Université Euro Méditerranéenne Fès

EuroMed University of Fez

Ecole d'Ingénierie Digitale et d'Intelligence Artificielle
(EIDIA)

Filière : 1ème année cyber-security

Semestre : 6

Module : Application repartie

Thème :

Gestion de carnet d'adresse

Soutenu le .. /…/24,

Encadré par :

Pr. Amamou Ahmed

Préparé par l'étudiante:

MALIKI Maroua

Année Universitaire : 2023-2024

Sommaire :

Introduction :

L’étape d’analyse et conception est un processus crucial qui consiste à examiner le système existant et à définir les caractéristiques de la future application. L’objectif principal est de formaliser les étapes préliminaires du développement afin de s'assurer que le système final répondra fidèlement aux besoins du client.

Dans cette section, nous détaillerons les étapes fondamentales pour le développement de notre système de gestion des carnets d’adresses. Nous commencerons par spécifier les besoins de notre application. Pour la conception et la réalisation de celle-ci, nous avons choisi de modéliser en utilisant le formalisme UML (Unified Modeling Language), qui offre une grande flexibilité grâce à l'utilisation de divers diagrammes.

Avant de passer à l’analyse pratique et à la conception, il est important de noter que notre système repose sur une architecture client-serveur et utilise RMI (Remote Method Invocation).

Principe de fonctionnement RMI :

• **Coté Serveur** A la création de l’objet, un stub et un skeleton (avec un port de communication) sont créés coté serveur.

• L'objet serveur s'enregistre auprès d'un annuaire (rmiregistry) en utilisant la classe Naming (méthode rebind).

• L'annuaire (rmiregistry) enregistre le stub de l’objet

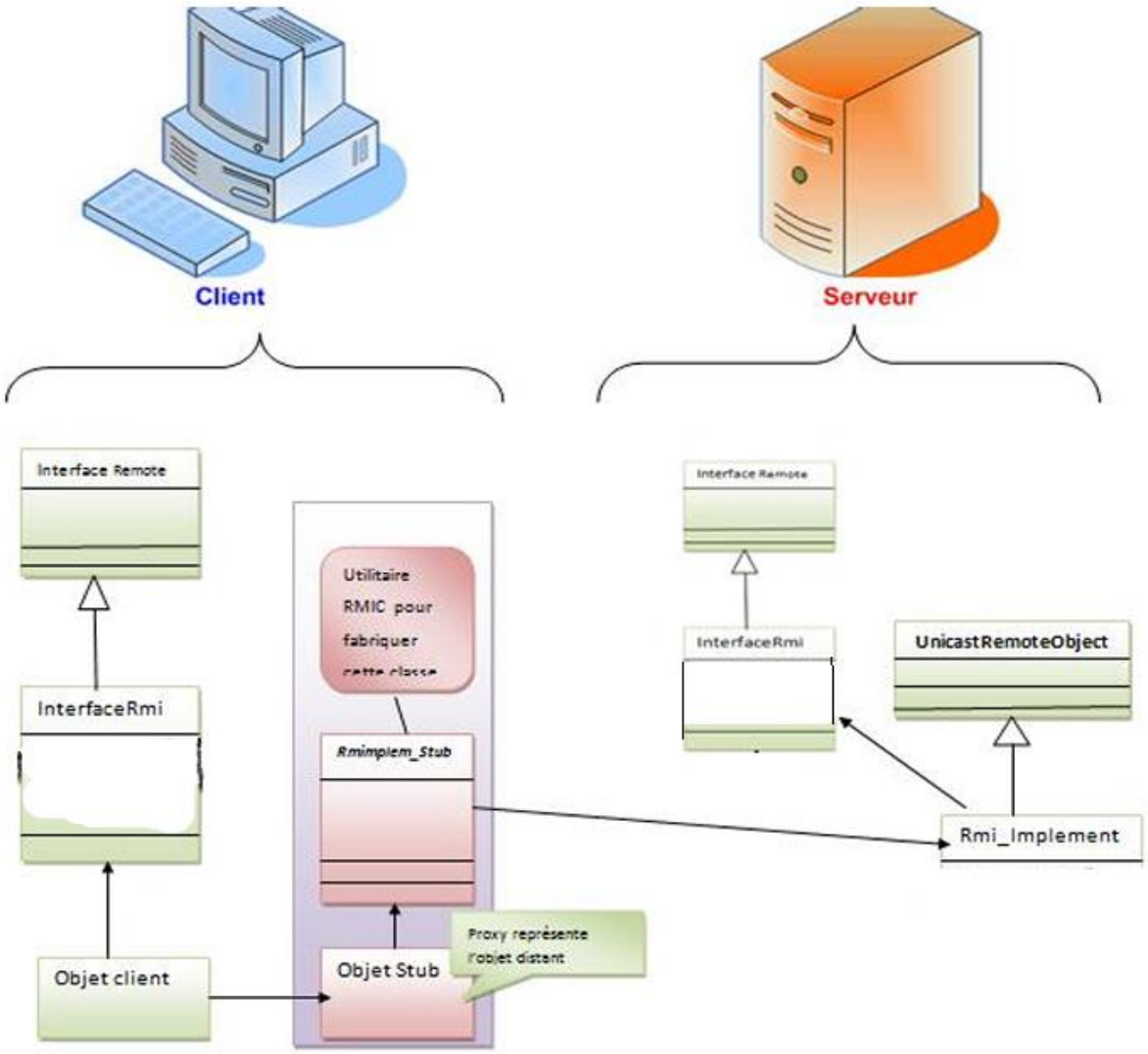
• L'annuaire est prêt à donner des références à l’objet Serveur.

• **Coté Client** L'objet client fait appel à l'annuaire (rmiregistry) en utilisant la classe Naming pour localiser l'objet serveur (méthode lookup).

• L'annuaire délivre une copie du stub.

• L’objet stub est installé et sa référence est retournée au client.

• Le client effectue l'appel à l’objet serveur par appel à l’objet stub. [



Année Universitaire : 2023-2024

Etude du Projet :

Un carnet d'adresses est un outil utilisé pour stocker et organiser les informations de contact des personnes que vous connaissez. Il peut s'agir d'un carnet physique ou d'un logiciel numérique.

Fonctions principales d'un carnet d'adresses:

- Stocker des informations de contact: Cela inclut généralement le nom, l'adresse e-mail, le numéro de téléphone et id pour chaque contact.
- Organiser les contacts: Les carnets d'adresses permettent de trier et de filtrer les contacts par différents critères, tels que le nom, id pour que chaque contact va avoir son unique adresse.
- Rechercher des contacts: La fonction de recherche permet de trouver rapidement un contact spécifique parmi une grande liste.
- Recherche des contacts par ID : cette fonction permet de trouver un contact exacte puisque chaque contact va avoir son unique id.
- Modifier ou supprimer un contact : De nombreux carnets d'adresses permettent de modifier les informations mais pad ID ou bien de supprimer directement.
- Récupération la liste de tous les contacts:

Types de carnets d'adresses:

- Carnets d'adresses physiques: Ce sont des carnets en papier où les informations de contact sont écrites à la main.
- Carnets d'adresses numériques: Ce sont des logiciels qui stockent les informations de contact sous forme électronique. Ils peuvent être intégrés à des systèmes d'exploitation, à des messageries électroniques ou à des applications de gestion des contacts dédiées.
- Carnets d'adresses en ligne: Ce sont des services web qui permettent de stocker et d'accéder aux informations de contact en ligne. Ils sont accessibles depuis n'importe quel appareil avec une connexion internet.

Spécification des besoins :

Les services proposés par notre application se résument dans les lignes suivantes :

• Cote serveur :

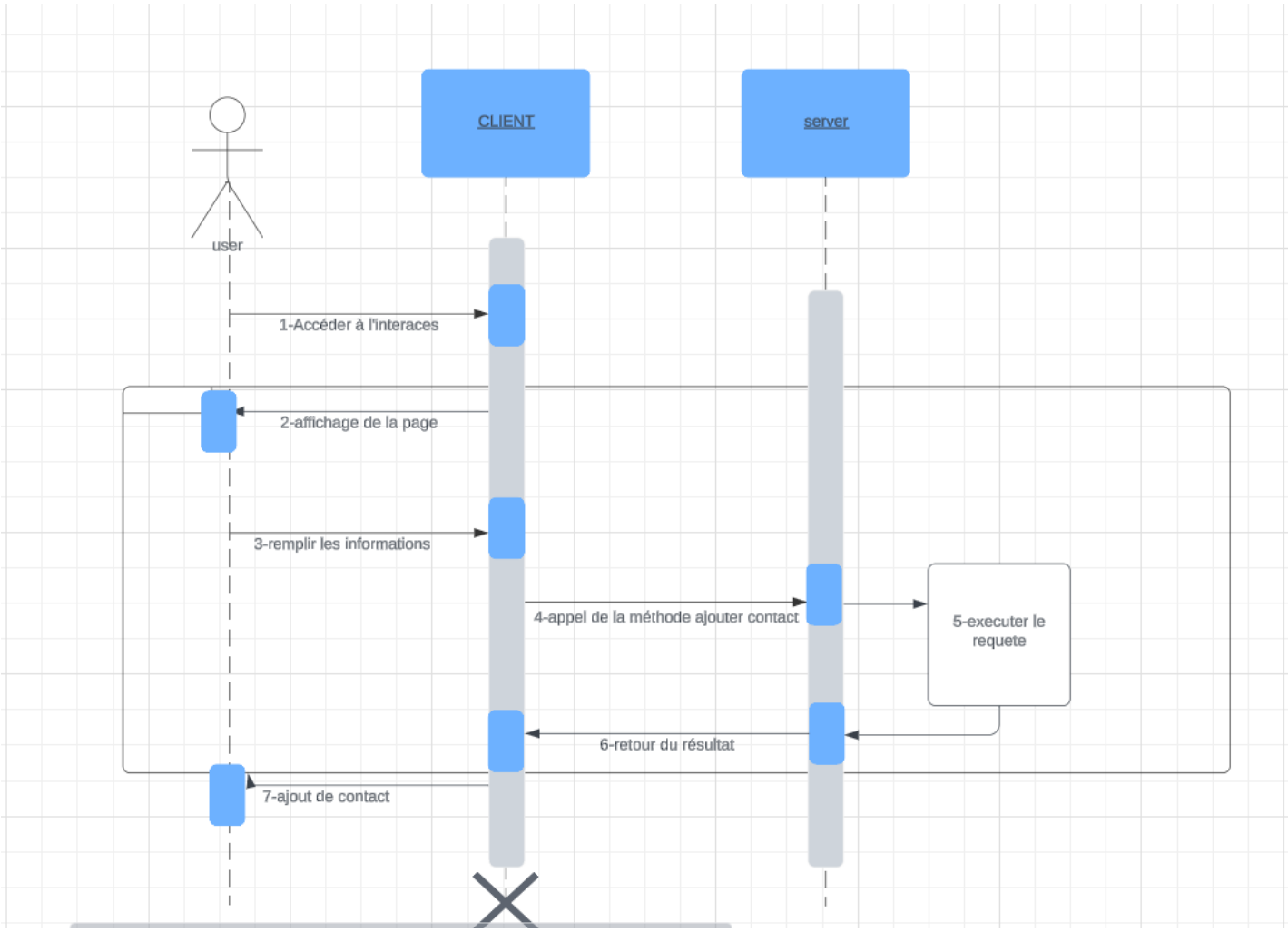
• **La gestion des adresses :** ajouter contact en remplissent ses information personnelles, le modifier ou le supprimer en générale traite les requetes demander par le client.

• Cote client:

Alors notre système doit répondre aux éxigences suivantes :

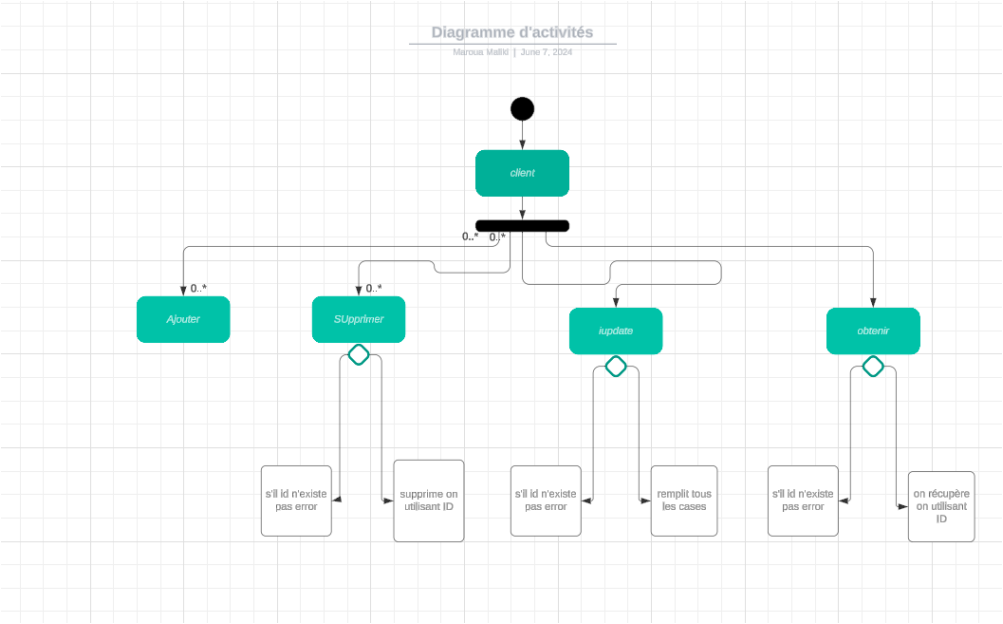
- **Gestion des contacts:**
  - Le système doit permettre aux utilisateurs d'ajouter de nouveaux contacts.
  - Le système doit exiger la saisie d'informations personnelles lors de l'ajout d'un nouveau contact.
  - Le système doit permettre aux utilisateurs de modifier les informations des contacts existants.
  - Le système doit permettre aux utilisateurs de supprimer des contacts existants.
- **Recherche de contacts:**
  - Le système doit permettre aux utilisateurs de rechercher des contacts par leur nom ou ID.

Diagramme de séquences :



le diagramme de séquence illustre efficacement les étapes clés de la création d'un nouveau compte utilisateur dans l'application. Il met en évidence les interactions entre le client, le serveur , offrant une compréhension claire de la séquence des événements et des messages échangés.

Diagramme d’activité :



Le diagramme représente un diagramme d'activités, qui est un outil utilisé pour modéliser les processus métier. Le processus représenté dans ce diagramme est le processus de gestion des contacts.

- **Ajouter un contact**: Cette étape consiste à ajouter un nouveau contact à la base de données.
- **Supprimer un** contact: Cette étape consiste à supprimer un contact de la base de données.
- **Mettre à jour les informations d'un** contact: Cette étape consiste à mettre à jour les informations d'un contact dans la base de données.
- **Obtenir les informations d'un contact**: Cette étape consiste à obtenir les informations d'un contact à partir de la base de données.

le diagramme représente un processus de gestion des conatcts qui permet aux utilisateurs d'ajouter, de supprimer, de mettre à jour et d'obtenir les informations des contacts.

Implementation :

Ce code Java implémente une application client-serveur pour la gestion de carnet d'adresse. Le serveur gère les méthodes de l'ajout, de la mise à jour, de la suppression et de la recherche de contacts. Le client fournit une interface graphique conviviale pour l'utilisateur afin d'interagir avec le serveur.

Le code utilise l'API Java RMI (Remote Method Invocation) pour la communication entre le client et le serveur. Cela permet au client d'invoquer des méthodes sur le serveur comme s'il s'agissait d'objets locaux.

Le code se compose de 4 parties : RemoteInterfac,Contact,Serveur,ClientGUI.

1. **RemoteInterface (fichier RemoteInterface.java) :**
   L'interface `RemoteInterfac` définit les méthodes distantes que le serveur doit implémenter pour communiquer avec les clients. Ces méthodes permettent aux clients d'effectuer des opérations sur les contacts stockés sur le serveur.

```java
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;
import java.util.ArrayList;
public interface  RemoteInterfac extends Remote{
        void addContact(String nom,String email, String telephone)throws RemoteException;
        void updateContact(Contact contact) throws RemoteException;
        void deleteContact(int id) throws RemoteException;
        List<Contact> searchContact(String nom) throws RemoteException;
        List<Contact> getAllContacts()throws RemoteException;
        Contact getContactById(int id) throws RemoteException;
}
```

2. **Class Contact (fichier Contact.java) :**

La classe `Contact` représente une entité "Contact" avec ses attributs et méthodes correspondants. Elle encapsule les informations d'un contact individuel. La classe `Contact` implémente l'interface `Serializable` pour permettre la sérialisation et le transfert d'objets `Contact` entre le client et le serveur via RMI. Les attributs privés sont utilisés pour encapsuler les données du contact et contrôler l'accès à ces données via les méthodes `get` et `set`.

```java
    public Contact(int id, String nom, String email, String telephone) {
        this.id = id;
        this.nom = nom;
        this.email = email;
        this.telephone = telephone;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getTelephone() {
        return telephone;
    }
    public void setTelephone(String telephone) {
        this.telephone = telephone;
    }
}
```

3. Serveur (fichier Serveur.java) :

La classe `Serveur` implémente l'interface `RemoteInterfac` et gère la logique métier de l'application. Elle stocke les contacts dans une liste, gère l'attribution d'ID uniques et implémente les méthodes distantes définies dans l'interface.

Attributs :

- **`contacts`**: Une liste de type `Contact` pour stocker tous les contacts gérés par le serveur.
- **`nextId`**: Un entier utilisé pour générer des ID uniques pour chaque nouveau contact.

Méthodes implémentées de l'interface `RemoteInterfac` :

- **`addContact(String nom, String email, String telephone)`**:
  - Crée un nouvel objet `Contact` avec les informations fournies.

- **`updateContact(Contact contact)`:**
  - Recherche le contact avec l'ID fourni dans le paramètre `contact`.
  - Si le contact est trouvé, met à jour ses informations avec les nouvelles valeurs du paramètre `contact`.
  - Si le contact n'est pas trouvé, lance une exception `RemoteException`.
- **`deleteContact(int id)`:**
  - Recherche le contact avec l'ID spécifié dans la liste `contacts`.
  - Si le contact est trouvé, le supprime de la liste.
  - Si le contact n'est pas trouvé, lance une exception `RemoteException`.
- **`searchContact(String nom)`:**
  - Parcourt la liste `contacts` et recherche les contacts dont le nom contient le terme de recherche spécifié.
  - Retourne une nouvelle liste contenant tous les contacts correspondants à la recherche.
- **`getAllContacts()`:**
  - Retourne une copie de la liste `contacts` contenant tous les contacts stockés sur le serveur.
- **`getContactById(int id)`:**
  - Recherche le contact avec l'ID spécifié dans la liste `contacts`.
  - Si le contact est trouvé, le retourne.
  - Si le contact n'est pas trouvé, retourne `null`.

Méthode main:

- Crée une instance de la classe `Serveur`.
- Définit un port RMI pour l'enregistrement du serveur.
- Enregistre le serveur auprès du registre RMI.
- Affiche un message indiquant que le serveur est prêt à recevoir des requêtes des clients.

```
public static void main(String[] args) {
      try {
          RemoteInterfac monservice = new Serveur();
         Registry registry= LocateRegistry.createRegistry(1099);
          registry.rebind("monservice", monservice);
          System.out.println("Serveur RPC EST PRET");
      } catch (Exception e) {
          e.printStackTrace();
      }
  }
```

4. ClientGUI (fichier ClientGUI.java) :

La classe `ClientGUI` implémente l'interface graphique du client et permet à l'utilisateur d'interagir avec le serveur pour gérer les contacts.

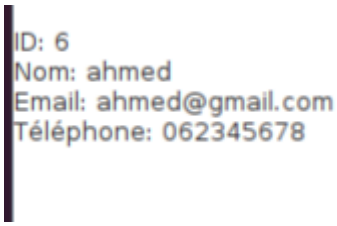
Attributs :

- `monservice`: Une variable de type `RemoteInterfac` qui représente la connexion au serveur distant.
- Champs graphiques pour la saisie des informations du contact (nom, email, téléphone, ID).
- Champ texte pour afficher les résultats des opérations (liste des contacts, messages d'erreur, etc.).

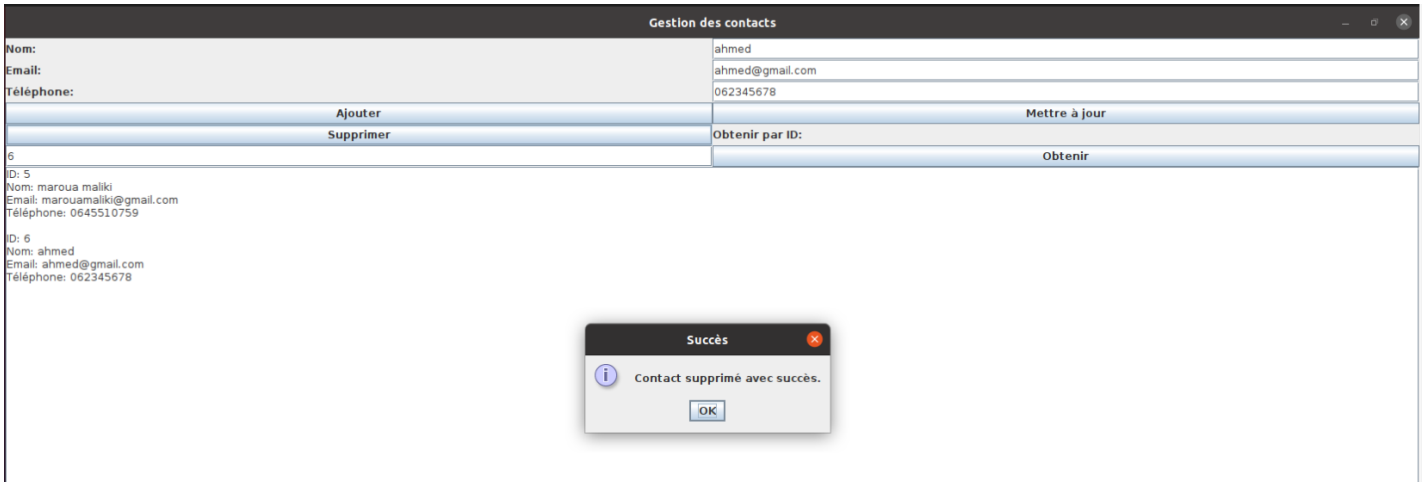
Méthodes :

- **`initComponents()`:**
  - Initialise les composants de l'interface graphique (champs de saisie, boutons, zone de texte).
  - Définit les actions des boutons pour déclencher les différentes opérations (ajouter, mettre à jour, supprimer, rechercher).
- **`connectToServer()`:**
  - Tente de se connecter au registre RMI et de récupérer l'objet `RemoteInterfac` du serveur.
  - Affiche un message de succès ou d'erreur en cas de problème de connexion.
- **`addContact()`:**
  - Récupère les informations du formulaire de saisie du contact (nom, email, téléphone).
  - Appelle la méthode `addContact` du serveur pour ajouter le nouveau contact.
  - Affiche un message de succès ou d'erreur en fonction du résultat de l'opération.
- **`updateContact()`:**
  - Récupère les informations du formulaire de saisie du contact (nom, email, téléphone, ID).
  - Crée un objet `Contact` avec les informations récupérées.
  - Appelle la méthode `updateContact` du serveur pour mettre à jour le contact.
  - Affiche un message de succès ou d'erreur en fonction du résultat de l'opération.
- **`deleteContact()`:**
  - Récupère l'ID du contact à supprimer du formulaire de saisie.
  - Appelle la méthode `deleteContact` du serveur pour supprimer le contact.
  - Affiche un message de succès ou d'erreur en fonction du résultat de l'opération.
- **`getContactById()`:**
  - Récupère l'ID du contact à rechercher du formulaire de saisie.
  - Appelle la méthode `getContactById` du serveur pour récupérer le contact.
  - Affiche les informations du contact trouvé ou un message d'erreur si aucun contact n'est trouvé.
- **`refreshContactList()`:**
  - Appelle la méthode `getAllContacts` du serveur pour

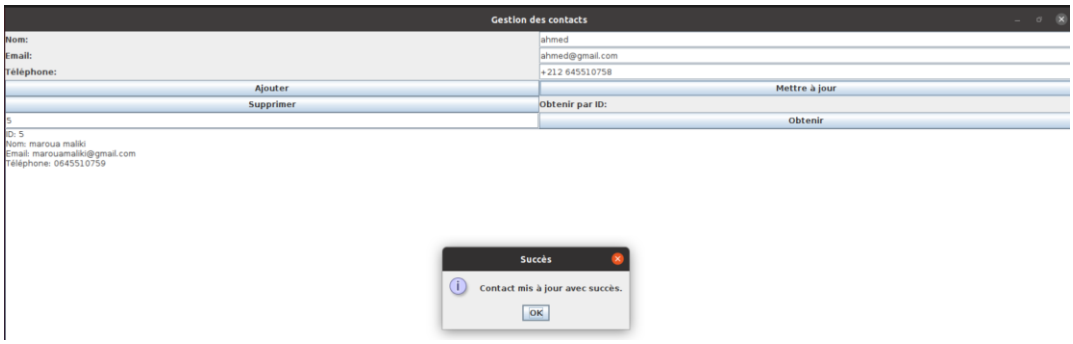
->Ajouter un contact :



->supprimer un contact :



->modifier le contact :



->obtenir un contact :

CONCLUSION :

En résumé, ce code Java fournit une base solide pour une application de gestion de carnet d'adresse fonctionnelle et évolutive. Il démontre une compréhension des concepts de programmation orientée objet, de la communication distribuée.