

Gestion des Étudiants

Java desktop

Mini projet

Réalisé par :
Mohammed Hdillou

Etablissement : lycée technique ibn Sina Kenitra
[BTS]

Filière : développement des systèmes d'information
DSI

Année universitaire : 2024/2025

Introduction Générale

Dans un contexte où la **transformation numérique** s'impose comme un levier stratégique de modernisation, les établissements d'enseignement sont appelés à adopter des solutions technologiques capables de répondre aux exigences croissantes en matière de **gestion de l'information**. Les flux d'informations relatifs aux étudiants, allant de l'inscription aux suivis pédagogiques, représentent une charge importante en termes de temps, de ressources et d'organisation lorsqu'ils sont traités de manière traditionnelle.

En effet, la **gestion manuelle** des données étudiantes — à travers des formulaires papier, des fichiers Excel, ou d'autres outils non centralisés — présente de nombreuses limites. Elle engendre souvent des **redondances**, des **erreurs de saisie**, une **perte de temps considérable**, et une **difficulté d'accès rapide** à l'information. De plus, l'archivage et la mise à jour de ces données peuvent s'avérer fastidieux, notamment dans les structures accueillant un grand nombre d'étudiants. Ces méthodes, bien que toujours en usage dans certaines structures, ne permettent plus de répondre aux standards actuels de **réactivité**, de **sécurité**, et de **traçabilité** des données.

C'est dans cette perspective que la **numérisation des processus administratifs** prend tout son sens. En optant pour une **solution numérique centralisée**, les établissements peuvent non seulement garantir l'unicité et la cohérence des informations, mais également automatiser un grand nombre de tâches répétitives, facilitant ainsi le travail du personnel administratif et réduisant les risques d'erreurs humaines. Une telle solution permet également une **meilleure accessibilité aux données**, une **consultation rapide** des dossiers étudiants, ainsi qu'un **gain de productivité** notable.

Contents

I. Problématique	6
1. Manque de centralisation des données :	6
2. Risque élevé d'erreurs ou de doublons :	6
3. Sécurité insuffisante des données :	6
4. Difficulté d'accès et de recherche de l'information :	6
II. Spécifications Fonctionnelles et Techniques de l'Application	7
1. Exigences Fonctionnelles	7
1. Ajout d'un étudiant	7
2. Modification des données d'un étudiant	7
3. Suppression d'un étudiant	7
4. Consultation de la liste des étudiants	7
5. Impression de la liste des étudiants	7
2. Exigences Non Fonctionnelles	8
III. Analyse et conception	9
1. Présentation RUP	9
2. Diagramme de cas d'utilisation	10
a. Identification des acteurs :	10
b. Identification des cas d'utilisation :	10
3. Diagramme de class de conception	11
1. Identification des classes principales	11
2. Attributs des classes	11
3. Relations entre les classes	12
UC1 : Authentification	13
1. Diagramme de séquence système de UC1	13
2. Diagramme de séquence <i>détaillé</i> de UC1	15
UC2 : Consulter la liste des Etudiants	15
1. Diagramme de séquence système de UC2	15
2. Diagramme de séquence <i>détaillé</i> de UC2	17
3. Diagramme de classe participantes de UC2	18
UC3 : Mettre à jour la liste des Etudiants	18
1. Diagramme de séquence système de UC3	18
.....	22
4. Diagramme de classe participants de UC3	22

5. Modèle Physique de Données (MPD)	23
1. Présentation du Modèle	23
2. Modèle Physique de Données (MPD)(schéma) :	23
<i>Figure 15 : Modèle Physique de Données (MPD)(schéma)</i>	23
IV. IV Tests et Validation	24
1. Introduction	24
2. Objectifs des tests	24
3. Technologies utilisées	24
4. Configuration Maven	25
1. Configuration <i>Junit</i> :	25
2. Configuration JaCoCo :	26
3. Configuration SonarLite:	26
5. Tests des Classes	27
1. Exemple de Test « Class Etudiant »	27
2. Exemple de test class « AdminDAO »	29
V. Présentation des Interfaces Graphiques de l'Application	30
a. Interface de connexion (Login)	30
b. L'Interface d'Accueil Administrateur	31
c. L'Interface de Consulter la liste des Etudiants	32
d. L'Interface de Gestion des Étudiants	32
Conclusion Générale	34

Références

- ✓ **Java SE – Oracle Documentation**
Langage de programmation orienté objet utilisé pour développer la logique métier de l'application.
<https://docs.oracle.com/javase/>
- ✓ **Java Swing – Oracle GUI Toolkit**
Bibliothèque Java pour créer des interfaces graphiques utilisateur (GUI).
<https://docs.oracle.com/javase/tutorial/uiswing/>
- ✓ **Apache Maven – Build Automation Tool**
Outil de gestion de projet et d'automatisation de compilation, utilisé pour gérer les dépendances et le cycle de vie du projet.
<https://maven.apache.org/guides/index.html>
- ✓ **JUnit 5 – Unit Testing Framework**
Framework de test unitaire pour Java, permettant de vérifier le bon fonctionnement du code.
<https://junit.org/junit5/docs/current/user-guide/>
- ✓ **JaCoCo – Java Code Coverage Library**
Outil permettant de mesurer le taux de couverture des tests dans le code source Java.
<https://www.jacoco.org/jacoco/>
- ✓ **MySQL – Relational Database Management System**
Système de gestion de base de données relationnelle utilisé pour stocker et gérer les informations des étudiants.
<https://dev.mysql.com/doc/>
- ✓ **Stack Overflow – Communauté de développeurs**
Forum technique consulté pour résoudre certains problèmes liés au développement et aux tests.
<https://stackoverflow.com>
- ✓ **Visual Paradigm – Modélisation UML**
Outil de modélisation utilisé pour concevoir les diagrammes UML comme le diagramme de cas d'utilisation, le diagramme de classes, etc.
<https://www.visual-paradigm.com/>

I. Problématique

La **gestion des données étudiantes** dans les établissements scolaires et universitaires demeure un **enjeu complexe** et souvent problématique, en particulier lorsqu'elle repose sur des méthodes manuelles ou des systèmes hétérogènes et non intégrés. Plusieurs difficultés majeures freinent l'efficacité de ces processus, impactant la qualité du service administratif et la prise de décision pédagogique.

1. **Manque de centralisation des données :**
Dans de nombreux établissements, les informations relatives aux étudiants (identité, parcours scolaire, résultats, présence, etc.) sont réparties entre plusieurs services ou stockées sur des supports variés : feuilles papier, tableurs, logiciels disparates ou bases de données non synchronisées. Ce morcellement rend la gestion des dossiers fastidieuse, augmente les délais de traitement et multiplie les risques de perte ou d'incohérence des informations. L'absence d'un **système centralisé** empêche également une vue d'ensemble rapide et fiable de la situation d'un étudiant ou d'un groupe d'étudiants.
2. **Risque élevé d'erreurs ou de doublons :**
En l'absence de **contrôles automatisés** et de règles d'unicité des données, les processus de saisie manuelle sont propices aux fautes de frappe, à la redondance d'informations ou à des contradictions internes (mêmes étudiants enregistrés sous des noms légèrement différents, dates erronées, champs incomplets, etc.). Ces erreurs peuvent avoir des conséquences importantes, comme une mauvaise orientation, des retards dans la remise des documents officiels, ou des problèmes lors des démarches administratives externes (inscriptions aux examens, demandes de bourse, etc.).
3. **Sécurité insuffisante des données :**
Les informations détenues par les établissements éducatifs sont souvent **sensibles et confidentielles** : noms, adresses, numéros d'identification, résultats scolaires, dossiers médicaux, etc. Lorsqu'elles sont conservées sur des supports non sécurisés ou accessibles sans contrôle rigoureux, elles peuvent faire l'objet d'un **accès non autorisé**, voire d'une **fuite de données**. Cette vulnérabilité constitue un **risque majeur**, à la fois pour la vie privée des étudiants et pour la responsabilité juridique de l'établissement. Le respect des réglementations sur la protection des données personnelles (comme le RGPD) impose donc la mise en place de mécanismes de sécurité efficaces : authentification, chiffrement, traçabilité des accès, etc.
4. **Difficulté d'accès et de recherche de l'information :**
Enfin, l'absence d'une **interface intuitive** ou d'un moteur de recherche performant complique considérablement la consultation, la mise à jour et l'analyse des données. Les agents administratifs ou enseignants peuvent perdre un temps précieux à naviguer entre différents fichiers ou à effectuer des recherches manuelles fastidieuses. Cela impacte directement leur productivité et la qualité du service rendu aux étudiants. Un système d'information mal conçu devient alors un frein plutôt qu'un appui à la mission éducative.

II. Spécifications Fonctionnelles et Techniques de l'Application

1. Exigences Fonctionnelles

L'application de gestion des étudiants devra offrir un ensemble de fonctionnalités permettant de réaliser les opérations courantes de manière simple, sécurisée et efficace. Les principales exigences fonctionnelles sont détaillées ci-dessous :

1. Ajout d'un étudiant

L'utilisateur disposera d'un formulaire de saisie pour enregistrer un nouvel étudiant. Les champs suivants seront obligatoires :

- Nom
- Prénom
- Date de naissance
- Email
- Branche
- Classe

2. Modification des données d'un étudiant

L'utilisateur pourra rechercher un étudiant à partir d'un tableau ou d'un champ de recherche, puis accéder à ses informations afin de les modifier.

Après modification, un message de confirmation s'affichera pour indiquer que les changements ont bien été enregistrés.

3. Suppression d'un étudiant

Une fonctionnalité permettra de supprimer un étudiant de la base de données. Afin d'éviter toute suppression accidentelle, une demande de confirmation explicite s'affichera avant la validation de l'action.

4. Consultation de la liste des étudiants

Un tableau dynamique affichera la liste complète des étudiants, avec leurs principales informations (nom, prénom, classe, email, etc.).

Ce tableau intégrera les fonctionnalités suivantes :

- Filtrage par nom de l'étudiant
- Recherche en temps réel

5. Impression de la liste des étudiants

L'application devra permettre à l'utilisateur d'imprimer la liste des étudiants ainsi que leurs informations détaillées.

Les options suivantes seront disponibles :

- Impression complète de la liste
- Impression filtrée (selon la classe, la branche, etc.)
- Prévisualisation avant impression

2. Exigences Non Fonctionnelles

Outre les fonctionnalités principales que doit remplir l'application, il est primordial de prendre en compte un ensemble d'exigences non fonctionnelles. Celles-ci déterminent les qualités globales du système, influençant directement l'expérience utilisateur, la maintenabilité, et la pérennité du projet. Ces exigences visent à garantir un haut niveau de performance, de sécurité et de confort d'utilisation, tout en assurant que l'application puisse évoluer dans le temps selon les besoins.

❖ Fiabilité

L'application doit fonctionner de manière stable et continue, sans interruptions majeures ni plantages fréquents. Elle doit être conçue pour résister aux erreurs inattendues et permettre la reprise rapide des activités en cas de problème. Une attention particulière sera portée à la gestion des exceptions, à la sauvegarde automatique des données critiques, ainsi qu'à la robustesse générale du code.

❖ Performance

Le système doit fournir des temps de réponse rapides aux actions de l'utilisateur, même en présence d'un grand nombre d'enregistrements (par exemple, des milliers d'étudiants). Les opérations telles que la recherche, le filtrage ou l'affichage de la liste des étudiants doivent être optimisées afin d'éviter les ralentissements. Cela implique une optimisation des requêtes à la base de données, une gestion efficace de la mémoire et l'utilisation de structures de données performantes.

❖ Sécurité

La sécurité est un élément central du système. Toutes les données sensibles, telles que les mots de passe, seront cryptées avant d'être stockées. De plus, des droits d'accès différenciés seront appliqués selon les rôles des utilisateurs (administrateur, gestionnaire, etc.). Une authentification sécurisée, une gestion des sessions, et des mécanismes de protection contre les accès non autorisés (comme les injections SQL ou les attaques XSS) seront mis en place pour garantir l'intégrité et la confidentialité des données.

❖ Ergonomie

L'interface utilisateur sera conçue pour être intuitive, claire et conviviale, même pour les utilisateurs n'ayant pas de compétences techniques avancées. Les formulaires de saisie seront simples à utiliser, les messages d'erreur explicites, et la navigation fluide. Le design sera également responsive, afin de s'adapter aux différents types d'écrans (ordinateurs de bureau, tablettes, etc.), et favoriser une prise en main rapide de l'outil.

❖ Évolutivité

Le système doit être conçu de manière modulaire afin de permettre son extension à d'autres entités ou fonctionnalités dans le futur. Par exemple, il doit être possible d'ajouter ultérieurement la gestion des enseignants, des modules, des absences ou des notes, sans nécessiter de refonte complète. Cette évolutivité repose sur une architecture logicielle bien pensée, documentée, et conforme aux standards de développement modernes.

III. Analyse et conception

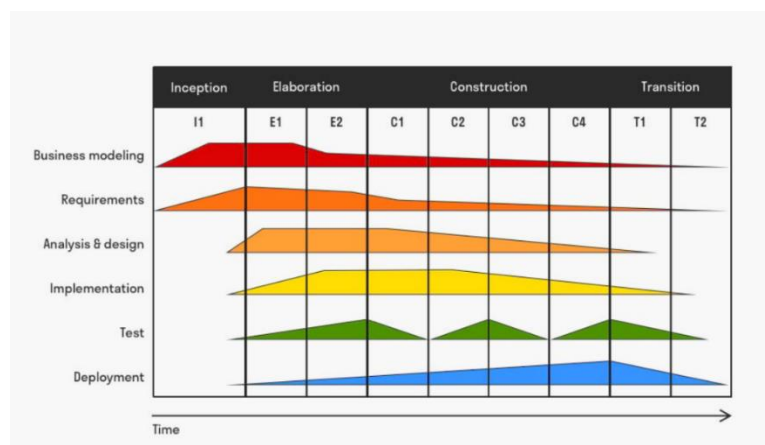
1. Présentation RUP

Le **Rational Unified Process (RUP)** est une méthode de développement logiciel qui repose sur un ensemble de pratiques, d'outils et de processus permettant de gérer les différentes étapes du cycle de vie d'un projet. Développée par **Rational Software**, cette méthode s'inspire de bonnes pratiques de gestion de projets agiles tout en apportant une structure et une discipline dans le développement.

RUP se base sur quatre phases principales :

1. **Inception (Conception initiale)** : Cette phase consiste à définir les objectifs du projet, ses besoins et les contraintes techniques. C'est ici qu'on effectue une analyse de faisabilité du projet.
2. **Elaboration (Élaboration)** : Durant cette phase, on affine les besoins et la solution technique, en clarifiant l'architecture du système et en identifiant les risques potentiels.
3. **Construction (Construction)** : Cette phase est dédiée à la réalisation du système, à l'intégration des différentes fonctionnalités et à la validation des résultats.
4. **Transition (Transition)** : L'objectif de cette phase est de mettre le système en production. Elle inclut la formation des utilisateurs et le support technique.

RUP se distingue par sa **flexibilité**, permettant aux équipes de projet de s'adapter aux exigences spécifiques tout en suivant un processus rigoureux. Il favorise la révision continue du produit tout au long du cycle de vie, ce qui garantit une meilleure gestion des risques et une meilleure qualité du logiciel.



2. Diagramme de cas d'utilisation

Les diagrammes de cas d'utilisation décrivent les fonctions générales et la portée d'un système. Ces diagrammes identifient également les interactions entre le système et ses acteurs.

a. Identification des acteurs :

- **Administrateur :**

Description : L'administrateur est un utilisateur principal du système, avec les droits les plus élevés. Il peut gérer tous les aspects de l'application, y compris la gestion des utilisateurs et des données.

b. Identification des cas d'utilisation :

A partir des exigences précédentes nous pouvons tirer les cas d'utilisation suivants :

- **UC1 : Authentification**
- **UC2 : Consulter la liste des Etudiants**
- **UC3 : Mettre à jour la liste des Etudiants**

- **Filière**
 - ID_Filière : identifiant
 - Nom_Filière
- **Administrateur**
 - ID_Admin
 - Nom, Email
 - MotDePasse

3. Relations entre les classes

- **Un étudiant est inscrit dans une seule classe**, mais une classe peut avoir plusieurs étudiants.
- **Une classe appartient à une seule filière**, et une filière peut contenir plusieurs classes.

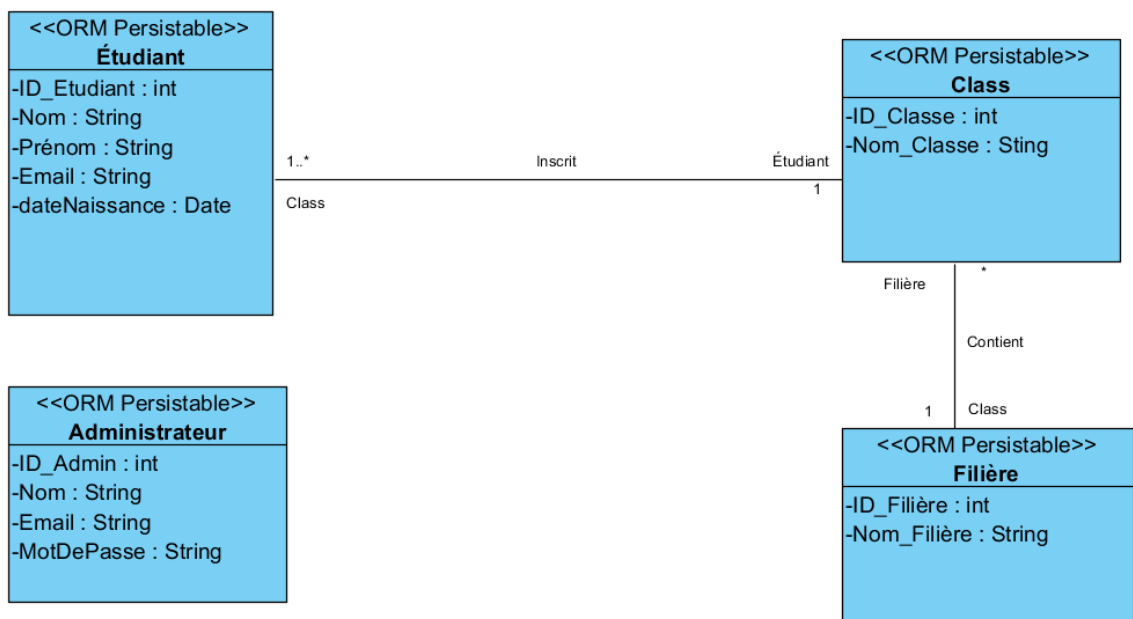


Figure2 : diagramme de class de conception

UC1 : Authentification

1. Diagramme de séquence système de UC1

❖ *Cas d'utilisation : Authentification*

Acteur principal : Utilisateur (Administrateur)

Précondition : L'utilisateur dispose d'un identifiant (email) et d'un mot de passe valides.

Postcondition : L'utilisateur est connecté avec succès et accède à la fenêtre de gestion du système.

Flow of Events

1. saisit son identifiant : email et son mot de passe.
2. clique sur le bouton "Se connecter"
5. if Information valides
4.1. SYSTEM affiche un message ("connexion réussit").
4.1. SYSTEM ouvrir la fenêtre de gestion
6. else
5.1. SYSTEM Message d'erreur
end if

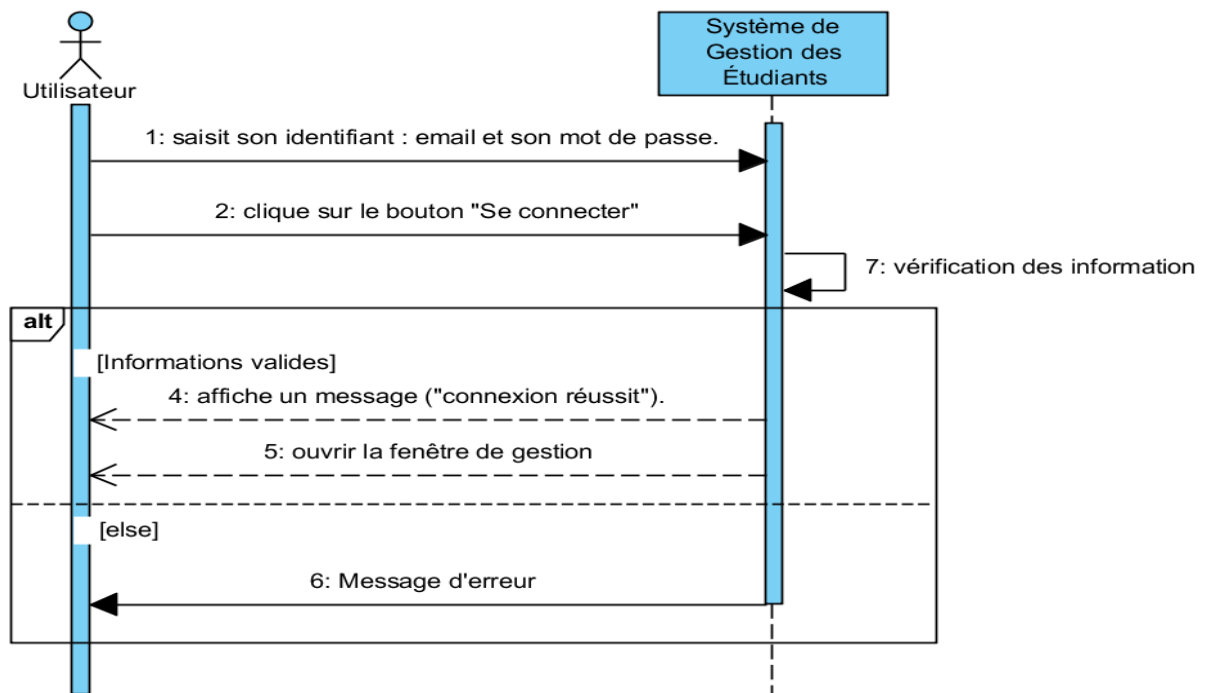


Figure 3 : diagramme de séquence Système (authentification)

bienvenue dans votre espace administratif

Login Her !

Email :

Mot de passe :

Figure4 : maquette « authentification »

2. Diagramme de séquence *détaillé* de UC1

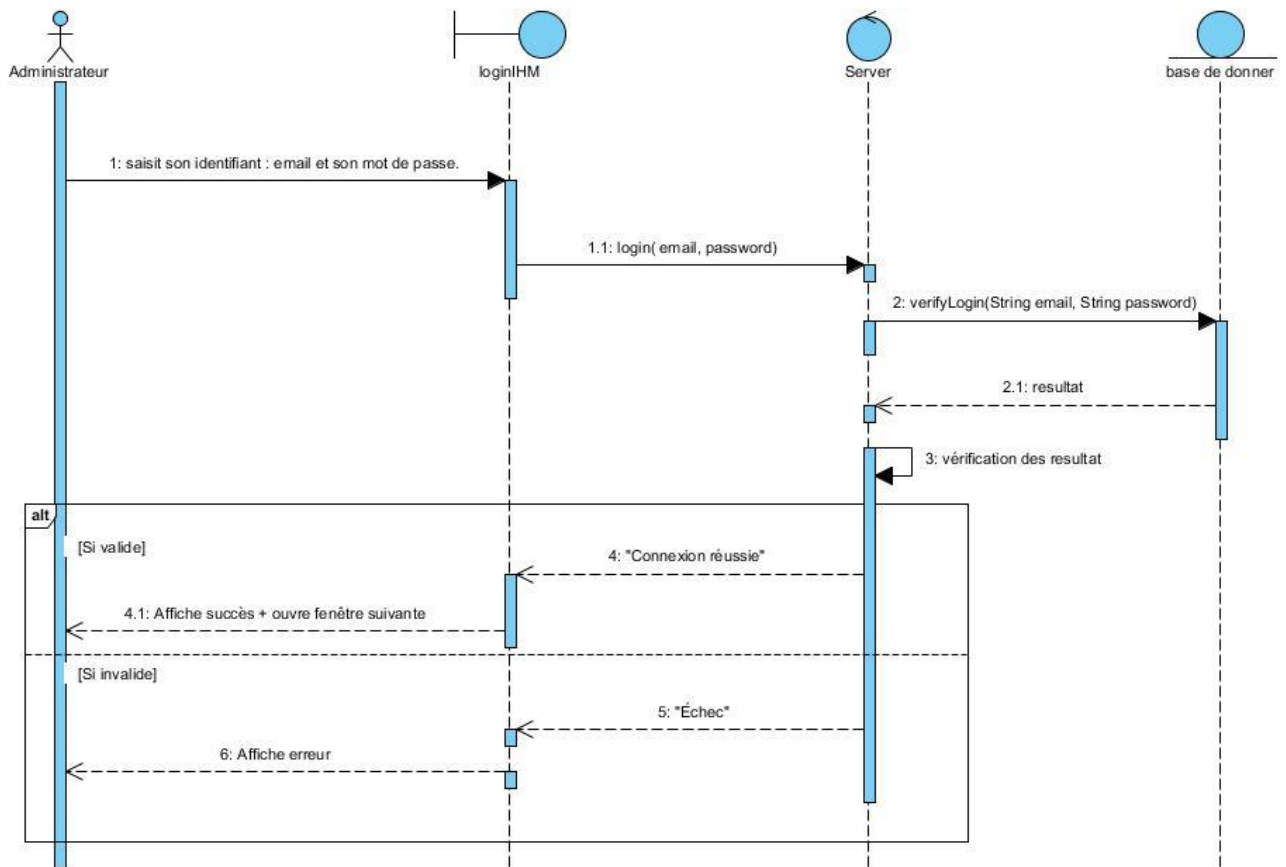


Figure5 : diagramme de séquence détaillé de US1

UC2 : Consulter la liste des Etudiants

1. Diagramme de séquence système de UC2

❖ Cas d'utilisation : Consulter la liste des Etudiants

Acteur principal : Utilisateur (Administrateur)

Précondition : L'administrateur est authentifié avec succès.

Postcondition : Le système affiche la fenêtre de gestion principale.

Flow of Events

1. L'administrateur demande de Consulter la liste des étudiants.
2. SYSTEM affiche la liste de tous les Etudiants
4. L'administrateur choisi la Class des étudiant à consulter
5. if Liste disponible
4.1. SYSTEM affiche la liste des Etudiants de la class sélectionner
5. else
5.1. SYSTEM Message ("Aucun étudiant trouvé".)
end if

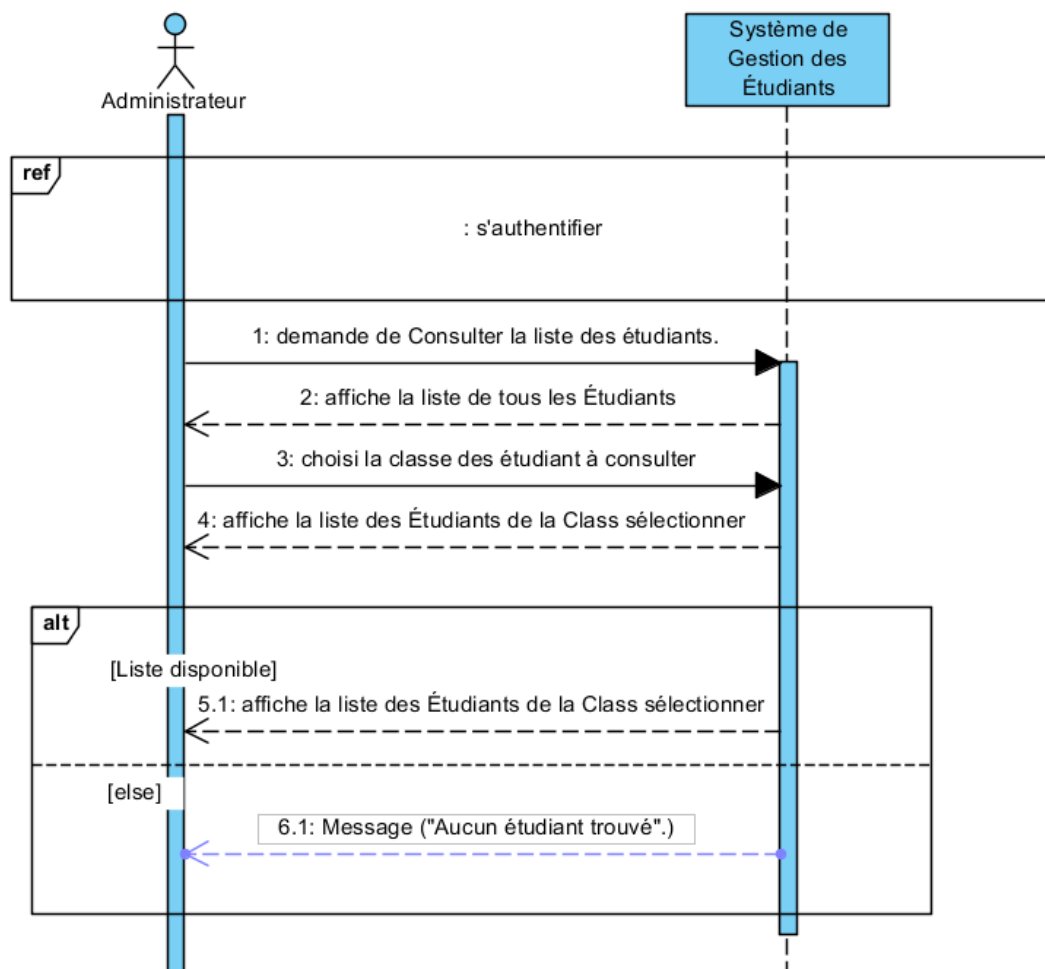


Figure6 : diagramme de séquence Système de US2

Consulter la liste des Etudiants

Étudiants de la classe :

Item 1

rechercher un Etudiant par Nom :

CodeEtudiant	Nom	Prenom	Email	Filier	Classe

Imprimer

Figure7 : maquette « Consulter la liste des Etudiants »

2. Diagramme de séquence détaillé de UC2

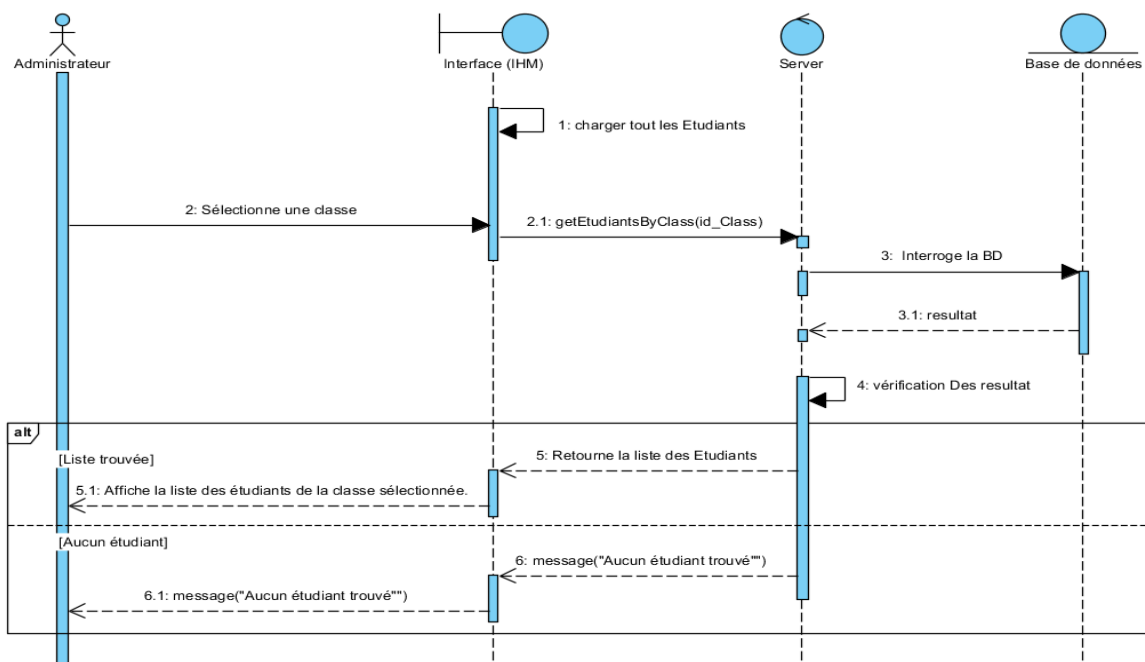


Figure8 : Diagramme de séquence détaillé de US2

3. Diagramme de classe participantes de UC2

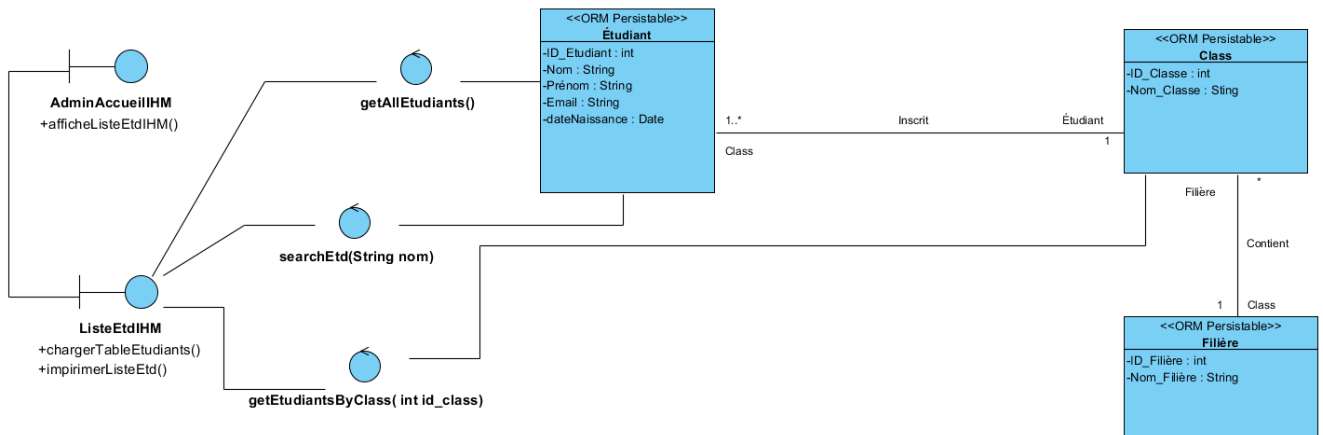


Figure 9 : Diagramme de classe Participantes de US2

UC3 : Mettre à jour la liste des Etudiants

1. Diagramme de séquence système de UC3

1. Cas d'utilisation : Ajouter un étudiant

Acteur principal : Administrateur

Précondition : L'administrateur est authentifié et a accès à la section de gestion des étudiants.

Postcondition : Un étudiant est ajouté avec succès dans la base de données avec les informations renseignées.

 Flow of Events

- | |
|--|
| 1. demande le formulaire d'ajout d'étudiant |
| 2. SYSTEM affiche le formulaire d'ajout d'étudiant. |

3. remplit les champs du formulaire
4. valide le formulaire
5. if Information valide
4.1. SYSTEM confirme l'ajout de l'étudiant
6. else
5.1. SYSTEM Message d'erreur
end if

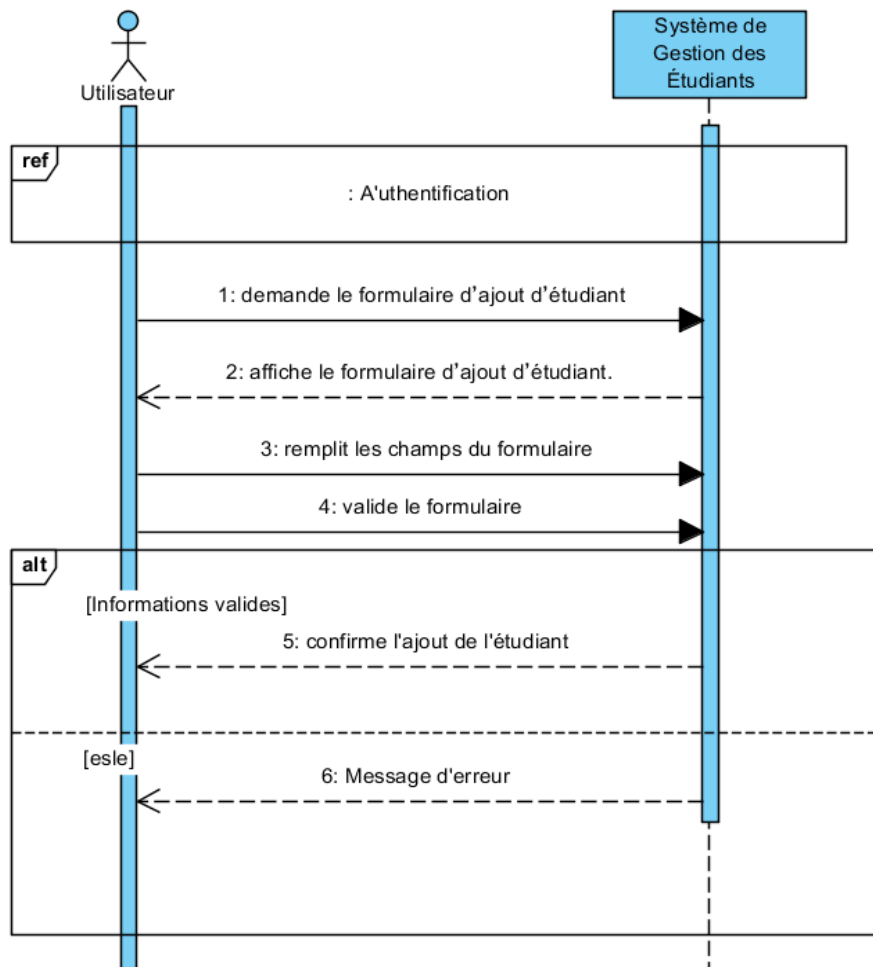


Figure 10 : Diagramme de séquence de cas ajouter un étudiant

2. Cas d'utilisation : Modifier un étudiant

Acteur principal : Administrateur

Précondition :

- L'administrateur est authentifié.
- L'étudiant à modifier est déjà enregistré dans le système.

Postcondition :

- Les informations de l'étudiant sont mises à jour avec succès dans la base de données

Flow of Events

1. accède à la gestion des étudiants.
2. SYSTEM affiche la liste des étudiants.
3. Sélectionne un étudiant à modifier.
4. met à jour les informations nécessaires.
5. valide les modifications.
6. if Informations valides
4.1. SYSTEM confirme l'a modification de l'étudiant
7. else
5.1. SYSTEM Message d'erreur
end if

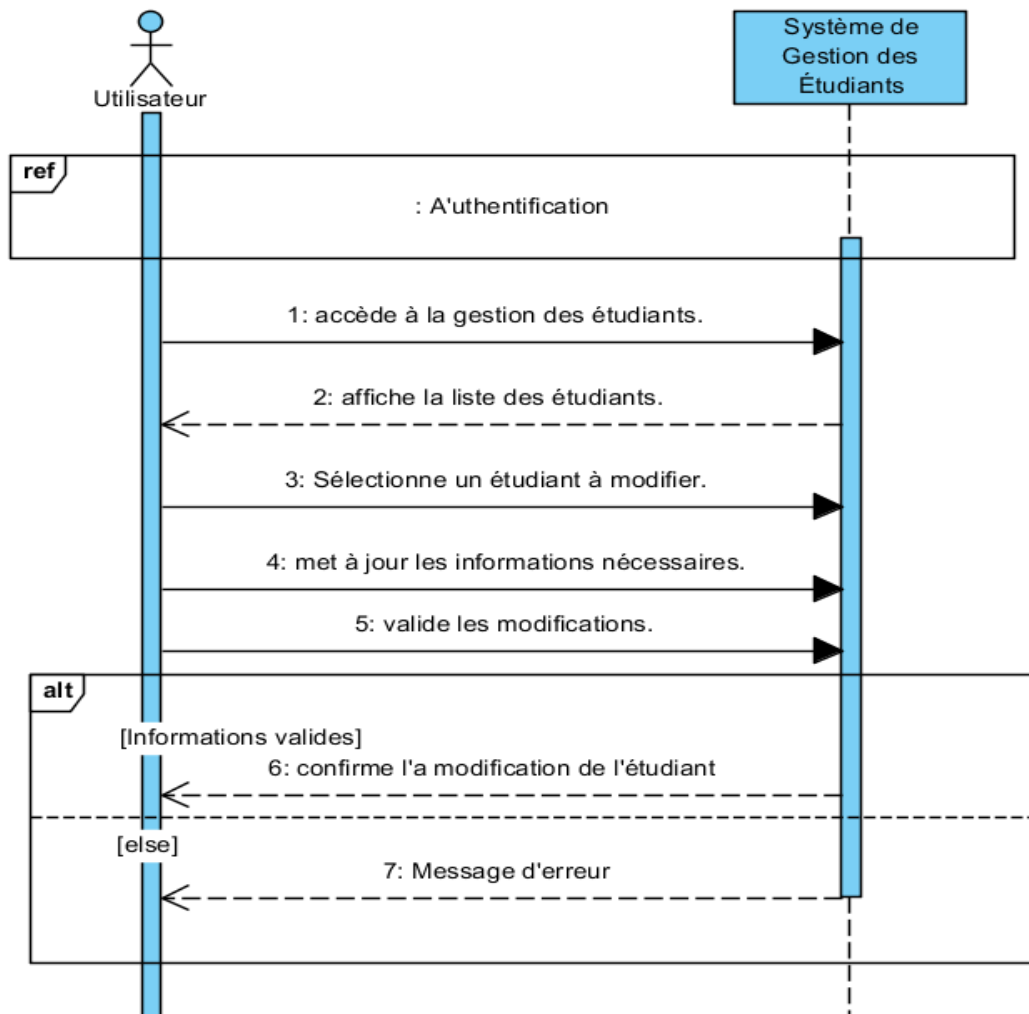


Figure 11 : Diagramme de séquence de cas modifier un étudiant

3. Cas d'utilisation : supprimer un étudiant

Acteur principal : Administrateur

Préconditions :

- L'administrateur est authentifié.
- L'étudiant à supprimer est déjà enregistré dans le système.

Postconditions :

- L'étudiant est supprimé de la base de données.
- Ses informations ne sont plus accessibles via l'application.

1. accède à la gestion des étudiants.
2. SYSTEM affiche la liste des étudiants.
3. sélectionne un étudiant à supprimer.
5. SYSTEM affiche un message de confirmation.
6. if administrateur confirme la suppression.
6.1. SYSTEM supprime l'étudiant de la base de données
6.2 SYSTEM système met à jour la liste des étudiants.
7. else
5.1. SYSTEM Message annulation de suppression
end if

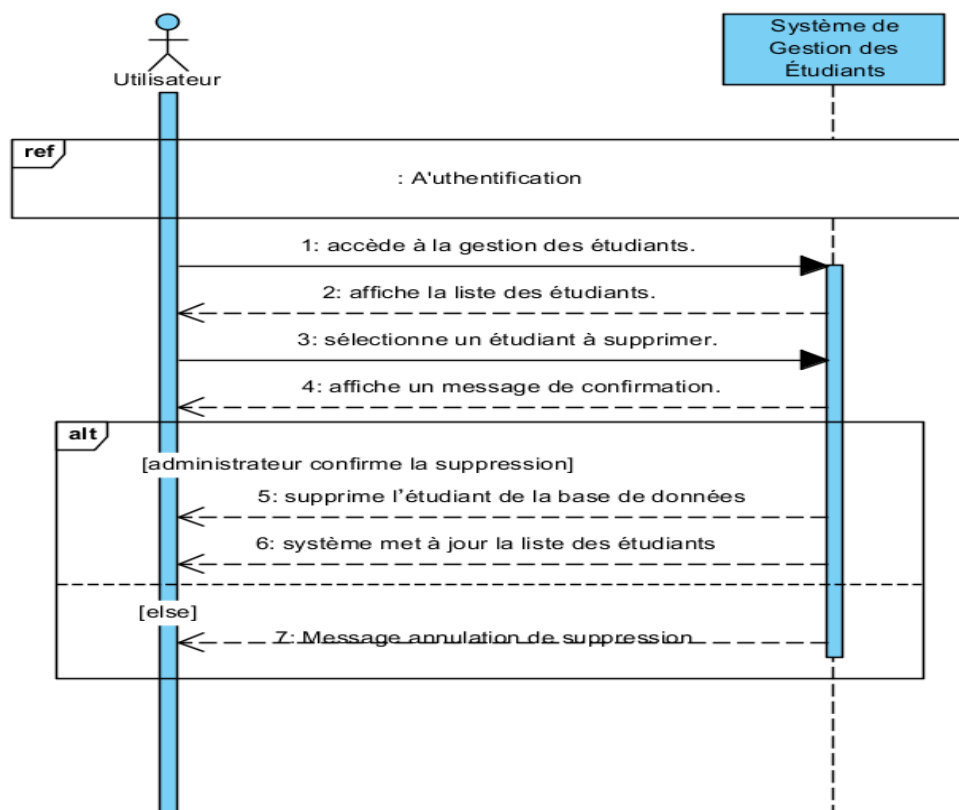


Figure 12 : Diagramme de séquence de cas supprimer un étudiant

<

Gestion Des Etudinats

Nom

Prenom

Email

date d'ennesanse

Étudiants de la classe :

Item 1 ▼

Ajouter

Modifer

Supremmer

CodeEtudiant	Nom	Prenom	Email	Filiere	Classe
--------------	-----	--------	-------	---------	--------

Imprimer

Figure 13 : maquette « Mettre à jour la liste des Etudiants »

4. Diagramme de classe participants de UC3

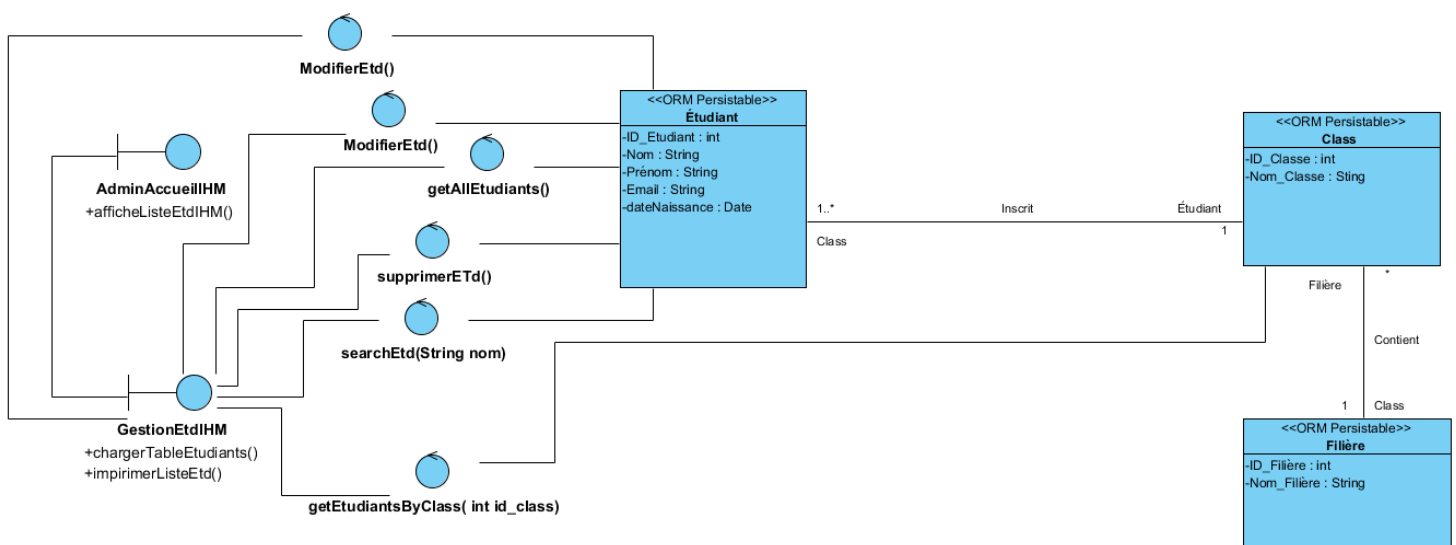


Figure 14 : Diagramme de classe Participantes de US3

5. Modèle Physique de Données (MPD)

1. Présentation du Modèle

Le Modèle Physique de Données (MPD) représente la structure réelle et détaillée de la base de données telle qu'elle sera implémentée. Il définit les **tables**, leurs **champs**, les **types de données**, ainsi que les **relations** entre elles à l'aide de **clés primaires** et **clés étrangères**.

Ce modèle assure l'**intégrité**, la **cohérence** et la **performance** de la base de données.

Le schéma ci-dessous illustre l'organisation des données du système et les liens entre les différentes entités.

2. Modèle Physique de Données (MPD)(schéma) :

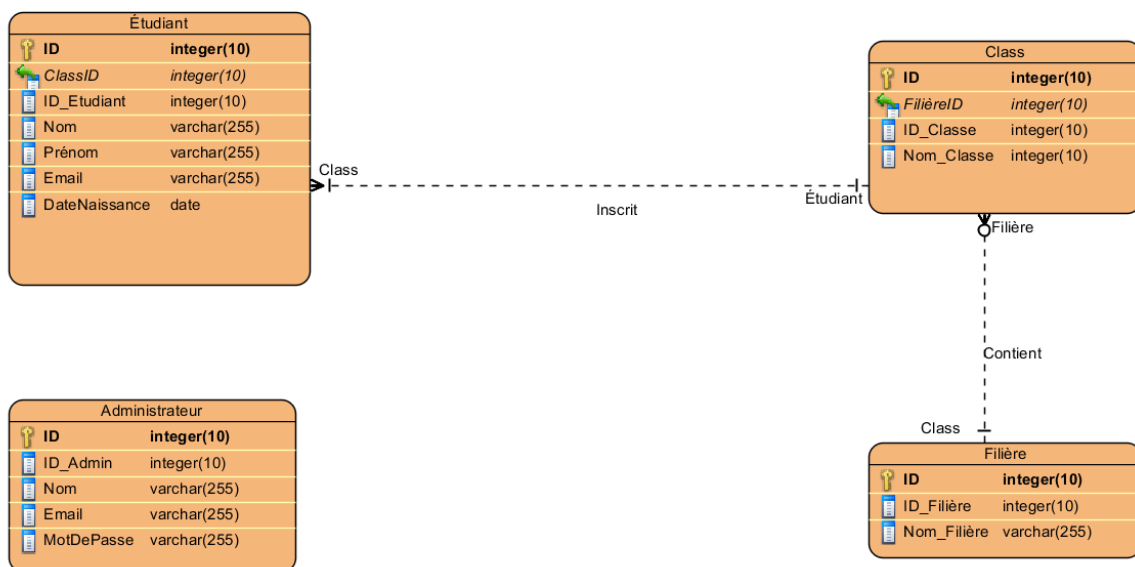


Figure 15 : Modèle Physique de Données (MPD)(schéma)

IV. IV Tests et Validation

1. Introduction

La phase de test constitue une étape essentielle dans le cycle de développement de toute application logicielle. Son objectif principal est de vérifier que les différentes fonctionnalités implémentées répondent bien aux exigences initiales et fonctionnent correctement dans divers scénarios d'utilisation.

Dans le cadre de ce projet de gestion des étudiants basé sur l'architecture RMI (Remote Method Invocation), la phase de test a permis de valider aussi bien le bon fonctionnement des composants serveur (tels que les services de connexion et de gestion des étudiants) que l'interaction entre les différentes couches de l'application (IHM, services, base de données).

Les tests jouent un rôle primordial pour garantir la **fiabilité**, la **robustesse** et la **stabilité** du système. Ils permettent de détecter d'éventuelles erreurs, de vérifier la logique métier et d'assurer une communication fluide entre le client et le serveur via les services distants. Grâce à cette étape, nous avons pu nous assurer que l'application répond aux attentes et fonctionne correctement dans des conditions normales d'utilisation.

2. Objectifs des tests

Les tests ont pour but de s'assurer que le système développé fonctionne conformément aux spécifications définies, en identifiant et en corrigeant les erreurs éventuelles avant la mise en production. Dans le cadre de ce projet de gestion des étudiants, les objectifs principaux des tests sont les suivants :

- **Vérifier la validité fonctionnelle** : s'assurer que chaque fonctionnalité (authentification, ajout/suppression d'un étudiant, consultation de la liste, etc.) fonctionne comme prévu.
- **Tester la communication client-serveur** : garantir que les appels distants via RMI se font correctement, sans perte de données ni erreur d'exécution.
- **Assurer la robustesse du système** : tester le comportement du système face à des cas limites (informations manquantes, identifiants erronés, connexions multiples, etc.).
- **Valider l'intégration des composants** : s'assurer que les différentes parties du projet (IHM, couche métier, base de données) fonctionnent ensemble de manière cohérente et fluide.
- **Détecter les anomalies et les corriger** : identifier les bugs ou incohérences logicielles afin de les corriger avant le déploiement.
- **Évaluer les performances** : s'assurer que le système reste réactif et stable même avec plusieurs requêtes simultanées.

3. Technologies utilisées

JUnit 5

JUnit est un framework de test unitaire pour le langage Java. Il permet de vérifier que chaque composant de l'application fonctionne comme prévu. Dans ce projet, JUnit a été utilisé pour tester les

différentes classes de la couche serveur, notamment `AdminDAO`, en simulant les appels et en vérifiant les résultats attendus.

JaCoCo (Java Code Coverage)

JaCoCo est un outil de couverture de code utilisé pour mesurer le pourcentage de code testé par les tests unitaires. Il permet d'identifier les parties du code qui ne sont pas encore couvertes par des tests, afin d'améliorer la fiabilité et la qualité du logiciel.

Maven

Maven est un outil de gestion de projet et d'automatisation de la compilation pour les projets Java. Il a été utilisé dans ce projet pour :

- Gérer les dépendances des bibliothèques comme **JUnit** et **JaCoCo**,
- Organiser le cycle de vie du projet (compilation, test, packaging),
- Exécuter les tests de manière automatisée,
- Générer des rapports de couverture de code avec JaCoCo.

L'utilisation de Maven a permis de simplifier la configuration du projet et de garantir une meilleure reproductibilité des tests.

SonarLint

SonarLint est une extension d'analyse statique de code qui permet de détecter automatiquement les **bugs, vulnérabilités, mauvaises pratiques et fuites de ressources** dans le code source.

Dans ce projet, SonarLint a été utilisé pour analyser le code côté serveur (DAO, services, RMI) et corriger les problèmes tels que :

- La non-fermeture des ressources (java:S2095),
- Les conditions inutiles ou toujours vraies/faux,
- Les conventions de nommage et la complexité cyclomatique.

Grâce à SonarLint, la qualité du code a été améliorée dès la phase de développement, en intégrant de bonnes pratiques recommandées par les standards internationaux (CERT, CWE, etc.).

4. Configuration Maven

Le fichier `pom.xml` contient la configuration nécessaire pour utiliser **JUnit** et **JaCoCo** :

1. Configuration *JUnit* :

```
<!-- JUnit Jupiter (API + Engine) -->
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-api</artifactId>
  <version>5.9.3</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.9.3</version>
  <scope>test</scope>
</dependency>
```

2. Configuration JaCoCo :

```
<!-- JaCoCo for test coverage -->
<plugin>
  <groupId>org.jacoco</groupId>
  <artifactId>jacoco-maven-plugin</artifactId>
  <version>${jacoco.version}</version>
  <executions>
    <execution>
      <goals>
        <goal>prepare-agent</goal>
      </goals>
      <configuration>
        <excludes>
          <exclude>sun/**</exclude>
          <exclude>com/sun/**</exclude>
          <exclude>jdk/**</exclude>
          <exclude>java/**</exclude>
          <exclude>javax/**</exclude>
        </excludes>
        <propertyName>surefire.jacoco.args</propertyName>
      </configuration>
    </execution>
    <execution>
      <id>report</id>
      <phase>verify</phase>
      <goals>
        <goal>report</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

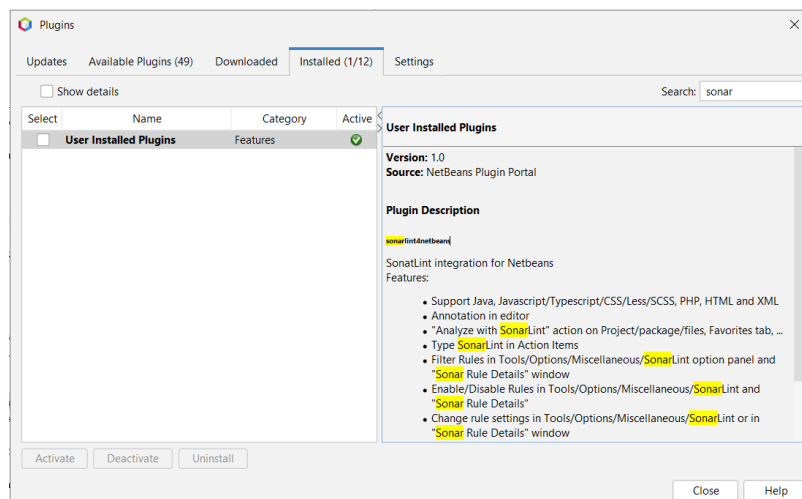
Une fois la configuration faite, les tests sont lancés via :

- mvn test
- mvn jacoco:report

3. Configuration SonarLite:

a. Installation de SonarLint :

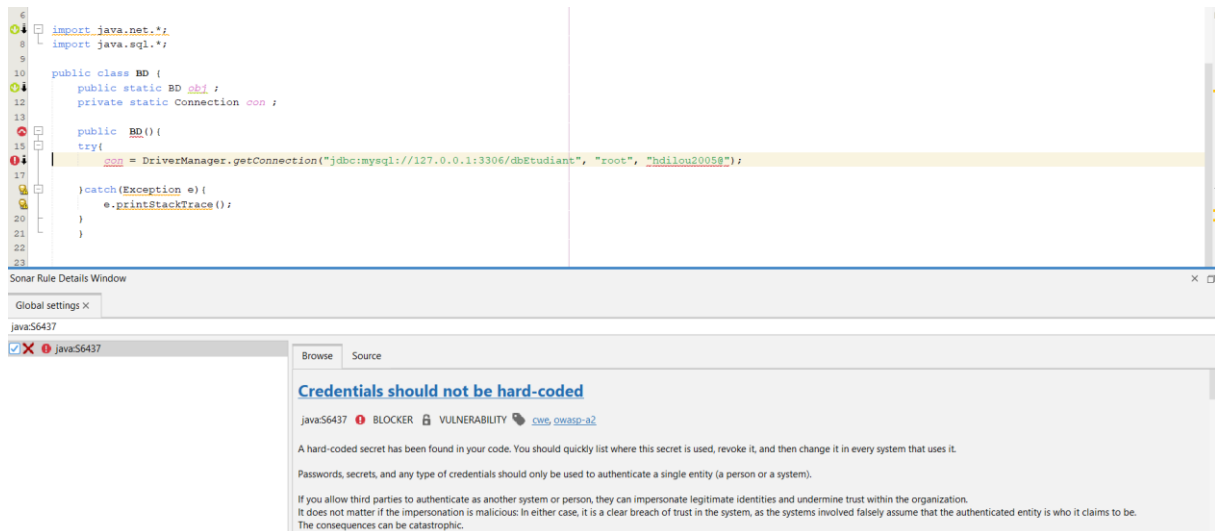
- Dans NetBeans, accéder à Outils > Plugins.
- Rechercher **SonarLint** dans l'onglet "Plugins disponibles".
- Installer le plugin, puis redémarrer NetBeans.



b. Activation de SonarLint :

- Une fois le plugin installé, SonarLint commence à analyser automatiquement les fichiers .java ouverte.
- Les erreurs et recommandations s'affichent dans l'éditeur sous forme de surlignages et dans l'onglet "Action Items" ou "SonarLint Report".

c. Exemple d'analyse SonarLint (classe BD) :



Explication de l'erreur SonarLint : java : S6437 — "Secret codé en dur" :

SonarLint a détecté une **vulnérabilité de sécurité critique** dans le code, car un mot de passe ("hdilou2005@") est directement écrit dans le code source. C'est ce qu'on appelle un **"secret codé en dur"**.

5. Tests des Classes

Les tests unitaires ont été réalisés pour vérifier le bon fonctionnement des différentes classes du projet. Chaque classe métier a fait l'objet de cas de test spécifiques afin de s'assurer que :

- Les méthodes retournent les résultats attendus,
- Les exceptions sont bien gérées en cas d'erreurs,
- Les comportements anormaux sont correctement anticipés.

1. Exemple de Test « Class Etudiant »

Pour assurer la qualité et le bon fonctionnement de la classe Etudiant, une classe de test nommée **EtudiantTest** a été créée. Cette classe contient l'ensemble des tests unitaires permettant de valider le comportement attendu des méthodes de la classe Etudiant

a. Code de test class JUnit « *EtudiantTest* »

```
class EtudiantTest {

    private Etudiant etudiant;
    private Date dateNaissance;

    @BeforeEach
    void setUp() {
        // Exemple de date
        dateNaissance = Date.valueOf("2000-05-15");

        etudiant = new Etudiant("Ali", "Ahmed", "ali.ahmed@example.com", dateNaissance, 2);
    }

    @Test
    void testConstructeurAvecParametres() {
        assertEquals("Ali", etudiant.getNom());
        assertEquals("Ahmed", etudiant.getPrenom());
        assertEquals("ali.ahmed@example.com", etudiant.getEmail());
        assertEquals(dateNaissance, etudiant.getDateNaissance());
        assertEquals(2, etudiant.getIdClasse());
    }

    @Test
    void testConstructeurSansParametres() {
        Etudiant e = new Etudiant();
        assertNotNull(e);
    }

    @Test
    void testSettersEtGetters() {
        etudiant.setId(10);
        etudiant.setNom("Khalid");
        etudiant.setPrenom("Youssef");
        etudiant.setEmail("khalid.youssef@example.com");
        etudiant.setDateNaissance(Date.valueOf("1999-10-01"));
        etudiant.setIdClasse(5);
        etudiant.setNomClasse("2ème Année BTS");
        etudiant.setNomFiliere("Informatique");



















        assertEquals(10, etudiant.getId());
        assertEquals("Khalid", etudiant.getNom());
        assertEquals("Youssef", etudiant.getPrenom());
        assertEquals("khalid.youssef@example.com", etudiant.getEmail());
        assertEquals(Date.valueOf("1999-10-01"), etudiant.getDateNaissance());
        assertEquals(5, etudiant.getIdClasse());
        assertEquals("2ème Année BTS", etudiant.getNomClasse());
        assertEquals("Informatique", etudiant.getNomFiliere());
    }
}
```

Rapport de couverture de code (JaCoCo)

L'exécution des tests a permis de produire un rapport de couverture, indiquant le pourcentage du code testé. Voici un aperçu des résultats obtenus :

GestionDesEtudiants > Entite > Etudiant

Etudiant

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
• Etudiant(String,String,String,Date,int)		100 %		n/a	0	1	0	7	0	1
• setId(int)		100 %		n/a	0	1	0	1	0	1
• setNom(String)		100 %		n/a	0	1	0	1	0	1
• setPrenom(String)		100 %		n/a	0	1	0	1	0	1
• setEmail(String)		100 %		n/a	0	1	0	1	0	1
• setDateNaissance(Date)		100 %		n/a	0	1	0	1	0	1
• setIdClasse(int)		100 %		n/a	0	1	0	1	0	1
• setNomClasse(String)		100 %		n/a	0	1	0	1	0	1
• setNomFiliere(String)		100 %		n/a	0	1	0	1	0	1
• Etudiant()		100 %		n/a	0	1	0	2	0	1
• getId()		100 %		n/a	0	1	0	1	0	1
• getNom()		100 %		n/a	0	1	0	1	0	1
• getPrenom()		100 %		n/a	0	1	0	1	0	1
• getEmail()		100 %		n/a	0	1	0	1	0	1
• getDateNaissance()		100 %		n/a	0	1	0	1	0	1
• getIdClasse()		100 %		n/a	0	1	0	1	0	1
• getNomClasse()		100 %		n/a	0	1	0	1	0	1
• getNomFiliere()		100 %		n/a	0	1	0	1	0	1
Total	0 of 77	100 %	0 of 0	n/a	0	18	0	25	0	18

Ces résultats témoignent d'une couverture adéquate des principales classes métiers, contribuant ainsi à garantir la fiabilité et la qualité globale du code.







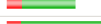

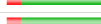








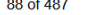
2. Exemple de test class « AdminDAO »

Pour assurer la qualité et le bon fonctionnement de la classe Etudiant, une classe de test nommée **AdminDAOTest** a été créée. Cette classe contient l'ensemble des tests unitaires permettant de valider le comportement attendu des méthodes de la classe Etudiant

a. Rapport de couverture de code (JaCoCo)

L'exécution des tests a permis de produire un rapport de couverture, indiquant le pourcentage du code testé. Voici un aperçu des résultats obtenus :

AdminDAO

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
• chargercmbClass()		79 %		75 %	1	3	4	19	0	1
• supprimerEtudiant(int)		61 %		50 %	1	2	3	9	0	1
• modifierEtudiant(int, String, String, String, Date, int)		78 %		50 %	1	2	3	14	0	1
• ajouterEtudiant(String, String, String, Date, int)		76 %		50 %	1	2	3	13	0	1
• searchETD(String)		88 %		100 %	0	2	3	24	0	1
• afficherEtudiantsParClasse(int)		88 %		100 %	0	2	3	23	0	1
• afficherEtudiants()		87 %		100 %	0	2	3	22	0	1
• verifyLogin(String, String)		82 %		50 %	2	3	4	13	0	1
• login(String, String)		0 %		n/a	1	1	1	1	1	1
• AdminDAO()		100 %		n/a	0	1	0	2	0	1
Total	88 of 487	81 %	6 of 20	70 %	7	20	27	140	1	10

V. Présentation des Interfaces Graphiques de l'Application

a. Interface de connexion (Login)

L'interface de connexion est la première page que rencontre l'utilisateur pour accéder à l'espace administratif de l'application. Conçue pour être simple et intuitive, elle comprend :

- Deux champs de saisie :
 - **Email** : pour renseigner l'adresse électronique.
 - **Mot de passe** : pour saisir le mot de passe associé.
- Un bouton "Login" pour valider et accéder à l'interface.

Cette interface assure une expérience utilisateur fluide tout en garantissant la sécurité d'accès à l'application.



Login Her !

Email :

Mot de passe :

Login

b. L'Interface d'Accueil Administrateur

L'interface d'accueil admin est conçue pour offrir une gestion centralisée et efficace des fonctionnalités administratives. Elle se compose des éléments suivants :

- **Section "Gestion des Étudiants"** : Permet d'accéder aux outils nécessaires pour administrer les dossiers des étudiants.
- **Option "Consulter la liste des Étudiants"** : Donne un accès direct à la liste complète des étudiants enregistrés dans le système.
- **Bouton "Exit"** : Permet de quitter rapidement la session en cours.

Cette interface allie convivialité et fonctionnalité, permettant à l'administrateur d'effectuer ses tâches de manière intuitive et sécurisée.



c. L'Interface de Consulter la liste des Etudiants

Cette interface affiche la liste des étudiants sous forme de tableau et propose une **recherche par nom** pour un accès rapide aux données.

Fonctionnalités :

- **Tableau clair** : Code, Nom, Prénom, Email, Filière, Classe.
- **Recherche par nom** : Permet de filtrer instantanément les étudiants.
- **Bouton "Imprimer"** : Exporte la liste pour archivage ou partage.

Utilité : Gestion simplifiée des étudiants avec accès rapide aux informations

Consulter la liste des Etudiants

Étudiants de la classe : Classe A

rechercher un Etudiant par Nom :

CodeEtudiant	Nom	Prenom	Email	Filier	Classe
34	Nom57	Prenom25	email535025@example.co...	Informatique	Classe A
47	Nom66	Prenom94	email735528@example.co...	Informatique	Classe A
103	Nom76	Prenom26	email996175@example.co...	Informatique	Classe A
119	Doe	John	john.doe@example.com	Informatique	Classe A
122	hamza	hamza	hamza@example.com	Informatique	Classe A
124	hamza	hamza	hamza20013@example.co...	Informatique	Classe A
126	Nouveau	Etudiant	etudiant_6de7ec2e@examp	Informatique	Classe A
128	A	Supprimer	supprimer_test_5f0fd562@	Informatique	Classe A
130	Test	Etudiant	test_e2a52d37@example.c.	Informatique	Classe A
131	A	Modifier	test_b5df38b2@example.co	Informatique	Classe A
132	A	Modifier	test_5967c9e2@example.c.	Informatique	Classe A
133	A	Supprimer	supprimer_test_f6736006@	Informatique	Classe A
135	Test	Etudiant	test_ac973845@example.c.	Informatique	Classe A
139	Dupont	Jean	etudiant_5ff1dd68@examp	Informatique	Classe A
140	A	Supprimer	supprimer_etudiant_5258d1	Informatique	Classe A
142	Dupont	Jean	etudiant_82567f8d@examp	Informatique	Classe A
143	A	Supprimer	supprimer_etudiant_e11b5d	Informatique	Classe A
145	Dupont	Jean	etudiant_6555518d@examp	Informatique	Classe A
146	A	Supprimer	supprimer_etudiant_046093	Informatique	Classe A
148	Dupont	Jean	etudiant_fae5ab06@examp	Informatique	Classe A
149	A	Supprimer	supprimer_etudiant_09df11	Informatique	Classe A
151	A	Supprimer	supprimer_etudiant_884fd2	Informatique	Classe A
153	Dupont	Jean	etudiant_7afa17a5@examp	Informatique	Classe A
155	Dupont	Jean	etudiant_2afc25d7@examp	Informatique	Classe A

Imprimer

d. L'Interface de Gestion des Étudiants

Cette interface offre une solution complète pour gérer les étudiants de la classe A en informatique, combinant visualisation des données et outils de gestion.

Fonctionnalités principales :

1. **Tableau interactif** présentant :
 - Toutes les informations essentielles (Code, Nom, Prénom, Email, Filière)
 - Des indicateurs visuels pour les actions clés (Ajouter/Modifier/Supprimer)
2. **Fonctions de gestion intégrées** :
 - Boutons d'action directement dans le tableau
 - Possibilité d'ajouter de nouveaux étudiants
 - Options pour modifier ou supprimer des entrées existantes
3. **Recherche par nom** : Permet de filtrer instantanément les étudiants.
4. **Bouton "Imprimer"** : Exporte la liste pour archivage ou partage.

<

Gestion Des Etudinsats

Nom

Nom66

Prenom

Prenom94

Email

email735528@example.com

date d'ennese

24 janv. 2013

Étudiants de la classe :

Classe A

Ajouter

Modifier

Supprimer

rechercher un Etudiant par Nom :

CodeEtudiant	Nom	Prenom	Email	Filier	Classe
34	Nom57	Prenom25	email535025@example.co	Informatique	Classe A
47	Nom66	Prenom94	email735528@example.co	Informatique	Classe A
103	Nom76	Prenom26	email996175@example.co	Informatique	Classe A
119	Doe	John	john.doe@example.com	Informatique	Classe A
122	hamza	hamza	hamza@example.com	Informatique	Classe A
124	hamza	hamza	hamza20013@example.co	Informatique	Classe A
126	Nouveau	Etudiant	etudiant_0de7ec2e@exam	Informatique	Classe A
128	A	Supprimer	supprimer_test_5f0fd562@	Informatique	Classe A
130	Test	Etudiant	test_e2a52d37@example.c	Informatique	Classe A
131	A	Modifier	test_b5df38b2@example.c	Informatique	Classe A
132	A	Modifier	test_5967c9e2@example.c	Informatique	Classe A
133	A	Supprimer	supprimer_test_f6736006	Informatique	Classe A
135	Test	Etudiant	test_ac973845@example.c	Informatique	Classe A
139	Dupont	Jean	etudiant_5ff1dd88@exampl	Informatique	Classe A
140	A	Supprimer	supprimer_etudiant_5258d	Informatique	Classe A
142	Dupont	Jean	etudiant_82567f8d@examp	Informatique	Classe A
143	A	Supprimer	supprimer_etudiant_e11b5	Informatique	Classe A
145	Dupont	Jean	etudiant_0555518d@exam	Informatique	Classe A
146	A	Supprimer	supprimer_etudiant_04609	Informatique	Classe A
148	Dupont	Jean	etudiant_fae5ab06@examp	Informatique	Classe A
149	A	Supprimer	supprimer_etudiant_09df11	Informatique	Classe A
151	A	Supprimer	supprimer_etudiant_884fd2	Informatique	Classe A
153	Dupont	Jean	etudiant_7afa17a5@examp	Informatique	Classe A
155	Dupont	Jean	etudiant_2afc25d7@examp	Informatique	Classe A
157	Dupont	Jean	etudiant_ba2823e2@exam	Informatique	Classe A
159	Dupont	Jean	etudiant_ce6cdaa1@exam	Informatique	Classe A
161	Dupont	Jean	etudiant_6f4131e4@exam	Informatique	Classe A

Imprimer

Conclusion Générale

Le développement de cette application de gestion des étudiants a permis de concevoir une solution efficace et fonctionnelle répondant aux besoins de suivi et de gestion des données étudiantes.

L'application offre une interface conviviale grâce à **Java Swing**, facilitant l'interaction avec l'utilisateur tout en assurant une bonne organisation des informations.

Le projet s'est appuyé sur des outils fiables et performants. Java a été utilisé comme langage principal, avec **Maven** pour la gestion du cycle de vie du projet et des dépendances. Pour garantir la robustesse et la fiabilité de l'application, une phase de test rigoureuse a été mise en place en utilisant **JUnit** pour l'écriture des tests unitaires, et **JaCoCo** pour mesurer la couverture de code.

Ce projet m'a permis de mettre en pratique mes compétences en programmation orientée objet, en conception d'interface graphique, ainsi qu'en assurance qualité logicielle. Il constitue une base solide et extensible, pouvant être enrichie par de nouvelles fonctionnalités à l'avenir.