissements d'enseignement supérieur plus particulièrement l'Université Assane SECK de Ziguinchor. L'objectif principal de ce projet est d'automatiser le processus d'inscription aux formations et aux unités d'enseignement (UE), tout en facilitant la gestion des groupes de travaux dirigés (TD) et de travaux pratiques (TP).

La solution développée permet aux étudiants de s'inscrire en ligne en sélectionnant leurs UE optionnelles selon les règles définies par l'établissement. Les responsables pédagogiques disposent, quant à eux, d'un espace dédié pour valider ou refuser les inscriptions, organiser les groupes et suivre l'évolution des étudiants inscrits.

Ce document détaille le cahier des charges du projet, la conception de l'architecture logicielle, le modèle de base de données ainsi que les différentes interfaces utilisateur mises en place. Une attention particulière est portée aux fonctionnalités réalisées et aux aspects techniques de l'implémentation.

I. Présentation du cahier des charges

I.1. Objectif du projet

Le projet vise à apporter une solution efficace et automatisée à la gestion des inscriptions pédagogiques dans les établissements d'enseignement supérieur, en particulier à l'UASZ (Université Assane SECK) de Ziguinchor. Il répond aux besoins des étudiants, des responsables pédagogiques et de l'administration en mettant en place un système optimisé pour gérer les inscriptions et la répartition des étudiants dans les groupes. Les principaux objectifs sont les suivants :

- **Automatiser la gestion des inscriptions pédagogiques**
- **Faciliter la validation et le suivi des inscriptions**
- **Assurer une répartition efficace des étudiants dans les groupes de TD et TP**
- **Offrir des fonctionnalités d'exportation des données**

I.2. Spécifications Techniques

Afin de garantir une solution robuste et évolutive, plusieurs choix technologiques et méthodologiques ont été adoptés. Ces choix permettent d'assurer l'efficacité du système, tant pour les étudiants que pour les responsables de l'UASZ.

- **Langage de développement** : java
- **Interface utilisateur** : Swing
- **Langage de modélisation** : UML (Unified Modeling Language)
- **Persistance des données** : Hibernate (ORM)
- **Génération des documents** : OpenPDF pour l'exportation des données en PDF

II. Modélisation et structure de la base de données

La base de données est essentielle au système de gestion des inscriptions pédagogiques, centralisant les informations de notre système. Sa conception a été réalisée pour répondre aux besoins du projet, assurant la fiabilité, l'évolutivité et une utilisation intuitive, servant de fondation au bon fonctionnement du système.

II.1. Diagramme Entité Association

La base de données est structurée autour de plusieurs entités principales qui interagissent entre elles à travers des relations spécifiques. Les entités et leurs relations sont illustrées dans le diagramme entité-association suivant :

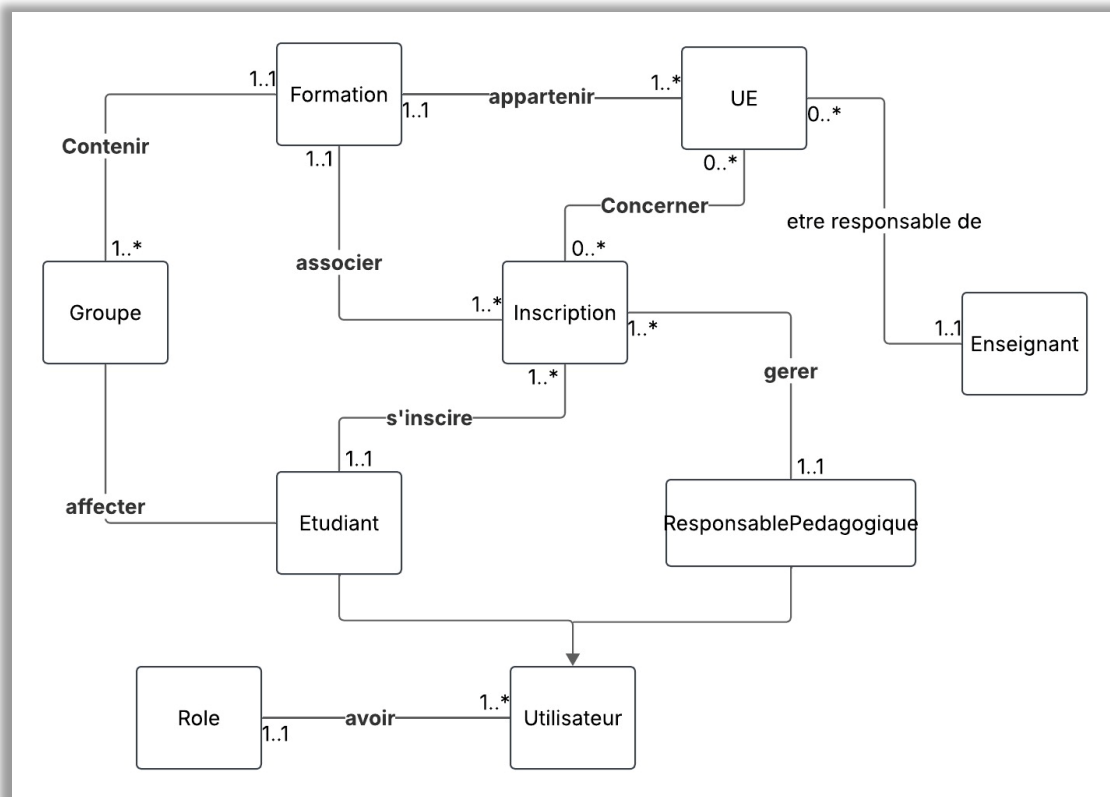


Figure 1: Diagramme entité association initiale

II.2. Relation entre les tables

➤ Relation Étudiant - Inscription

Un étudiant peut s'inscrire à plusieurs formations.

Une inscription est liée à un étudiant et à une formation spécifique.

Cardinalité : Un étudiant peut avoir plusieurs inscriptions (relation un-à-plusieurs).

➤ Relation Inscription - Formation

Une inscription est associée à une seule formation, mais une formation peut comporter plusieurs inscriptions.

Cardinalité : Une inscription appartient à une seule formation, et chaque formation peut avoir plusieurs inscriptions (relation un-à-plusieurs).

➤ Relation Inscription - UE

Chaque inscription peut concerner plusieurs unités d'enseignement (UE). Cela crée une relation plusieurs-à-plusieurs entre les tables Inscription et UE, qui sera gérée à travers une table de jonction appelée inscription_ue_optionnelle.

➤ **Relation Formation - UE**

Une formation comprend plusieurs unités d'enseignement (UE).

Une UE appartient à une formation donnée.

Cardinalité : Une formation peut inclure plusieurs UE, et chaque UE appartient à une seule formation (relation un-à-plusieurs).

➤ **Relation Étudiant - Groupe de TD/TP**

Chaque étudiant est affecté à un groupe de TD/TP spécifique, en fonction de la répartition automatique basée sur des critères comme l'ordre alphabétique ou la capacité des groupes.

Cardinalité : Un étudiant peut être affecté à un seul groupe de TD/TP, mais un groupe peut accueillir plusieurs étudiants (relation plusieurs-à-un).

➤ **Relation Responsable – Inscription**

Un responsable peut gérer une ou plusieurs inscriptions.

Une inscription est gérée par un et un seul responsable.

Cardinalité : Un responsable peut gérer plusieurs inscriptions, et chaque inscription est gérée par un seul responsable (relation un-à-plusieurs).

➤ **Relation Responsable – Formation**

Un responsable peut gérer une ou plusieurs formations.

Une formation est gérée par un et un seul responsable.

Cardinalité : Un responsable peut gérer plusieurs formations, et chaque formation est gérée par un seul responsable (relation un-à-plusieurs).

➤ **Relation Enseignant – UE**

Un enseignant est responsable de plusieurs UE

Une UE est dispensée par un et un seul enseignant

Cardinalité : Un enseignant peut dispenser plusieurs UE, et chaque UE est gérée par un seul enseignant (relation un-à-plusieurs).

➤ Relation Utilisateur– Rôle

Un utilisateur peut avoir un et un seul rôle.

Un rôle concerne un et un seul utilisateur.

Cardinalité : Un utilisateur peut jouer un seul rôle, et chaque rôle n'appartient qu'à un seul utilisateur (relation un-à-un)

➤ Relation Groupe – Formation

Une formation peut contenir un ou un plusieurs groupes.

Un groupe se trouve dans une et une seule formation.

Cardinalité : Une formation peut contenir plusieurs groupes, et chaque groupe se trouve dans une et une formation (relation un-à-plusieurs).

III. Structure des POJO et conception logicielle

III.1.Arborescence du projet

L'arborescence du projet a été organisée pour séparer clairement les différentes responsabilités de l'application et garantir une gestion fluide du code source. Chaque dossier et fichier a une fonction bien précise dans l'architecture de l'application. Voici la structure de base :

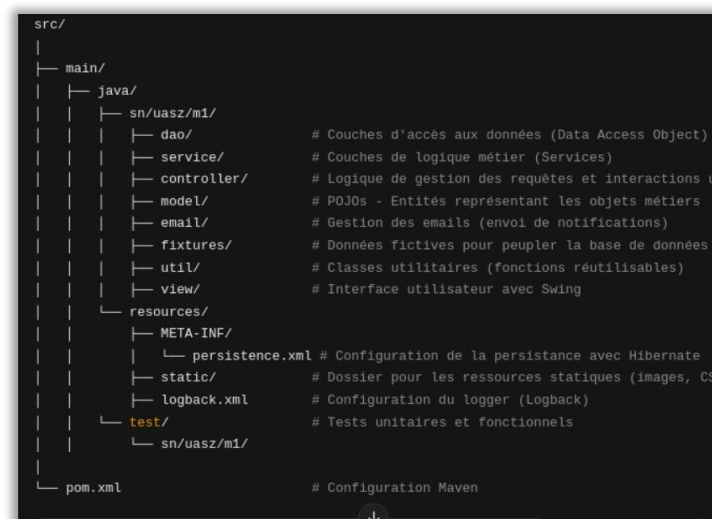
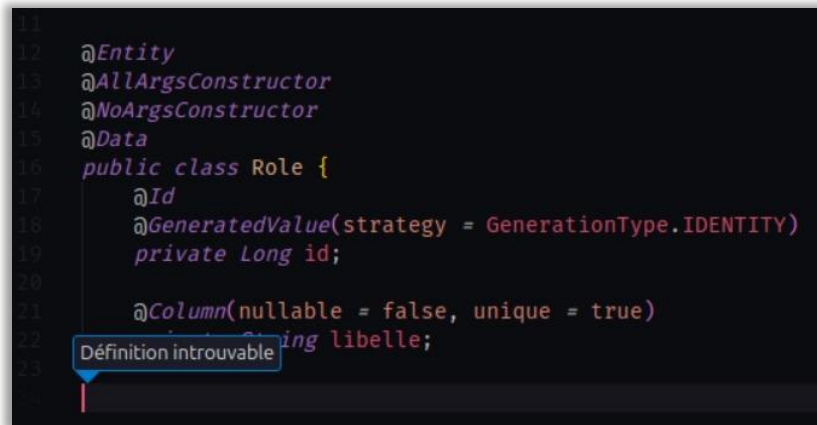


Figure 2:Arborescence du projet

III.2.Présentation des classes POJO

Les POJO représentent les entités de la base de données et sont utilisés pour la persistance des données via Hibernate.

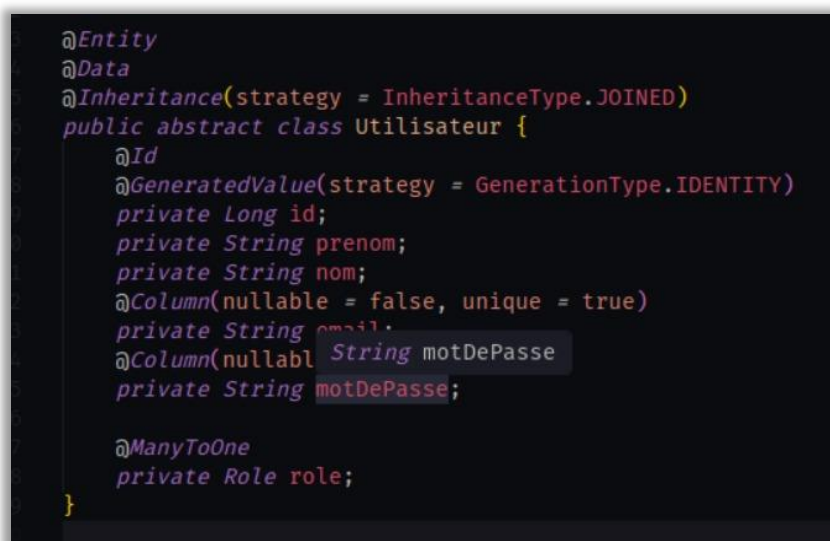
❖ **La classe Role** : Désigne le rôle d'un utilisateur.



```
11
12 @Entity
13 @AllArgsConstructor
14 @NoArgsConstructor
15 @Data
16 public class Role {
17     @Id
18     @GeneratedValue(strategy = GenerationType.IDENTITY)
19     private Long id;
20
21     @Column(nullable = false, unique = true)
22     private String libelle;
23 }
```

Figure 3:Classe Role.java

❖ **La classe Utilisateur** : La classe Utilisateur est une entité représentant un utilisateur du système de gestion des inscriptions pédagogiques.



```
3 @Entity
4 @Data
5 @Inheritance(strategy = InheritanceType.JOINED)
6 public abstract class Utilisateur {
7     @Id
8     @GeneratedValue(strategy = GenerationType.IDENTITY)
9     private Long id;
10    private String prenom;
11    private String nom;
12    @Column(nullable = false, unique = true)
13    private String email;
14    @Column(nullable = false)
15    private String motDePasse;
16
17    @ManyToOne
18    private Role role;
19 }
```

Figure 4:Classe Utilisateur.java

❖ **La classe ResponsablePdagogique** : La classe responsablePdagogique est une classe qui étend de la classe utilisateur et représente un responsable pédagogique, un utilisateur chargée de la gestion et de la validation des inscriptions dans une formation.

```

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
public class ResponsablePedagogique extends Utilisateur {
    private String departement;

    @OneToMany(mappedBy = "responsable", cascade = CascadeType.ALL)
    private List<Formation> formations = new ArrayList<>();

    @OneToMany(mappedBy = "responsable", cascade = CascadeType.ALL)
    private List<UE> ues = new ArrayList<>();
}

```

Figure 5: Classe ResponsablePedagogique.java

- ❖ **La classe Etudiant :** La classe Etudiant est une entité représentant un étudiant inscrit dans une formation dans notre système

```

@Entity
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class Etudiant extends Utilisateur {
    ... @Column(unique = true, length = 9)
    ... private String ine;
    ... private LocalDate dateNaissance;
    ... @Enumerated(EnumType.STRING)
    ... private Sexe sexe;
    ... private String adresse;
}

```

Figure 6: Classe Etudiant.java

- ❖ **La classe Formation :** Elle représente une formation dans l'établissement de notre système.

```

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Formation {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false, unique = true)
    private String libelle;
    private int niveau;

    @ManyToOne
    @JoinColumn(name = "responsable_id")
    private ResponsablePedagogique responsable;
}

```

Figure 7: Classe Formation.java

- ❖ **La classe Groupe :** Elle représente un groupe de travaux dirigés (TD) ou de travaux pratiques (TP) dans lequel les étudiants sont répartis.

```
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Groupe {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private int capacite;
    @Enumerated(EnumType.STRING)
    private TypeGroupe type;

    @ManyToOne
    @JoinColumn(name = "formation_id", nullable = false)
    private Formation formation;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(
        name = "Repartition",
        joinColumns = @JoinColumn(name = "groupe_id"),
        inverseJoinColumns = @JoinColumn(name = "etudiant_id")
    )
    private List<Etudiant> etudiants = new ArrayList<>();
}
```

Figure 8: Classe Groupe.java

- ❖ **La classe Enseignant :** Elle représente un enseignant qui est chargé d'un ou plusieurs groupes dans une unité d'enseignement (UE).

```
@Entity
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class Enseignant {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private String prenom;
    @Column(unique = true)
    private String email;
    private String specialite;

    public Enseignant(String nom, String prenom, String email, String specialite) {
        this.nom = nom;
        this.prenom = prenom;
        this.email = email;
        this.specialite = specialite;
    }
}
```

Figure 9: Classe Enseignant.java

- ❖ **La classe UE :** Elle représente une unité d'enseignement dans le programme d'une formation.

```

20 @NoArgsConstructor
21 @EqualsAndHashCode
22 public class UE {
23     @Id
24     @GeneratedValue(strategy = GenerationType.IDENTITY)
25     private Long id;
26     @Column(unique = true, nullable = false, length = 7)
27     private String code;
28     private String libelle;
29     private int volumeHoraire;
30     private int coefficient;
31     private int credit;
32     private boolean obligatoire = true;
33
34     @ManyToOne
35     @JoinColumn(name = "enseignant_id")
36     private Enseignant enseignant;
37
38     @ManyToOne
39     @JoinColumn(name = "formationId")
40     private Formation formation = null;
41
42     @ManyToOne
43     @JoinColumn(name = "created_by")
44     private ResponsablePedagogique responsable;
45 }

```

Figure 10: Classe UE.java

- ❖ **La classe Inscription :** Elle représente l'acte d'inscription d'un étudiant à une Formation et à des UE.

```

@NoArgsConstructor
@Data
public class Inscription {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "etudiant_id", nullable = false)
    private Etudiant etudiant;

    @ManyToOne
    @JoinColumn(name = "formation_id", nullable = false)
    private Formation formation;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name = "inscription_ue_optionnelle",
        joinColumns = @JoinColumn(name = "inscription_id"),
        inverseJoinColumns = @JoinColumn(name = "ue_id"))
    private List<UE> uesOptionnelles = new ArrayList<>();

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name = "ue_inscrites",
        joinColumns = @JoinColumn(name = "inscription_id"),
        inverseJoinColumns = @JoinColumn(name = "ue_id"))
    private List<UE> ues = new ArrayList<>();

    @Enumerated(EnumType.STRING)
    private StatutInscription statut = StatutInscription.EN_ATTENTE;
}

```

Figure 11: Classe Inscription.java

IV. Fonctionnalités réalisées

Le projet intègre plusieurs fonctionnalités essentielles visant à simplifier et automatiser la gestion des inscriptions pédagogiques. Ces fonctionnalités couvrent l'ensemble du processus, de l'inscription des étudiants à l'exportation des données en passant par la gestion des groupes et des utilisateurs.

IV.1. Inscriptions pédagogiques

- L'inscription des étudiants aux formations.
- Le choix des unités d'enseignement (UE) optionnelle a cette formation.

```
public void inscrireEtudiant(Long formationId, List<UE> ueChoisies) {
    try {
        // Récupérer l'utilisateur connecté
        Utilisateur connectedUser = SessionManager.getUtilisateur();
        Etudiant etudiant = etudiantService.getEtudiantById(connectedUser.getId());
        if (etudiant == null) {
            throw new RuntimeException(message:"Étudiant non trouvé.");
        }

        // Vérifier que la formation existe
        Formation formation = formationService.getFormationById(formationId);
        if (formation == null) {
            throw new IllegalArgumentException(s:"Formation non trouvée.");
        }

        // Vérifier si l'étudiant est déjà inscrit à cette formation
        boolean inscriptionExistante = inscriptionDAO.isInscriptionExistanteByFormation(etudiant.getId(),
            formationId);
        if (inscriptionExistante) {
            throw new RuntimeException(message:"L'étudiant a déjà une inscription à cette formation.");
        }

        boolean isInscription = inscriptionDAO.isInscriptionExistante(etudiant.getId());
        if (isInscription) {
            throw new RuntimeException(message:"L'étudiant est déjà inscrit à une formation.");
        }

        // Récupérer le responsable pédagogique de la formation
        ResponsablePedagogique responsable = formationService.getResponsablePedagogique(formationId);

        // Récupérer toutes les UEs optionnelles disponibles pour cette formation
        List<UE> uesOptionnellesDisponibles = formationService.getOptionalUEs(formationId);

        // Si la formation n'a pas d'UEs optionnelles
        if (uesOptionnellesDisponibles.isEmpty()) {
            // Créer une inscription sans UEs optionnelles
            Inscription inscription = new Inscription();
            inscription.setEtudiant(etudiant);
            inscription.setFormation(formation);
            inscription.setUesOptionnelles(new ArrayList<>());
            inscription.setUes(ues);
            inscriptionDAO.save(inscription);

            // Notifier le responsable de l'inscription de l'étudiant
            notificationService.notifierResponsableInscription(responsable.getEmail(), formation.getLibelle(),
                etudiant);

            System.out.println(x:"Inscription réussie sans UE optionnelle.");
            return; // Fin de la méthode si aucune UE n'est disponible

            // Sinon le cas où des UEs optionnelles sont choisies
            List<UE> ues = new ArrayList<>();
            System.out.println(x:"UEs sélectionnées par l'étudiant :");
            for (UE ue : ueChoisies) {
                ues.add(ue);
            }

            // Création de l'inscription pédagogique avec les UEs sélectionnées
            Inscription inscription = new Inscription();
            inscription.setEtudiant(etudiant);
            inscription.setFormation(formation);
            inscription.setUes(ues);
            inscription.setUesOptionnelles(ues);

            // Enregistrement de l'inscription
            inscriptionDAO.save(inscription);

            // Notifier le responsable de l'inscription
            notificationService.notifierResponsableInscription(responsable.getEmail(), formation.getLibelle(),
                etudiant);

            System.out.println(x:"Inscription pédagogique réussie avec les UEs sélectionnées !");
        } catch (Exception e) {
            throw new RuntimeException("Erreur lors de l'inscription pédagogique : " + e.getMessage());
        }
    }
}
```

Figure 12:Méthode *InscrireEtudiant* dans *InscriptionService.java*

```
// Créer une inscription sans UEs optionnelles
Inscription inscription = new Inscription();
inscription.setEtudiant(etudiant);
inscription.setFormation(formation);
inscription.setUesOptionnelles(new ArrayList<>());
inscription.setUes(ues);
inscriptionDAO.save(inscription);

// Notifier le responsable de l'inscription de l'étudiant
notificationService.notifierResponsableInscription(responsable.getEmail(), formation.getLibelle(),
    etudiant);

System.out.println(x:"Inscription réussie sans UE optionnelle.");
return; // Fin de la méthode si aucune UE n'est disponible

// Sinon le cas où des UEs optionnelles sont choisies
List<UE> ues = new ArrayList<>();
System.out.println(x:"UEs sélectionnées par l'étudiant :");
for (UE ue : ueChoisies) {
    ues.add(ue);
}

// Création de l'inscription pédagogique avec les UEs sélectionnées
Inscription inscription = new Inscription();
inscription.setEtudiant(etudiant);
inscription.setFormation(formation);
inscription.setUes(ues);
inscription.setUesOptionnelles(ues);

// Enregistrement de l'inscription
inscriptionDAO.save(inscription);

// Notifier le responsable de l'inscription
notificationService.notifierResponsableInscription(responsable.getEmail(), formation.getLibelle(),
    etudiant);

System.out.println(x:"Inscription pédagogique réussie avec les UEs sélectionnées !");
} catch (Exception e) {
    throw new RuntimeException("Erreur lors de l'inscription pédagogique : " + e.getMessage());
}
}
```

Figure 13:Méthode *InscrireEtudiant* dans *InscriptionService.java* (Suite)

IV.2. Validation et Refus des inscriptions pédagogique

Une fois l'inscription effectuée par l'étudiant, le responsable peut l'accepter ou la refuser. Son choix entraîne une mise à jour du statut de l'inscription. Une notification automatique par e-mail est ensuite envoyée à l'étudiant pour l'informer de la décision prise. Les captures d'écran ci-dessous illustrent ce processus.

```

public void accepterInscription(long inscriptionId) {
    Inscription inscription = getInscriptionById(inscriptionId);
    if (inscription == null) {
        throw new IllegalArgumentException(s:"L'inscription demandée est introuvable.");
    }

    if (inscription.getStatut() != StatutInscription.EN_ATTENTE) {
        throw new IllegalArgumentException(s:"L'inscription ne peut pas être modifiée.");
    }

    // Inscrire l'étudiant aux UEs obligatoires
    Formation formation = inscription.getFormation();
    List<UE> uesObligatoires = formationService.getRequiredUEs(formation.getId());
    List<UE> uesOptionnelles = inscription.getUesOptionnelles();

    List<UE> ues = new ArrayList<>();
    ues.addAll(uesObligatoires);
    ues.addAll(uesOptionnelles);
    inscription.setUes(ues);

    // inscription.setStatut(StatutInscription.ACCEPTEE);

    inscriptionDAO.update(inscription);

    // accepter l'inscription
    inscriptionDAO.accepterInscription(inscriptionId);

    // Envoyer un email de confirmation
    Etudiant etudiant = inscription.getEtudiant();
    notificationService.notifierEtudiantInscription(etudiant.getEmail(), formation.getLibelle());
    envoyerEmailValidation(etudiant, formation, uesObligatoires, uesOptionnelles);
}

```

Figure 14:Méthode accepterInscription

```

public void refuserInscription(Long id) {
    Inscription inscription = getInscriptionById(id);

    // * Récupérer l'étudiant et la formation associée
    Etudiant etudiant = inscription.getEtudiant();
    Formation formation = inscription.getFormation();

    // Inscription refuser
    inscriptionDAO.refuserInscription(id);

    // envoyer une notification
    notificationService.notifierEtudiantInscriptionRefuser(etudiant.getEmail(), formation.getLibelle());
    envoyerEmailRefus(etudiant, inscription.getFormation());
}

```

Figure 15:Méthode refuserInscription

IV.3. Répartition automatique

- Répartition automatique des étudiants dans les groupes
- Méthode permettant de répartir des étudiants dans les groupes de TD et TP en fonction des effectifs.
- Attribution des étudiants de manière équilibrée et alphabétique.
- Mise à jour automatique en cas d'ajout de nouveaux inscrits, en fusionnant les répartitions existantes.
- Garantie qu'un étudiant ne peut appartenir qu'à un seul groupe de TD et un seul groupe de TP.


```

public void repartirEtudiants(Long formationId) {
    List<Etudiant> etudiants = inscriptionDAO.findEtudiantsByFormation(formationId);
    if (etudiants.isEmpty()) {
        throw new RuntimeException(message:"Aucun étudiant inscrit dans cette formation.");
    }

    // Trier tous les étudiants (y compris ceux déjà répartis)
    etudiants.sort(Comparator.comparing(Etudiant::getNom).thenComparing(Etudiant::getPrenom));

    // Récupérer les étudiants déjà répartis
    List<Etudiant> etudiantsRepartis = groupeDAO.findEtudiantsRepartis(formationId);

    // Déterminer les nouveaux étudiants
    List<Etudiant> nouveauxEtudiants = new ArrayList<>(etudiants);
    nouveauxEtudiants.removeAll(etudiantsRepartis);

    if (nouveauxEtudiants.isEmpty()) {
        System.out.println(x:"❌ Aucun nouvel étudiant à répartir.");
        return;
    }

    System.out.println("🔴 Nouveaux étudiants à répartir : " + nouveauxEtudiants.size());

    // Répartition dans les groupes existants pour les types TD et TP
    fusionnerEtudiantsDansGroupes(formationId, TypeGroupe.TD, nouveauxEtudiants);
    fusionnerEtudiantsDansGroupes(formationId, TypeGroupe.TP, nouveauxEtudiants);
}

```

Figure 16:Méthode repartirEtudiant

IV.4. Exportation des données en format CSV et PDF

➤ Format CSV :

Génération de fichiers CSV contenant la liste des étudiants d'un groupe de TD pour faciliter l'organisation pédagogique.

```

public void exportEtudiantsByGroupe(Groupe groupe, String filePath) {
    List<Etudiant> etudiants = groupeService.listerEtudiantsParGroupe(groupe.getId());

    try (FileWriter writer = new FileWriter(filePath)) {
        // Écriture de l'en-tête
        writer.append(csq:"INE,Nom,Prenom,Email\n");

        // Écriture des étudiants
        for (Etudiant etudiant : etudiants) {
            writer.append(etudiant.getLine()).append(csq:",")
                .append(etudiant.getNom()).append(csq:",")
                .append(etudiant.getPrenom()).append(csq:",")
                .append(etudiant.getEmail()).append(csq:"\n");
        }

        // Pop-up de réussite
        JOptionPane.showMessageDialog(parentComponent:null, "CSV généré avec succès : " + filePath, title:"Succès",
            JOptionPane.INFORMATION_MESSAGE);
    } catch (IOException e) {
        // Pop-up d'erreur en cas d'échec
        JOptionPane.showMessageDialog(parentComponent:null, "Erreur lors de l'écriture du fichier CSV : " + e.getMessage(),
            title:"Erreur", JOptionPane.ERROR_MESSAGE);
        throw new RuntimeException(message:"Erreur lors de l'écriture du fichier CSV", e);
    }
}

```

Figure 17:Méthode exportEtudiantsByGroupe

➤ Format PDF :

Liste des étudiants inscrits à une UE avec le logo de l'université, la date d'exportation et l'identité de l'exportateur.

V. IHM (Interface Homme-Machine)

➤ Choix de la formation :

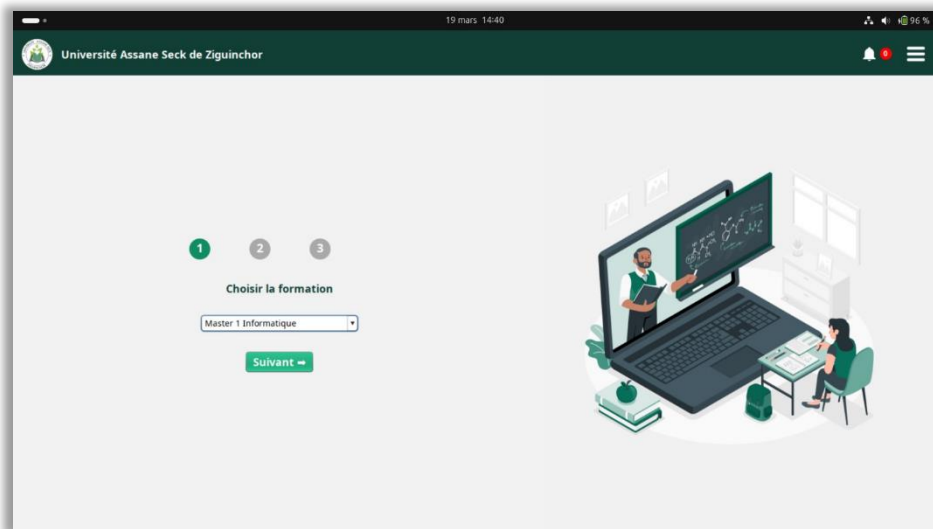


Figure 18: Interface d'inscription (Choix de la formation)

➤ Choix des UE optionnelles :

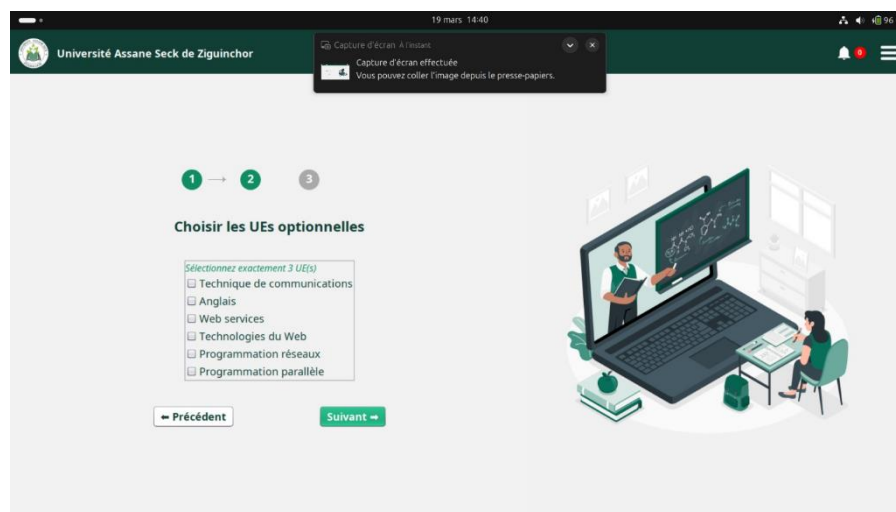


Figure 19: Interface d'inscription (Choix des UE optionnelles)

➤ **Confirmation d'inscription :**

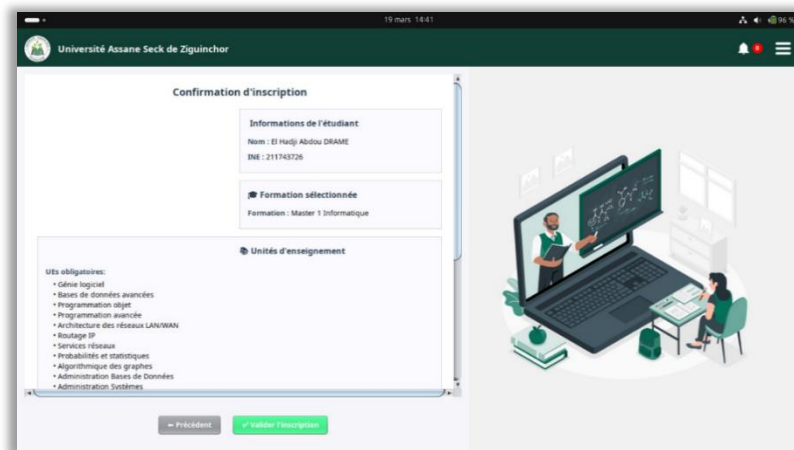


Figure 20: Interface d'inscription (Confirmation d'Inscription)

➤ **Interface du responsable pédagogique :**

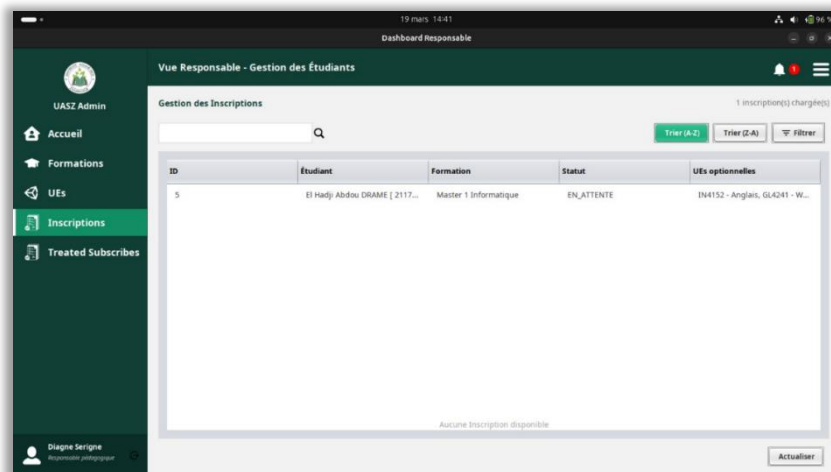


Figure 21: Interface du responsable pédagogique

➤ **Validation de l'inscription fait par le responsable :**

| | | | | |
|---|--------------------------------|-----------------------|----------|---------------------------------|
| 5 | El Hadji Abdou DRAME [2117... | Master 1 Informatique | ACCEPTEE | IN4152 - Anglais, GL4241 - W... |
|---|--------------------------------|-----------------------|----------|---------------------------------|

Figure 22: Validation d'une inscription

➤ **Courriel reçu par l'étudiant :**

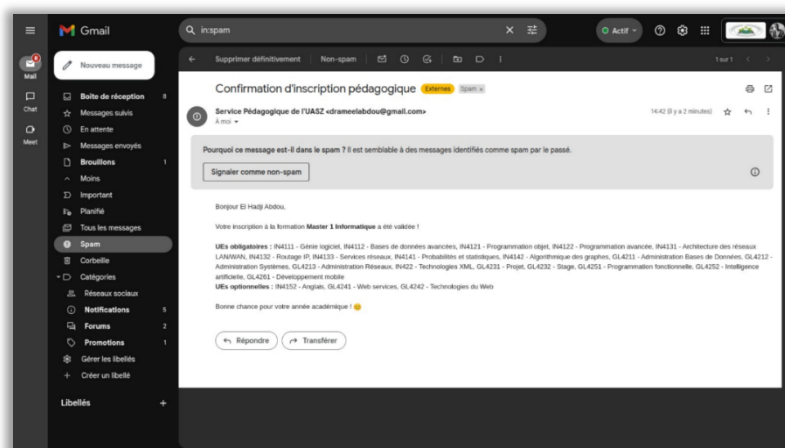


Figure 23: Email reçu par l'étudiant après validation
~ 14 ~

➤ Répartition des étudiants par groupe :



Figure 24: Répartition des étudiants par groupe

➤ Exportation des données en format PDF :

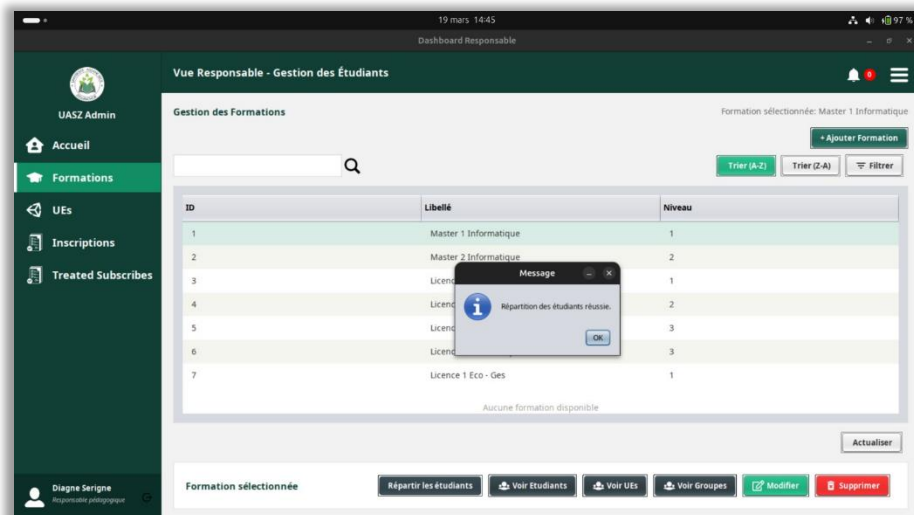


Figure 25: Exportation des données en format PDF

➤ Exemple : Exportation de la liste des étudiants inscrits dans l'UE : Anglais

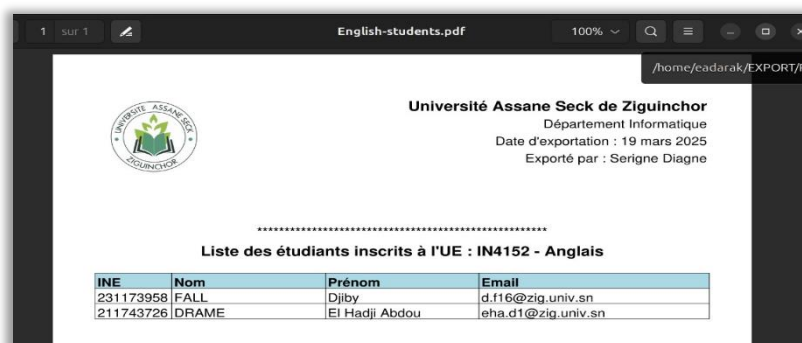


Figure 26: Exemple : English-Student.pdf

➤ **Exportation des données en format CSV :**

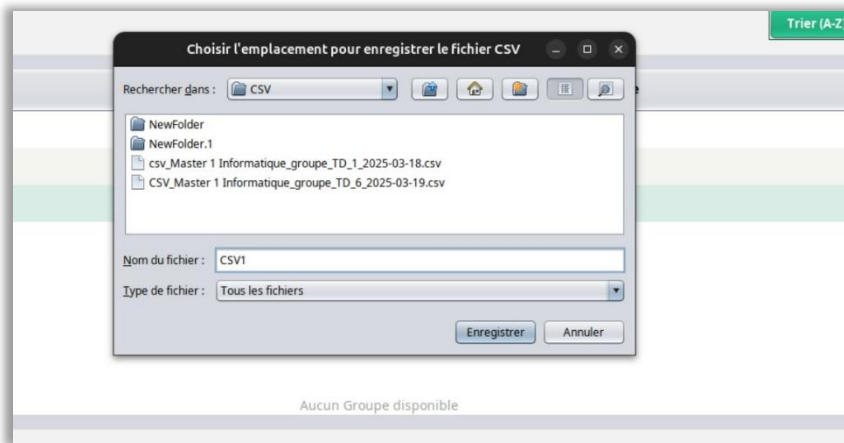


Figure 27:Exportation des données en format CSV

➤ **Exemple : Exportation des données en format csv d'un groupe d'étudiant**

| | A | B | C | D | E | |
|---|-----------|-------|----------------|--------------------|---|--|
| 1 | INE | Nom | Prenom | Email | | |
| 2 | 434841864 | FALL | Marame | m.f46@zig.univ.sn | | |
| 3 | 36316957 | DIA | Aimerou | a.d242@zig.univ.sn | | |
| 4 | 211743726 | DRAME | El Hadji Abdou | eha.d1@zig.univ.sn | | |
| 5 | | | | | | |
| 6 | | | | | | |
| 7 | | | | | | |

Figure 28:Exportation des données en format csv d'un groupe d'étudiant

Conclusion

Le développement de cette application de gestion des inscriptions pédagogiques a permis d'automatiser un processus essentiel dans les établissements d'enseignement supérieur. Grâce à une interface intuitive et des fonctionnalités bien définies, le système facilite l'inscription des étudiants, la validation par les responsables pédagogiques et la répartition automatique dans les groupes de TD et TP.

L'intégration de technologies comme Java, Swing, Hibernate et OpenPDF a permis d'assurer une solution robuste et évolutive. De plus, les fonctionnalités d'exportation en PDF et CSV garantissent une meilleure gestion et un suivi efficace des données.

Ce projet pourrait être amélioré en intégrant de nouvelles fonctionnalités, telles qu'une interface web pour une accessibilité accrue. Il constitue une base solide pour une gestion moderne et optimisée des inscriptions pédagogiques.