



الجامعة الأورومتوسطية بفاس  
EUROMED UNIVERSITY OF FES  
UNIVERSITÉ EUROMED DE FÈS

## **RAPPORT DE PROJET :**

A large, dark blue ribbon graphic with a 3D effect, featuring two circular cutouts on its top surface. It is centered horizontally and contains the project title in white text.

APPLICATION DE GESTION DE  
NOTES AVEC JAVA RMI

**Réalisé par : NABINTU MALENGELA GLORIA**

**Supervisé par : Mr AMHED AMAMOU**

**Année Universitaire 2023-2024**

## **1. INTRODUCTION**

La communication occupe une place très importante dans la vie humaine. Cette dernière est indispensable dans tous domaines : la vie sociale, professionnelle, civique, et personnelle.

À ce stade, il ne fait désormais plus aucun doute que les technologies de l'information et de la communication représentent la révolution la plus importante et la plus innovante qui a marqué la vie de l'humanité en ce siècle. En effet, loin d'être un éphémère phénomène de mode, ou une tendance passagère, ces technologies viennent nous apporter de multiples comforts à notre mode de vie, car ils ont révolutionné le travail des individus par leur capacité de traitement d'information, d'une part, et de rapprochement des dimensions espace/temps,

d'une autre.

La progression des Technologies de la communication a créé l'interconnexion de réseaux de données solides qui sont profondément impact. Au début, les premiers réseaux étaient limités à échanger les informations reposant sur des caractères entre des systèmes informatique connectes. et avec les dernière générations d'outils de télécommunication. les

reseaux sont en voie d'amelioration, ils ont prient en charge le transtert audio, des flux video.

du texte et des graphismes entre des périphériques de types très différents, la rétroaction devient plus aisée, et les messages se sont beaucoup enrichis.

La disponibilité du réseau permanente nécessite une technologie efficace et fiable. Dans ce cercle s'inscrit notre projet.

Le projet consiste à développer une application de gestion des notes utilisant Java RMI (Remote Method Invocation) et Java Swing pour l'interface graphique. L'application permet à chaque utilisateur de créer, lire, modifier et supprimer des notes. Ce rapport présente la structure du projet, les différentes étapes de développement, le fonctionnement du système et les défis rencontrés.

## **2. ARCHITECTURE**

L'architecture est basée sur le modèle client-serveur avec l'utilisation de Java RMI pour les communications entre le client et le serveur.

- Serveur RMI : Gère la logique pour la gestion des notes et expose des méthodes accessibles aux clients via RMI.
- Client RMI : Fournit une interface utilisateur pour interagir avec le serveur et gérer les notes.
- Registre RMI : Sert de répertoire pour que les clients puissent localiser les objets distants.

### 3. STRUCTURE DES FICHIERS

Le projet est organisé en plusieurs classes réparties entre le serveur et le client :

#### 3.1. Serveur

- Note.java : Classe représentant une note avec titre et contenu.
- NoteManager.java : Interface définissant les méthodes pour gérer les notes.
- NoteManagerImpl.java : Implémentation de l'interface NoteManager.
- NoteServer.java : Classe principale pour démarrer le serveur RMI.

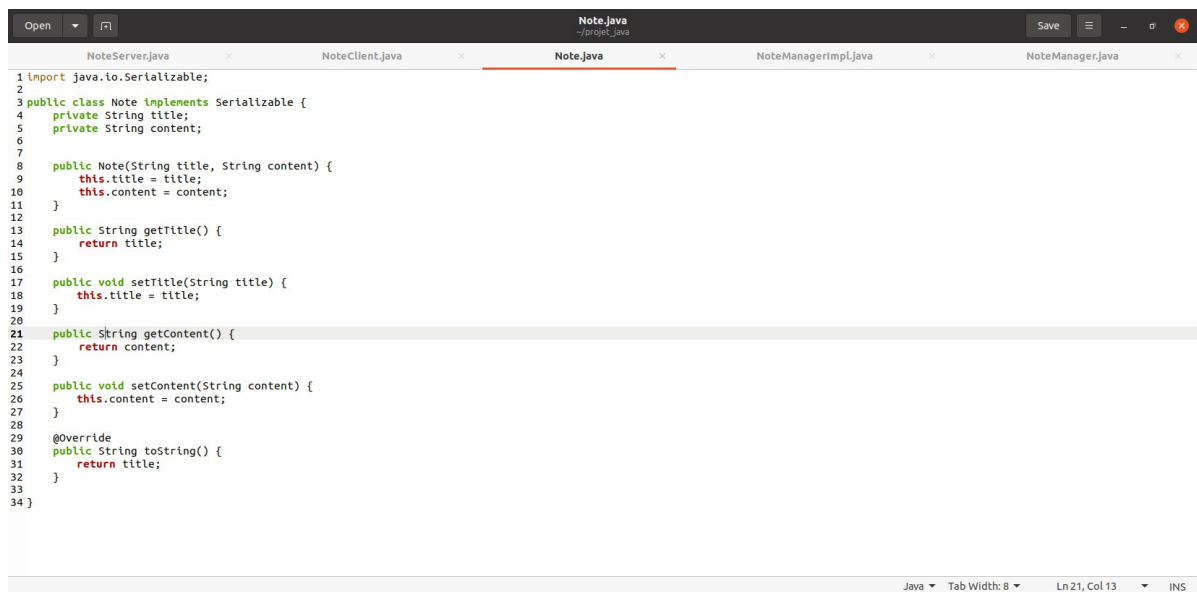
#### 3.2. Client

- NoteClient.java : Classe principale du client avec l'interface utilisateur Swing.

### 4. DEVELOPPEMENT DU SERVEUR

#### 4.1. Classe Note

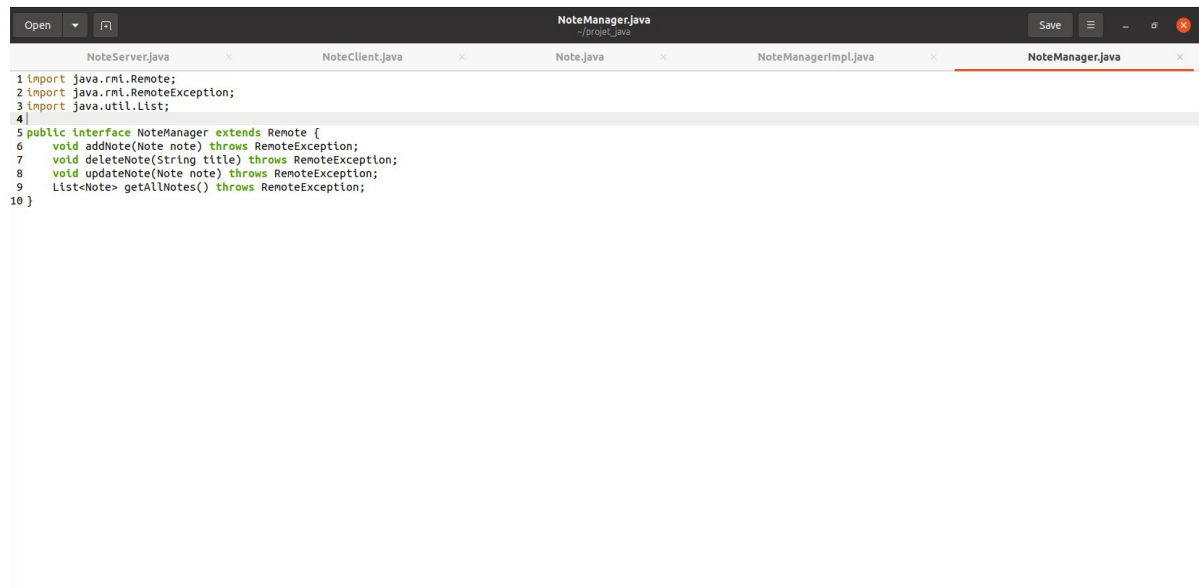
La classe Note représente une note individuelle avec un titre et un contenu.



```
1 import java.io.Serializable;
2
3 public class Note implements Serializable {
4     private String title;
5     private String content;
6
7
8     public Note(String title, String content) {
9         this.title = title;
10        this.content = content;
11    }
12
13    public String getTitle() {
14        return title;
15    }
16
17    public void setTitle(String title) {
18        this.title = title;
19    }
20
21    public String getContent() {
22        return content;
23    }
24
25    public void setContent(String content) {
26        this.content = content;
27    }
28
29    @Override
30    public String toString() {
31        return title;
32    }
33
34 }
```

## 4.2. Interface NoteManager

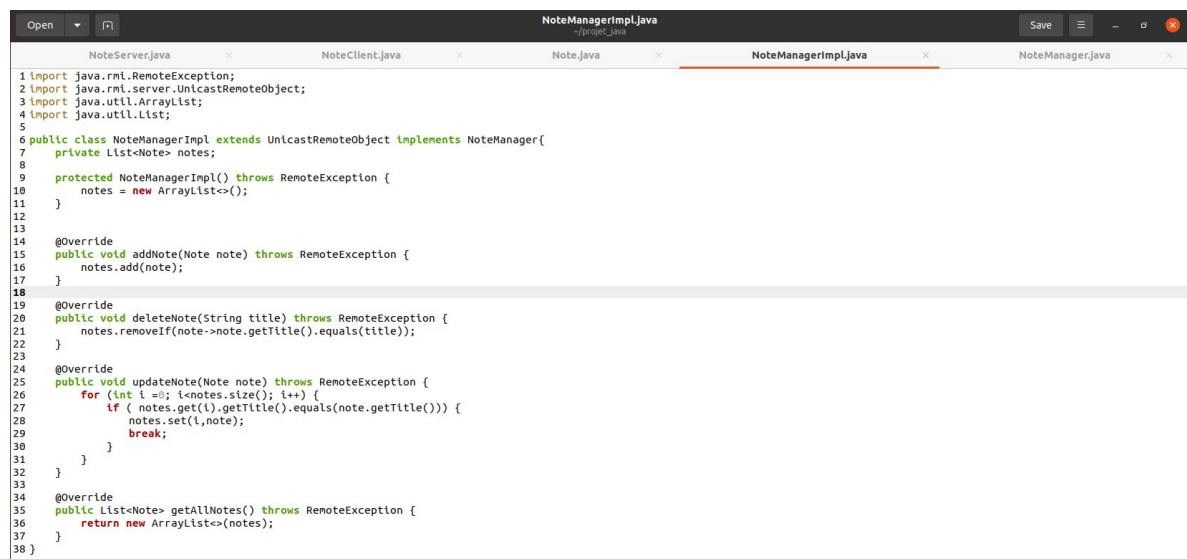
L'interface NoteManager définit les méthodes pour gérer les notes.



```
1 import java.rmi.Remote;
2 import java.rmi.RemoteException;
3 import java.util.List;
4
5 public interface NoteManager extends Remote {
6     void addNote(Note note) throws RemoteException;
7     void deleteNote(String title) throws RemoteException;
8     void updateNote(Note note) throws RemoteException;
9     List<Note> getAllNotes() throws RemoteException;
10 }
```

## 4.3. Implémentation NoteManagerImpl

La classe NoteManagerImpl implémente l'interface NoteManager.



```
1 import java.rmi.RemoteException;
2 import java.rmi.server.UnicastRemoteObject;
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class NoteManagerImpl extends UnicastRemoteObject implements NoteManager {
7     private List<Note> notes;
8
9     protected NoteManagerImpl() throws RemoteException {
10         notes = new ArrayList<>();
11     }
12
13     @Override
14     public void addNote(Note note) throws RemoteException {
15         notes.add(note);
16     }
17
18     @Override
19     public void deleteNote(String title) throws RemoteException {
20         notes.removeIf(note -> note.getTitle().equals(title));
21     }
22
23     @Override
24     public void updateNote(Note note) throws RemoteException {
25         for (int i = 0; i < notes.size(); i++) {
26             if (notes.get(i).getTitle().equals(note.getTitle())) {
27                 notes.set(i, note);
28                 break;
29             }
30         }
31     }
32
33     @Override
34     public List<Note> getAllNotes() throws RemoteException {
35         return new ArrayList<>(notes);
36     }
37 }
38 }
```

## 4.4. Classe NoteServer

La classe NoteServer démarre le serveur RMI et lie l'implémentation du gestionnaire de notes au registre RMI.

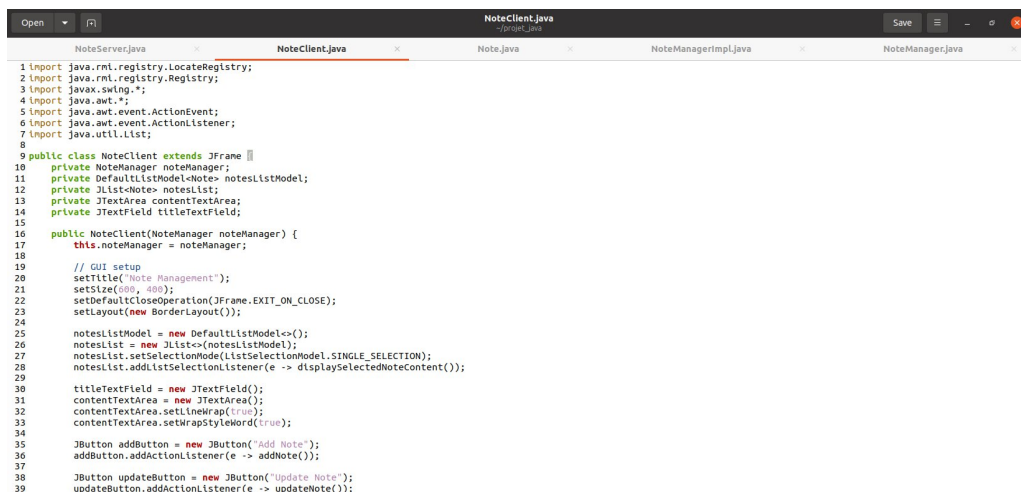


```
1 import java.rmi.registry.LocateRegistry;
2 import java.rmi.registry.Registry;
3
4 public class NoteServer {
5     public static void main(String[] args) {
6         try {
7             NoteManager noteManager = new NoteManagerImpl();
8             Registry registry = LocateRegistry.createRegistry(1099);
9             registry.rebind("NoteManager", noteManager);
10
11             System.out.println("Note Server is ready.");
12         } catch (Exception e) {
13             e.printStackTrace();
14         }
15     }
16 }
17 }
```

## 5. DEVELOPPEMENT DU CLIENT

### 5.1. Classe NoteClient

La classe NoteClient fournit une interface graphique pour gérer les notes. Elle permet d'ajouter, de mettre à jour, de supprimer et d'afficher les notes.



```
1 import java.rmi.registry.LocateRegistry;
2 import java.rmi.registry.Registry;
3 import javax.swing.*;
4 import java.awt.*;
5 import java.awt.event.ActionEvent;
6 import java.awt.event.ActionListener;
7 import java.util.List;
8
9 public class NoteClient extends JFrame {
10     private NoteManager noteManager;
11     private DefaultListModel<Note> notesListModel;
12     private JList<Note> notesList;
13     private JTextArea contentTextArea;
14     private JTextField titleTextField;
15
16     public NoteClient(NoteManager noteManager) {
17         this.noteManager = noteManager;
18
19         // GUI setup
20         setTitle("Note Management");
21         setSize(400, 400);
22         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23         setLayout(new BorderLayout());
24
25         notesListModel = new DefaultListModel<>();
26         notesList = new JList<>(notesListModel);
27         notesList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
28         notesList.addListSelectionListener(e -> displaySelectedNoteContent());
29
30         titleTextField = new JTextField();
31         contentTextArea = new JTextArea();
32         contentTextArea.setLineWrap(true);
33         contentTextArea.setWrapStyleWord(true);
34
35         JButton addButton = new JButton("Add Note");
36         addButton.addActionListener(e -> addNote());
37
38         JButton updateButton = new JButton("Update Note");
39         updateButton.addActionListener(e -> updateNote());
40     }
41
42     private void displaySelectedNoteContent() {
43         // Implementation of displaySelectedNoteContent
44     }
45
46     private void addNote() {
47         // Implementation of addNote
48     }
49
50     private void updateNote() {
51         // Implementation of updateNote
52     }
53 }
```

## 6. DEPLOIEMENT ET EXECUTION

### 6.1. Compilation des fichiers Java

Compiler toutes les classes en utilisant javac :

```
javac Note.java NoteManager.java NoteManagerImpl.java NoteServer.java NoteClient.java
```

### 6.2. Démarrage du registre RMI

Démarrer le registre RMI dans un terminal :

```
rmiregistry
```

### 6.3. Démarrage du serveur

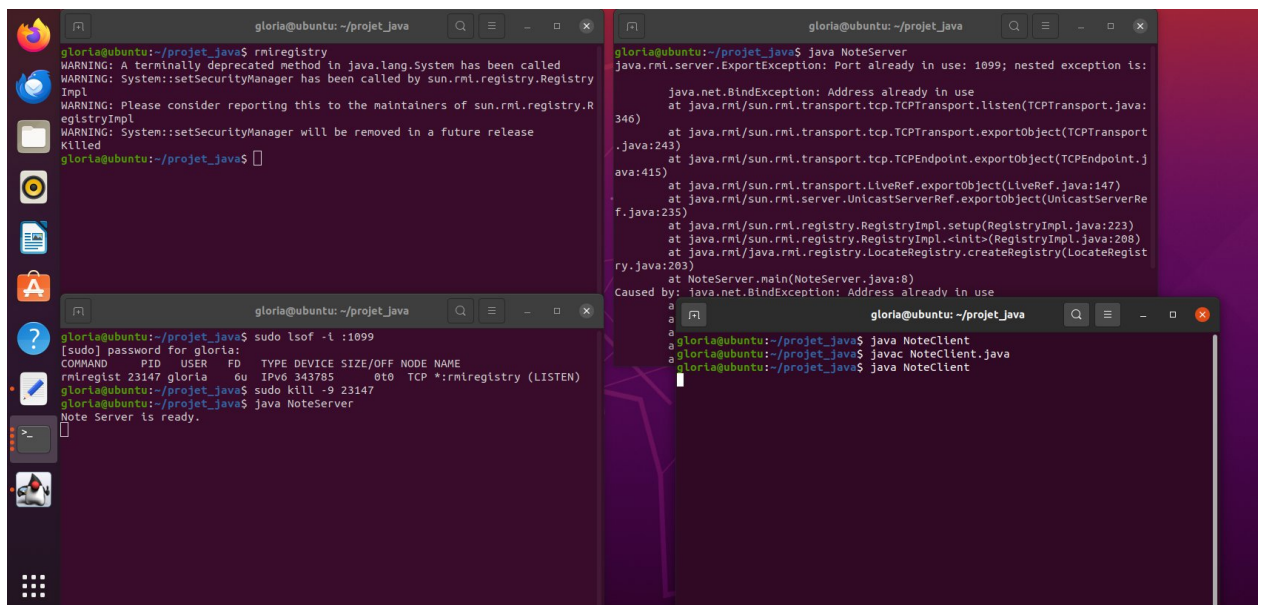
Démarrer le serveur RMI dans un autre terminal :

```
java NoteServer
```

### 6.4. Exécution du client

Exécuter le client dans un autre terminal :

```
java NoteClient
```

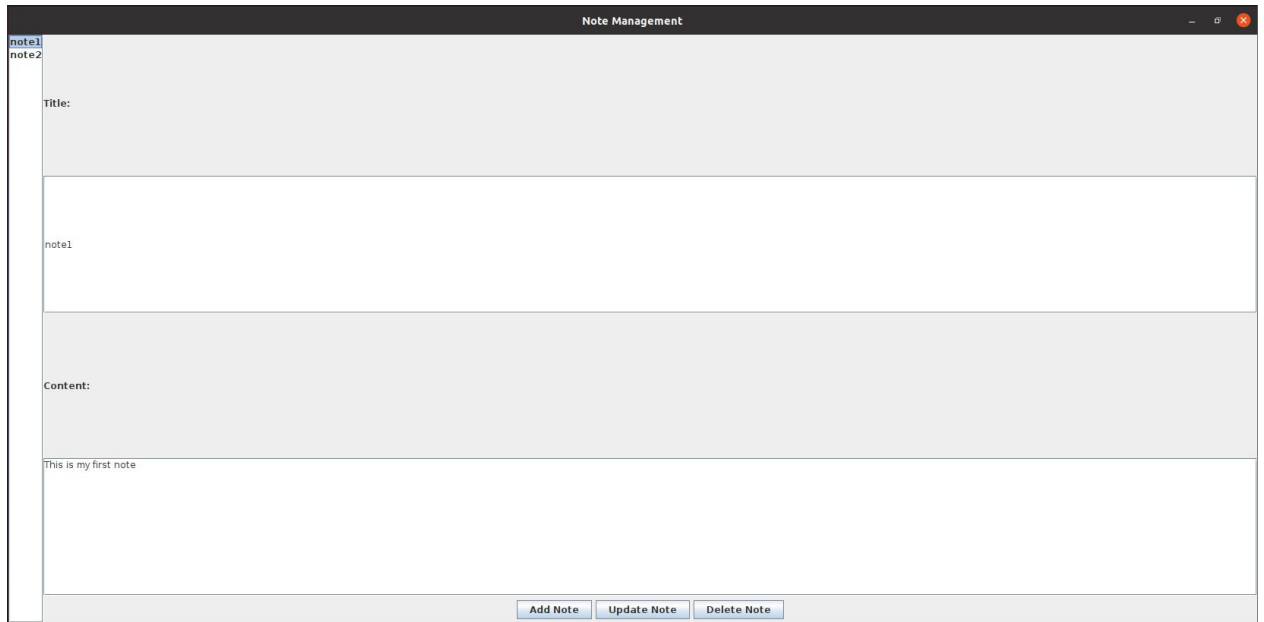


The screenshot displays three terminal windows from a user named 'gloria' on an Ubuntu system, working in the directory '~/projet\_java'.

- Top-left terminal:** Shows the execution of `rmiregistry`. It displays several warnings from the Java runtime regarding deprecated methods and the future removal of the security manager, followed by a 'Killed' message.
- Bottom-left terminal:** Shows the execution of `java NoteServer`. It displays a `java.net.BindException: Address already in use` error, indicating that port 1099 is already occupied.
- Right terminal:** Shows the execution of `java NoteClient`. It displays a `java.net.BindException: Address already in use` error, indicating that port 1099 is already occupied.

The error messages in the bottom-left and right terminals are identical, showing the stack trace for the `BindException` and the specific port (1099) that is already in use.

Une fois le client lancé, l'interface de notre application s'affichera sous forme de fenêtre Swing :



## **7. CONCLUSION**

Ce projet démontre l'utilisation de Java RMI pour créer une application distribuée avec une interface utilisateur Swing. Les fonctionnalités principales incluent la création, la mise à jour, la suppression et l'affichage des notes.

Les défis rencontrés incluent la gestion des erreurs de réseau, la synchronisation des données entre client et serveur, et la création d'une interface utilisateur intuitive. Des améliorations futures pourraient inclure la gestion des utilisateurs, des autorisations, et l'amélioration de l'interface utilisateur.