



Gestion des inscriptions pédagogiques

Application Java avec JPA et Swing

Rapport de Projet

Seydina Mouhamadou Al Hamine NDIAYE & Abdoulaye GAYE

M1 Informatique

20 mars 2025

Résumé

Ce rapport présente la conception et le développement d'une application de gestion des inscriptions pédagogiques pour les formations universitaires. Le projet implémente une solution complète permettant aux responsables pédagogiques de gérer les formations, les unités d'enseignement et les groupes, et aux étudiants de s'inscrire aux UE optionnelles. Développée en Java avec JPA pour la persistance des données et Swing pour l'interface utilisateur, l'application offre une gestion efficace du processus d'inscription pédagogique, y compris des fonctionnalités d'export de données et de notification par email.

Table des matières

1	Introduction	3
1.1	Contexte et objectifs	3
1.2	Méthodologie de travail	3
2	Architecture de l'application	3
2.1	Structure générale	3
2.2	Schéma de la base de données	4
2.3	Entités JPA	5
2.3.1	Entité Utilisateur	5
2.3.2	Entité Etudiant	6
2.3.3	Entité Enseignant	6
2.3.4	Entité ResponsablePedagogique	7
2.3.5	Entité Formation	8
2.3.6	Entité UE	9
2.3.7	Entité Groupe	10
2.3.8	Entité Etudiant	11
2.3.9	Entité UE	11
3	Fonctionnalités implémentées	11
3.1	Gestion des formations et UEs	11
3.2	Gestion des groupes	13
3.3	Inscription pédagogique des étudiants	13
3.4	Validation des inscriptions	16
3.5	Listes et statistiques	17
3.6	Fonctionnalités supplémentaires	18
3.6.1	Le hachage des mots de passe	18
3.6.2	Notification par email	20
3.6.3	Gestion des droits d'accès	20
4	Interface utilisateur	21
4.1	Conception de l'interface	21
5	Conclusion	22
5.1	Bilan du projet	22
5.2	Perspectives d'évolution	22
5.3	Compétences acquises	23
A	Guide d'utilisation	23
A.1	Installation et configuration	23
A.2	Compte de démonstration	24
B	Dépendances du projet	24
C	Bilan personnel	26

1 Introduction

1.1 Contexte et objectifs

La gestion des inscriptions pédagogiques est un processus critique pour toute institution d'enseignement supérieur. Elle permet d'organiser les enseignements, de gérer les effectifs et d'assurer un suivi administratif efficace des étudiants. Notre projet vise à développer une application complète répondant à ce besoin, en automatisant le processus d'inscription et en offrant une interface conviviale tant pour les responsables pédagogiques que pour les étudiants.

L'application permet de gérer l'ensemble du cycle d'inscription pédagogique, depuis la création des formations et des UEs jusqu'à la répartition des étudiants dans les groupes de TD et TP, en passant par la validation des choix d'UEs optionnelles. Elle intègre également des fonctionnalités avancées comme l'export de données et les notifications par email.

1.2 Méthodologie de travail

Pour mener à bien ce projet, nous avons adopté une approche agile centrée sur la collaboration et l'itération continue. Les principales étapes de notre démarche ont été les suivantes :

1. Analyse des besoins et des fonctionnalités requises
2. Conception du schéma de base de données
3. Création des entités JPA
4. Développement des couches d'accès aux données
5. Implémentation de la logique métier
6. Conception et développement de l'interface utilisateur

L'utilisation de Git et [github](#) nous a permis de collaborer efficacement et de suivre l'évolution du projet au fil du temps, tout en maintenant un historique complet des modifications.

2 Architecture de l'application

2.1 Structure générale

Notre application suit une architecture en couches classique, avec une séparation claire des responsabilités :

FIGURE 1 – Architecture générale de l'application

L'application est divisée en plusieurs packages principaux :

- **model** : Contient les entités JPA représentant les objets métier
- **dao** : Gère l'accès aux données et les interactions avec la base de données
- **dto** : Définit les objets de transfert de données pour la communication entre les couches

- **email** : Contient les classes responsables de l'envoi et de la gestion des emails
- **gui** : Regroupe les composants de l'interface utilisateur Swing
- **interface** : Définit les interfaces et contrats pour les différentes couches du système
- **interfacesEcouleur** : Implémente les écouteurs d'événements pour la gestion des interactions utilisateur
- **model** : Contient les objets de domaine et la logique métier principale
- **util** : Contient des classes utilitaires (gestion des emails, export de données, etc.)
- **service** : Implémente la logique métier de l'application

2.2 Schéma de la base de données

La base de données relationnelle est au cœur de notre application. Voici le schéma conceptuel que nous avons conçu :

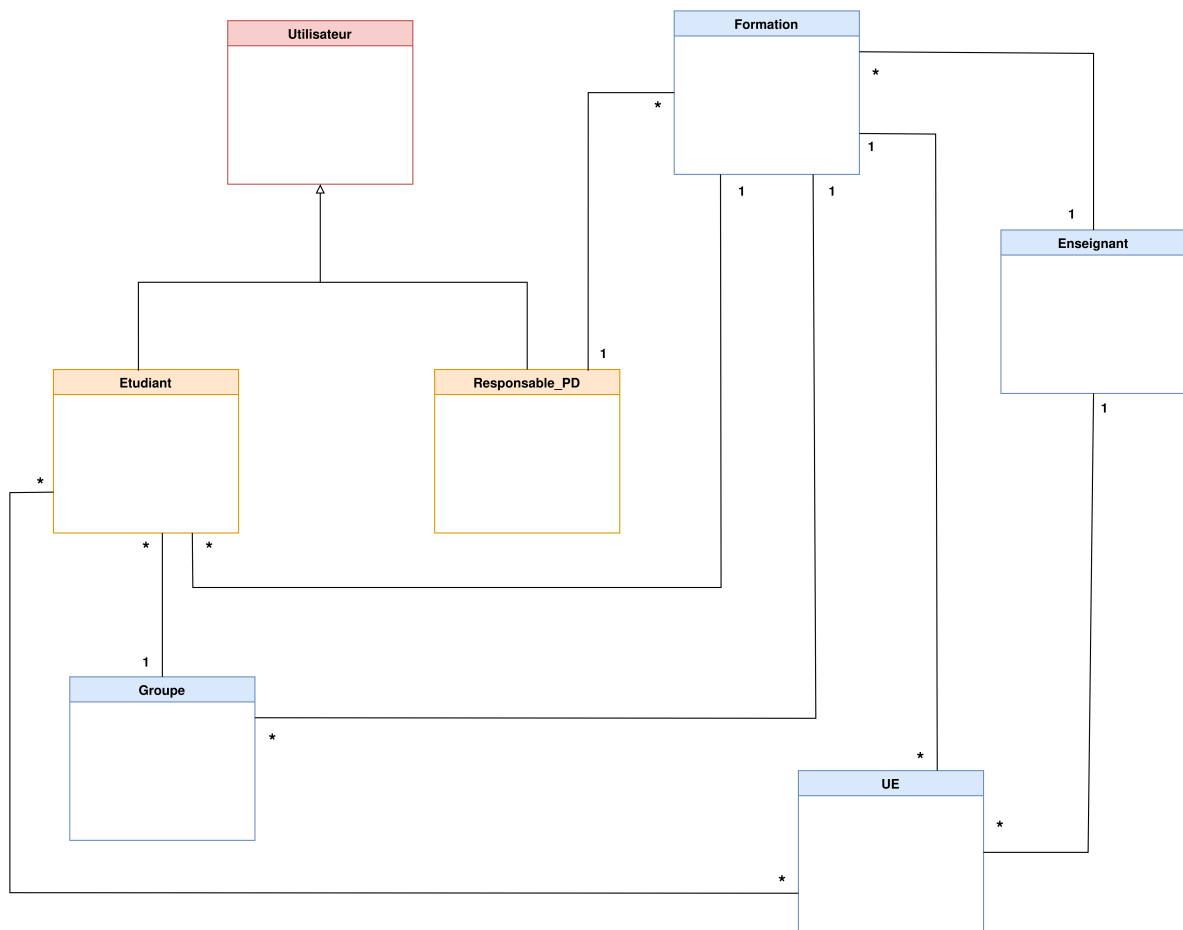


FIGURE 2 – Schéma de la base de données

Le schéma comprend les entités principales suivantes :

- **Utilisateur** : Stocke les informations d'authentification (entité parente)
- **Etudiant** : Étend Utilisateur et stocke les informations personnelles des étudiants
- **Enseignant** : Représente le responsable pour les formations et il a aussi une liste d'UE

- **Responsable_PD** : Étend Utilisateur (Responsable Pédagogique)
- **Formation** : Définit les formations disponibles et leurs caractéristiques
- **UE** : Représente les unités d'enseignement, avec leurs propriétés académiques
- **Groupe** : Gère les groupes de TD (Travaux Dirigés) et TP (Travaux Pratiques)

Les relations entre ces entités permettent de modéliser l'ensemble des fonctionnalités requises, avec des contraintes d'intégrité adaptées.

2.3 Entités JPA

Les entités JPA sont le reflet du schéma de base de données, avec les annotations nécessaires pour gérer les relations et les contraintes. Voici quelques exemples des principales entités :

2.3.1 Entité Utilisateur

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "role")
@Data
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "utilisateur")
public abstract class Utilisateur {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nom;
    private String prenom;
    private LocalDate dateNaissance;

    @Enumerated(EnumType.STRING)
    private Sexe sexe;

    private String adresse;

    @Column(unique = true)
    private String email;

    private String password;

    // Méthode pour récupérer le rôle depuis la colonne "role"
    public String getRole() {
        return this.getClass().getSimpleName();
    }
}
```

Listing 1 – Entité Utilisateur

2.3.2 Entité Etudiant

```

@EqualsAndHashCode(callSuper = true, exclude = {"ues", "groupe", "
    formation"}) // Exclure toutes les relations
@ToString(exclude = {"ues", "groupe", "formation"}) // viter les
    cycles dans toString aussi
@Data
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@DiscriminatorValue("ETUDIANT")
public class Etudiant extends Utilisateur {

    @Column(unique = true)
    private String ine;

    @ManyToOne
    @JoinColumn(name = "groupe_id")
    private Groupe groupe;

    // @ManyToMany(fetch = FetchType.LAZY)
    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name = "etudiant_ue", joinColumns = @JoinColumn(name =
        "etudiant_id"), inverseJoinColumns = @JoinColumn(name = "
        ue_id"))
    private Set<UE> ues = new HashSet<>();

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

    // @Default(false)
    private boolean inscriptionValidee;
}
}

```

Listing 2 – Entité Etudiant

2.3.3 Entité Enseignant

```

@Data
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "enseignant")
public class Enseignant {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
}

```

```

@Column(unique = true)
private String matricule;
private String nom;
private String prenom;
private String email;
private String telephone;

@OneToMany(mappedBy = "enseignant", cascade = CascadeType.ALL,
    fetch = FetchType.EAGER)
private Set<UE> ues = new HashSet<>();

@OneToMany(mappedBy = "responsableFormation", fetch = FetchType.
    EAGER)
private Set<Formation> formations = new HashSet<>();

@Override
public String toString() {
    return this.prenom + " " + this.nom;
}
}
}

```

Listing 3 – Entité Enseignant

2.3.4 Entité ResponsablePédagogique

```

@EqualsAndHashCode(callSuper = true)
@Data
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@DiscriminatorValue("ResponsablePédagogique")
public class ResponsablePédagogique extends Utilisateur {

    @OneToMany(mappedBy = "responsable", cascade = CascadeType.ALL,
        fetch = FetchType.EAGER)
    private Set<Formation> formations = new HashSet<>();

    // M thodes helper
    public void addFormation(Formation formation) {
        this.formations.add(formation);
        formation.setResponsable(this);
    }

    public void removeFormation(Formation formation) {
        this.formations.remove(formation);
        formation.setResponsable(null);
    }
}
}

```

Listing 4 – Entité ResponsablePedagogique

2.3.5 Entité Formation

```

@Data
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "formation")
public class Formation {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nom;
    @Column(unique = true)
    private String code;
    private Integer nombreOptionsRequis;
    private Integer maxEffectifGroupe;

    @Enumerated(EnumType.STRING)
    private Niveau niveau;

    @OneToMany(mappedBy = "formation", cascade = CascadeType.ALL,
        fetch = FetchType.EAGER)
    private Set<UE> ues = new HashSet<>();

    @OneToMany(mappedBy = "formation", cascade = CascadeType.ALL,
        fetch = FetchType.EAGER)
    private Set<Etudiant> etudiants = new HashSet<>();

    @OneToMany(mappedBy = "formation", cascade = CascadeType.ALL,
        fetch = FetchType.EAGER)
    private Set<Groupe> groupes = new HashSet<>();

    @ManyToOne
    @JoinColumn(name = "responsable_id")
    private ResponsablePedagogique responsable;
    @ManyToOne
    @JoinColumn(name = "responsableFormation_id")
    private Enseignant responsableFormation;

    // M thodes helper
    public void addUE(UE ue) {
        this.ues.add(ue);
        ue.setFormation(this);
    }

    public void removeUE(UE ue) {
        this.ues.remove(ue);
        ue.setFormation(null);
    }
}

```



```

    }

    @Override
    public boolean equals(Object o) {
        if (this == o)
            return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Formation formation = (Formation) o;
        return Objects.equals(id, formation.id);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id);
    }

    @Override
    public String toString() {
        return "Formation{" +
            "id=" + id +
            ", nom='" + nom + '\'' +
            ", responsablePedagogique='" +
            (responsable != null ? responsable.getNom() : "null") +
            '\'' +
            '}';
    }
}
}

```

Listing 5 – Entité Formation

2.3.6 Entité UE

```

@EqualsAndHashCode(exclude = { "etudiants" })
@Data
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "ue")
public class UE {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true)
    private String code;

    private String nom;
    private Integer volumeHoraire;
}

```

```

private Double coefficient;
private Integer credits;

@ManyToMany(mappedBy = "ues", fetch = FetchType.EAGER)
private Set<Etudiant> etudiants = new HashSet<>();

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

@ManyToOne
@JoinColumn(name = "enseignant_id")
private Enseignant enseignant;

private boolean obligatoire;

@Override
public boolean equals(Object o) {
    if (this == o)
        return true;
    if (o == null || getClass() != o.getClass())
        return false;
    UE ue = (UE) o;
    return Objects.equals(id, ue.id);
}

@Override
public int hashCode() {
    return Objects.hash(id);
}

@Override
public String toString() {
    return "UE{" +
        "id=" + id +
        ", nom='" + nom + '\'' +
        ", volumeHoraire=" + volumeHoraire +
        '}';
}
}
}

```

Listing 6 – Entité UE

2.3.7 Entité Groupe

```

@EqualsAndHashCode(of = {"id"})
@ToString(exclude = {"etudiants", "formation"})
@Data
@Getter
@Setter
@NoArgsConstructor

```

```
@AllArgsConstructor
@Entity
@Table(name = "groupe")
public class Groupe {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private Integer numero;

    @Enumerated(EnumType.STRING)
    private TypeGroupe typeGroupe;

    @OneToMany(mappedBy = "groupe", cascade = CascadeType.ALL)
    private Set<Etudiant> etudiants = new HashSet<>();

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

    // M thodes helper
    public void addEtudiant(Etudiant etudiant) {
        this.etudiants.add(etudiant);
        etudiant.setGroupe(this);
    }

    public void removeEtudiant(Etudiant etudiant) {
        this.etudiants.remove(etudiant);
        etudiant.setGroupe(null);
    }
}
```

Listing 7 – Entité Groupe

2.3.8 Entité Etudiant

2.3.9 Entité UE

3 Fonctionnalités implémentées

3.1 Gestion des formations et UEs

Le responsable pédagogique dispose d'un ensemble complet de fonctionnalités pour gérer les formations et les UEs :

Système de Gestion Pédagogique Mbaye Tamba (r) Déconnexion

Gérer les Formations

Rechercher: dans Code

Code	Intitulé	Niveau	Responsable	EmailResp	Nb_ue_optionnel	Max_Effectif_Groupe
L3INFO	Licence 3 Informatique	L3			2	3
M1INFO	Master 1 Informatique	M1			3	4
M2PHY	M2 Physique	M2			4	6
M1MAT	M1 Mathématiques	M1			1	4
M2ÉCO	M2 Économie	M2			2	3
L1PHY	L1 Physique	L1			4	5
L2MAT	L2 Mathématiques	L2			3	2

Détails de la formation

Code:

Intitulé:

Niveau:

nombre d'ue Optionnel:

max Effectif groupe:

Responsable:

Formation sélectionnée: L3INFO

FIGURE 3 – Interface de gestion des formations

- Création, modification et suppression de formations
- Ajout et retrait d'UEs dans une formation
- Définition du caractère obligatoire ou optionnel des UEs
- Verrouillage des UEs pour empêcher leur retrait

Le code suivant illustre la logique métier pour l'inscription d'un étudiant à des UEs optionnelles :

```
@Override
public boolean inscrireUEsOptionnelles(Long etudiantId, List<Long>
selectedUEIds) {
    try (EntityManager em = JPAUtil.getEntityManager();) {
        em.getTransaction().begin();

        // Récupérer l'étudiant depuis la base de données
        Etudiant etudiant = em.find(Etudiant.class, etudiantId);
        if (etudiant == null) {
            System.out.println("étudiant non trouvé.");
            return false;
        }

        // Récupérer les UEs sélectionnées
        TypedQuery<UE> query = em.createQuery(
            "SELECT ue FROM UE ue WHERE ue.id IN :ueIds AND"
            "ue.obligatoire = false", UE.class);
        query.setParameter("ueIds", selectedUEIds);
        List<UE> selectedUEs = query.getResultList();

        // Ajouter les UEs optionnelles à l'étudiant
        for (UE ue : selectedUEs) {
            etudiant.getUes().add(ue);
            ue.getEtudiants().add(etudiant);
        }
    }
}
```

```

    }

    em.merge(etudiant);
    em.getTransaction().commit();

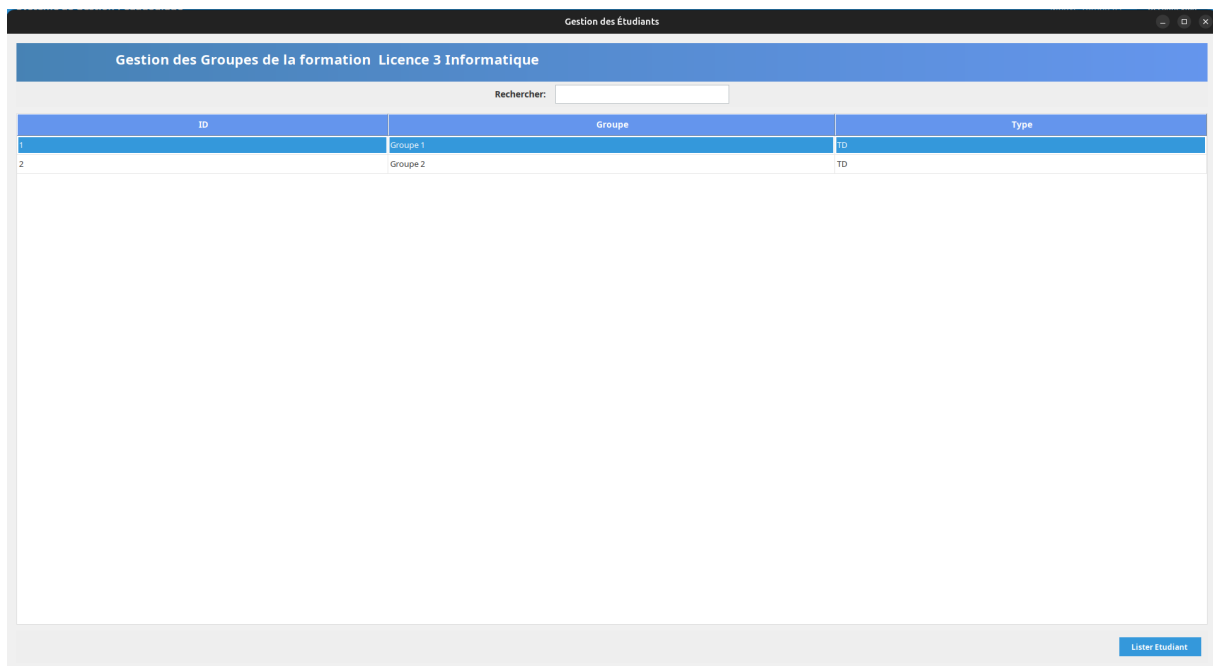
    return true;
} catch (Exception e) {
    System.err.println("Erreur lors de l'inscription des UEs
        optionnelles : " + e.getMessage());
    return false;
}
}

```

Listing 8 – Service de choix pour les UEs optionnelles d'une formation

3.2 Gestion des groupes

L'application permet une gestion efficace des groupes de TD et TP :



The screenshot shows a web application window titled "Gestion des Étudiants". Inside, there's a header "Gestion des Groupes de la formation Licence 3 Informatique". Below the header is a search bar labeled "Rechercher:". A table displays the following data:

ID	Groupe	Type
1	Groupe 1	TD
2	Groupe 2	TD

At the bottom right of the interface, there is a button labeled "Lister Etudiant".

FIGURE 4 – Interface de gestion des groupes

- Création automatique de groupes de TD et TP pour chaque formation lors de validation de l'inscription d'un étudiant
- Définition de la taille maximale des groupes
- Répartition automatique des étudiants dans les groupes
- Visualisation des effectifs par groupe

3.3 Inscription pédagogique des étudiants

Les étudiants peuvent effectuer leur inscription pédagogique via une interface dédiée :

The screenshot shows a web browser window titled "Inscription Étudiant - UASZ". The page has a blue header with a plus icon and the text "Inscription Nouvel Étudiant". Below the header, the form contains the following fields:

- Prénom :** Text input field containing "Abdoulaye".
- Nom :** Text input field containing "GAYE".
- Adresse :** Text input field containing "Khombole".
- Date de Naissance :** Date picker showing "October 13, 1999" with a calendar icon and a dropdown arrow.
- Sexe :** Radio buttons for "Homme" (selected) and "Femme".
- Formation :** Dropdown menu showing "Licence 3 Informatique".

At the bottom of the form, there are two buttons: "Annuler" (grey) and "Valider" (orange).

FIGURE 5 – Formulaire d'inscription

The screenshot displays a web application window titled "Inscription Étudiant - UASZ". The main heading is "Inscription Nouvel Étudiant". The form contains the following fields:

- Prénom :** Abdoulaye
- Date :** (empty)
- Formation :** Licence 3 Informatique

A modal dialog titled "Inscription Réussie" is overlaid on the form, displaying the following information:

- Inscription réussie !** (with a green checkmark icon)
- Email :** abdoulaye.gaye@etu.uasz.sn
- Mot de passe :** 928132
- OK** button

At the bottom of the application window, there are two buttons: "Annuler" (with a red X icon) and "Valider" (with a green checkmark icon).

FIGURE 6 – Identifiant Obtenue apres inscription

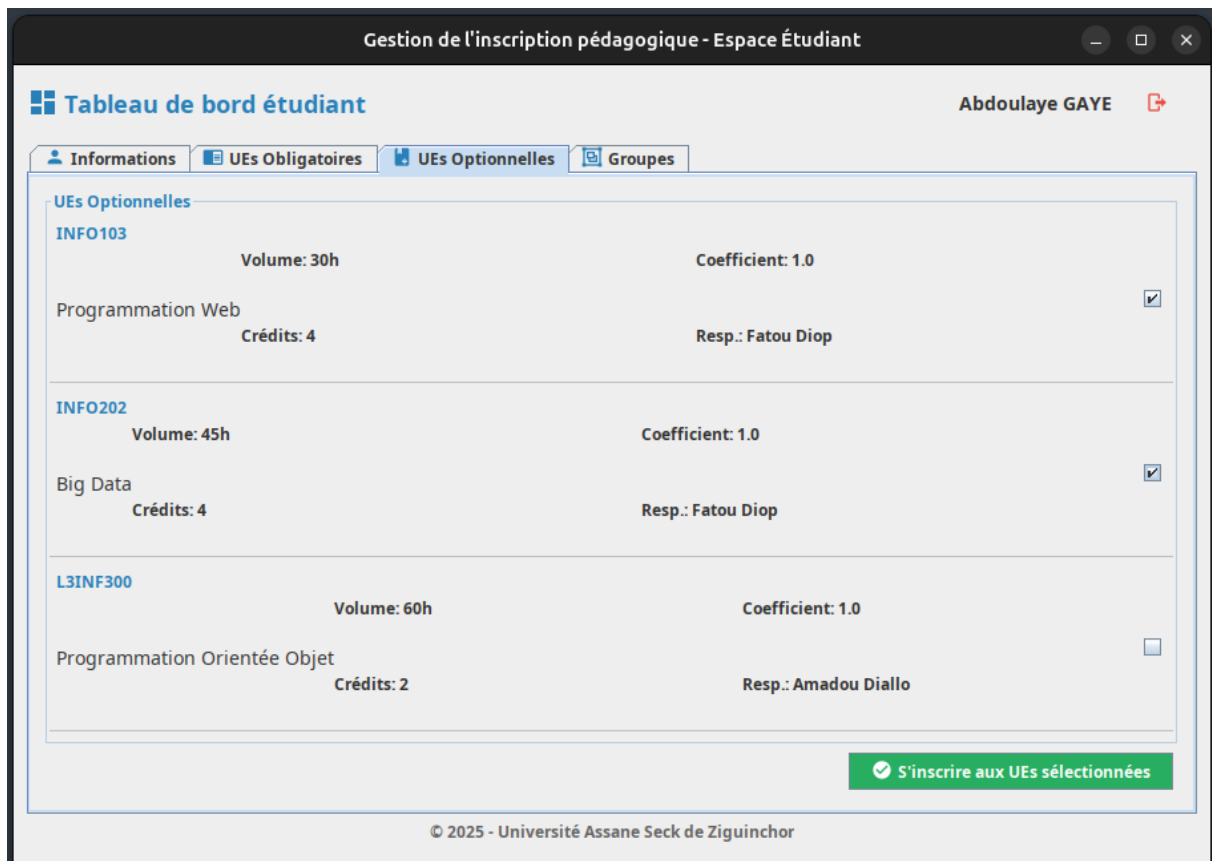


FIGURE 7 – Interface d’inscription pédagogique

Le processus d’inscription comprend les étapes suivantes :

1. Connexion avec les identifiants personnels obtenue lors de l’inscription
2. Sélection des UEs optionnelles parmi celles proposées
3. Soumission de la demande d’inscription
4. Attente de la validation par le responsable pédagogique
5. Réception d’une notification par email de confirmation

3.4 Validation des inscriptions

Le responsable pédagogique dispose d’une interface pour valider les inscriptions des étudiants :

Gestion des Étudiants de la formation Licence 3 Informatique							
Rechercher: <input type="text"/>							
INE	Prénom	Nom	Adresse	DateNaissance	Email	Groupe	Inscription
INE004	Mamadou	Faye	Saint-Louis	1999-12-10	mamadou.faye@example.co...	Non Affecté	● En AttenteX
INE009	Jannie	Goldner	Manyburgh	1996-04-28	jannie.goldner@example.com	Non Affecté	● ✓Validée
INE018	Emil	Bauch	Lake Billyside	2004-08-10	emil.bauch@example.com	Non Affecté	● ✓Validée
202540610	Abdoulaye	GAYE	Thionville	1999-10-13	abdoulaye.gaye@etu.uaoz.sn	Non Affecté	● En AttenteX
INE010	Tyrell	Kuhn	Hugoborough	2006-11-28	tyrell.kuhn@example.com	Non Affecté	● En AttenteX
INE012	Loyce	Wisock	Schowalterchester	2005-04-18	loyce.wisock@example.com	Non Affecté	● En AttenteX
INE019	Ming	Crooks	Donnellhaven	1996-04-27	ming.crooks@example.com	Non Affecté	● ✓Validée
INE022	Kacy	Grimes	North Taynaside	2001-03-26	kacy.grimes@example.com	Non Affecté	● ✓Validée
INE001	Fall	Ngor	Dakar	2000-05-15	e	1	● ✓Validée
INE005	Shala	Hammes	Port Cruzport	2003-01-30	shala.hammes@example.com	Non Affecté	● ✓Validée
INE006	Bella	O'Hara	Jacquelineville	1997-04-10	bella.o'hara@example.com	Non Affecté	● ✓Validée
INE013	Sueann	Wunsch	South Burtonview	2004-01-21	sueann.wunsch@example.c...	Non Affecté	● ✓Validée
INE023	Arlen	Walter	Labadiemouth	1999-04-11	arlen.walter@example.com	Non Affecté	● En AttenteX
INE024	Stevie	Hammes	Hintzside	1998-09-12	stevie.hammes@example.co...	1	● ✓Validée
INE008	Rueben	Klein	North Robertshire	2004-02-11	rueben.klein@example.com	Non Affecté	● ✓Validée
INE014	Florencia	Prosacco	Lake Anglaton	1997-05-22	florencia.prosacco@example...	Non Affecté	● En AttenteX
INE017	Asia	Schuster	East Joelberg	1997-07-15	asia.schuster@example.com	Non Affecté	● ✓Validée
INE021	Zane	Batz	South Caroleborough	2005-01-30	zane.batz@example.com	Non Affecté	● En AttenteX
INE007	Aubrey	Marquardt	Kymberlyview	2000-10-01	aubrey.marquardt@exampl...	Non Affecté	● ✓Validée
INE015	Latisha	Bednar	New Kamilahfurt	2003-12-21	latisha.bednar@example.com	Non Affecté	● ✓Validée
INE020	Rick	Bechtelar	Lake Josepberg	1999-01-16	rick.bechtelar@example.com	Non Affecté	● ✓Validée

FIGURE 8 – Interface de validation des inscriptions

Cette interface permet de :

- Visualiser les demandes d'inscription en attente
- Vérifier la conformité des choix d'UEs optionnelles
- Valider ou refuser les inscriptions
- Notifier automatiquement les étudiants par email

3.5 Listes et statistiques

L'application offre diverses fonctionnalités de reporting :

Code	Intitulé	Crédits	Coefficient	Volume Horaire	Enseignant	Caractéristique
INFO101	Algorithmique Avancée	5	1.5	90	Amadou Diallo	obligatoire
INFO102	Bases de Données	4	1.0	45	Fatou Diop	obligatoire
INFO103	Programmation Web	4	1.0	30	Fatou Diop	optionnel
INFO201	Intelligence Artificielle	6	1.5	60	Amadou Diallo	obligatoire
INFO202	Big Data	4	1.0	45	Fatou Diop	optionnel
L3INF300	Programmation Orientée Objet	2	1.0	60	Amadou Diallo	optionnel
L3INF307	DevOps	3	1.0	60	Amos Klein	obligatoire

FIGURE 9 – Interface de listes des UEs

Le responsable pédagogique peut générer :

- La liste des UEs d'une formation
- La liste des étudiants par groupe de TD et de TP

3.6 Fonctionnalités supplémentaires

En plus des fonctionnalités requises, nous avons implémenté plusieurs améliorations :

3.6.1 Le hachage des mots de passe

L'application permet d'acher les mots de passe claire en crypté. :

```
private void validerInscription() {
    // Validation des champs
    if (prenomField.getText().trim().isEmpty() ||
        nomField.getText().trim().isEmpty() ||
        adresseField.getText().trim().isEmpty() ||
        dateNaissancePicker.getDate() == null ||
        (!hommeButton.isSelected() && !femmeButton.isSelected()) ||
        formationBox.getSelectedItem() == null) {

        showErrorDialog("Veuillez remplir tous les champs obligatoires");
        return;
    }

    // Rcupération des données
    String prenom = prenomField.getText().trim();
    String nom = nomField.getText().trim();
    String adresse = adresseField.getText().trim();
    LocalDate dateNaissance = dateNaissancePicker.getDate();
}
```

```

Sexe sexe = hommeButton.isSelected() ? Sexe.MASCULIN : Sexe.
    FEMININ;
Formation formation = ((FormationWrapper)formationBox.
    getSelectedItem()).getFormation();

// G n r ation des identifiants
String email = genererEmail(prenom, nom);
String plainPassword = genererMotDePasse(); // Stockez le mot
    de passe en clair pour l'affichage
String hashedPassword = BCrypt.hashpw(plainPassword, BCrypt.
    gensalt()); // Cryptez le mot de passe

// Cr ation de l' tudiant
Etudiant etudiant = new Etudiant();
etudiant.setPrenom(prenom);
etudiant.setNom(nom);
etudiant.setSexe(sexe);
etudiant.setFormation(formation);
etudiant.setEmail(email);
// etudiant.setPassword(password);
etudiant.setPassword(hashedPassword); // Utilisez le mot de
    passe crypt
etudiant.setAdresse(adresse);
etudiant.setDateNaissance(dateNaissance);
etudiant.setIne(genererIne());
etudiant.setUes(formation.getUes() != null ? formation.getUes
    ().stream().filter(fnctn -> fnctn.isObligatoire()).collect
    (Collectors.toSet()) : Collections.emptySet());

// Persistence
try {
    em.getTransaction().begin();
    em.persist(etudiant);
    em.getTransaction().commit();

    // showSuccessDialogWithCopy(email, password);
    // Utilisez le mot de passe en clair pour l'affichage
    showSuccessDialogWithCopy(email, plainPassword);
    dispose();

} catch (Exception e) {
    e.printStackTrace();
    if (em.getTransaction().isActive()) {
        em.getTransaction().rollback();
    }
    showErrorDialog("Erreur lors de l'inscription: " + e.
        getMessage());
}
}

```

Listing 9 – Service de hachage

3.6.2 Notification par email

Un système de notification par email a été implémenté pour informer les étudiants de la validation de leur inscription :

```
@Override
    public void sendEmail(String toEmail, String subject, String body
    ) throws EmailException {
        try {
            // Configuration SMTP
            Properties props = new Properties();
            props.put("mail.smtp.auth", "true");
            props.put("mail.smtp.starttls.enable", "true");
            props.put("mail.smtp.host", smtpHost);
            props.put("mail.smtp.port", smtpPort);

            // Cr ation de la session
            Session session = Session.getInstance(props, new
                Authenticator() {
                    @Override
                    protected PasswordAuthentication
                        getPasswordAuthentication() {
                            return new PasswordAuthentication(username,
                                password);
                        }
                });

            // Cr ation du message
            Message message = new MimeMessage(session);
            message.setFrom(new InternetAddress(username));
            message.setRecipients(Message.RecipientType.TO,
                InternetAddress.parse(toEmail));
            message.setSubject(subject);

            // Configuration du contenu HTML
            message.setContent(body, "text/html; charset=utf-8");

            // Envoi du message
            Transport.send(message);

            System.out.println("\n###Email sent successfully to " +
                toEmail + "\n");

        } catch (MessagingException e) {
            throw new EmailException("Erreur lors de l'envoi de l'
                email", e);
        }
    }
```

Listing 10 – Service d'envoi d'email

3.6.3 Gestion des droits d'accès

Un système de gestion des droits d'accès a été mis en place pour sécuriser l'application :



FIGURE 10 – Interface d’authentification

Ce système permet de :

- Différencier les rôles (responsable pédagogique, étudiant)
- Restreindre l’accès aux fonctionnalités en fonction du rôle
- Sécuriser les données sensibles

4 Interface utilisateur

4.1 Conception de l’interface

L’interface utilisateur a été conçue pour être intuitive et fonctionnelle, en suivant les principes de l’ergonomie logicielle :

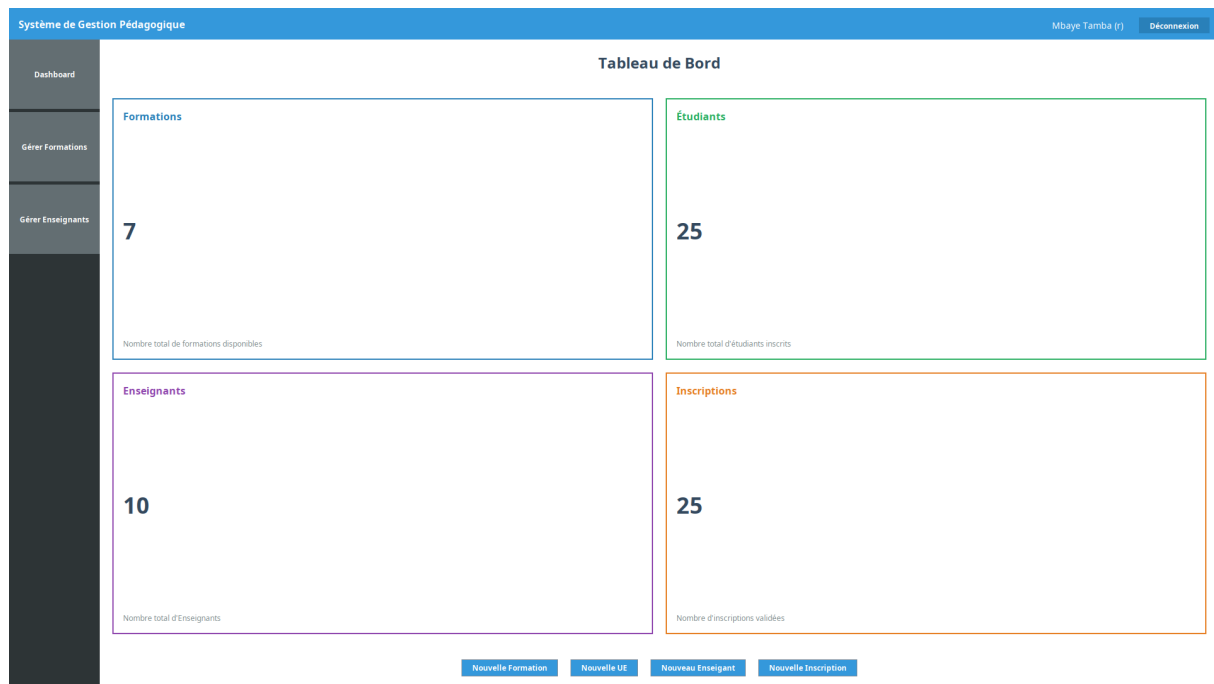


FIGURE 11 – Interface principale de l’application

Nous avons utilisé la bibliothèque Swing pour développer l’interface, avec une attention particulière portée à :

- L’organisation logique des fonctionnalités
- La cohérence visuelle entre les différents écrans
- La facilité de navigation
- Les retours visuels sur les actions effectuées
- L’accessibilité des fonctionnalités fréquemment utilisées

5 Conclusion

5.1 Bilan du projet

Ce projet nous a permis de développer une application complète et fonctionnelle pour la gestion des inscriptions pédagogiques. Nous avons réussi à implémenter l’ensemble des fonctionnalités requises, tout en y ajoutant plusieurs améliorations significatives comme l’export de données et les notifications par email.

L’utilisation de JPA pour la persistance des données et de Swing pour l’interface utilisateur s’est avérée être un choix pertinent, offrant à la fois flexibilité et robustesse. La structure modulaire de l’application facilite sa maintenance et son évolution future.

Les principaux défis rencontrés ont concerné :

- La modélisation des relations entre les entités
- La gestion des transactions et de la concurrence

5.2 Perspectives d’évolution

Plusieurs pistes d’amélioration pourraient être envisagées pour les versions futures de l’application :

1. Développement d'une interface web pour compléter l'application desktop
2. Implémentation d'un système de gestion des emplois du temps
3. Intégration avec d'autres systèmes d'information de l'établissement
4. Ajout de fonctionnalités statistiques avancées
5. Mise en place d'un système de notification push
6. Développement d'applications mobiles pour les étudiants

5.3 Compétences acquises

Ce projet nous a permis de renforcer nos compétences dans plusieurs domaines :

- Conception et modélisation de bases de données relationnelles
- Programmation Java et utilisation du framework JPA
- Développement d'interfaces graphiques avec Swing
- Gestion de projet collaboratif avec Git et Github
- Implémentation de fonctionnalités avancées (export PDF/CSV, emails)
- Tests et debugging d'applications complexes

En conclusion, ce projet constitue une base solide pour le développement d'applications de gestion académique plus avancées et démontre notre capacité à concevoir et implémenter des solutions logicielles complètes répondant à des besoins métiers réels.

A Guide d'utilisation

A.1 Installation et configuration

Pour installer et exécuter l'application, suivez les étapes ci-dessous :

1. Assurez-vous que Java 17 ou supérieur est installé sur votre système
2. Configurez une base de données MariaDB avec les paramètres suivants :
 - Base de données : gestion_inscription_pedagogique
 - Utilisateur : m1Informatique
 - Mot de passe : m1Informatique
3. Ajouter la classe email.properties dans le package resources et ajouter le smtp.

```
smtp.host=smtp.gmail.com
smtp.port=587
smtp.username=example@gmail.com
smtp.password=mot_de_passe
```

4. Exécutez la methode main() de la classe Gestion_Inscription_Initiator qui dans le package utils
5. Lancez l'application à l'aide du fichier JAR fourni :

```
java -jar gestion-inscriptions.jar
```

A.2 Compte de démonstration

Pour tester l'application, vous pouvez utiliser les comptes suivants :

— **Responsable pédagogique :**

- Login : r
- Mot de passe : r

— **Étudiant :**

- Login : e
- Mot de passe : e

B Dépendances du projet

Voici les principales dépendances utilisées dans le projet :

- **Hibernate** : Framework ORM pour la persistance des données
- **MariaDB Java Client** : Pilote JDBC pour la connexion à MariaDB
- **iText** : Bibliothèque pour la génération de fichiers PDF
- **JavaMail** : API pour l'envoi d'emails
- **Lombok** : Bibliothèque pour réduire la verbosité du code
- **Ikonli** : Bibliothèque pour l'affichage d'icônes dans Swing
- **LGoodDatePicker** : Composant pour la sélection de dates en Java Swing
- **BCrypt** : Bibliothèque pour le hachage des mots de passe
- **JavaFaker** : Bibliothèque pour la génération de fausses données
- **JUnit** : Framework de test unitaire
- **Spring Context Support** : Module de Spring pour l'injection de dépendances

```
<dependencies>

  <dependency>
    <groupId>org.openrewrite.tools</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.37</version>
  </dependency>

  <dependency>
    <groupId>org.kordamp.ikonli</groupId>
    <artifactId>ikonli-swing</artifactId>
    <version>12.3.1</version>
  </dependency>

  <dependency>
    <groupId>org.kordamp.ikonli</groupId>
    <artifactId>ikonli-materialdesign-pack</artifactId>
    <version>12.3.1</version>
  </dependency>

  <dependency>
    <groupId>com.itextpdf</groupId>
    <artifactId>itextpdf</artifactId>
```



```

    <!-- <version>5.5.13.2</version> -->
    <version>5.5.13</version>
</dependency>
<dependency>
    <groupId>org.eclipse.angus</groupId>
    <artifactId>angus-mail</artifactId>
    <version>2.0.2</version>
</dependency>

<dependency>
    <groupId>org.kordamp.ikonli</groupId>
    <artifactId>ikonli-core</artifactId>
    <version>12.3.1</version>
</dependency>

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>7.0.0.Alpha1</version>
</dependency>

<dependency>
    <groupId>com.github.lgooddatepicker</groupId>
    <artifactId>LGoodDatePicker</artifactId>
    <version>11.2.1</version> <!-- V rifiez la derni re version
    -->
</dependency>

<!-- Java Mail API -->
<dependency>
    <groupId>com.sun.mail</groupId>
    <artifactId>jakarta.mail</artifactId>
    <version>2.0.1</version>
</dependency>

<!-- Pour l'injection de dependances (optionnel) -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>6.0.11</version>
</dependency>

<!-- Pour le hachage des mots de passe -->
<dependency>
    <groupId>org.mindrot</groupId>
    <artifactId>jbcrypt</artifactId>
    <version>0.4</version>
</dependency>

<!-- Pour utilisation de donne -->
<dependency>
    <groupId>com.github.javafaker</groupId>
    <artifactId>javafaker</artifactId>
    <version>1.0.2</version>
</dependency>

<dependency>
    <groupId>org.mariadb.jdbc</groupId>

```

```
<artifactId>mariadb-java-client</artifactId>
<version>3.5.2</version>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
</dependencies>
```

Listing 11 – Configuration des dépendances dans pom.xml

C Bilan personnel

Ce projet a été une expérience enrichissante pour notre équipe, nous permettant de mettre en pratique nos connaissances théoriques dans un contexte réel de développement logiciel. Nous avons particulièrement apprécié :

- La possibilité de travailler sur un projet complet, de la conception à l'implémentation
- L'expérience acquise dans l'utilisation des technologies JPA et Swing
- La collaboration en binôme, qui a favorisé l'échange d'idées et la résolution de problèmes
- Le développement de fonctionnalités avancées qui répondent à des besoins concrets

Les divergences rencontrées, notamment dans la gestion des relations entre entités et la conception d'une interface ergonomique, ont été surmontées grâce à une approche méthodique et à la recherche de solutions adaptées.

Cette expérience nous a également permis de mieux comprendre les enjeux de la gestion académique et de développer une application qui pourrait être utilisée dans un contexte réel d'enseignement supérieur.