REPUBLIQUE DU CAMEROUN
Paix – Travail – Patrie
.....
ECOLE NATIONALE SUPERIEURE DES POSTES, DES TELECOMMUNICATIONS ET DES TIC
.....

REPUBLIC OF CAMEROON
Peace – Work – Fatherland
.....
NATIONAL ADVANCED SCHOOL OF POSTS, TELECOMMUNICATIONS AND ICT
.....

CYCLE : INGENIEUR DES TRAVAUX DES TELECOMMUNICATIONS / CLASSE : ITT3IR

Réalisation d'une Application De Monitoring Réseau et de Commande Distante en C#

Projet effectué dans le cadre du cours de Programmation et applications réseaux

Rédigé Par :

SIMO WAFFO BREL MOREL
PIANTA PINDO CHIRANEL
NGONO NGONO CECILE
SANGUEN BATANGA
NKANWA JOEL

Elève ingénieur des travaux des télécommunications en troisième année (IR)

A l'Ecole Nationale Supérieure des Postes, des Télécommunications et des TIC

Sous l'Encadrement de :

Dr. NZEBOP NDENOKA GERARD

Année academique: 2025-2026

Application de Surveillance et d'Administration Réseau

Table des Matières

1. Introduction

1.1 Présentation du Projet

NetAdmin Pro est une application client-serveur développée en .NET 10 pour la surveillance et l'administration à distance de machines en réseau. Le système permet une collecte en temps réel des informations système (matériel, processus, performances) et offre une interface graphique WPF centralisée pour la supervision et le contrôle des clients connectés.

1.2 Objectifs

- Surveillance en temps réel des ressources système (CPU, RAM, stockage)
- Gestion centralisée de multiples machines clientes
- Système d'authentification sécurisé avec gestion des rôles
- Interface utilisateur intuitive avec visualisation graphique des données
- Notifications et alertes en temps réel via Discord
- Persistance des données et traçabilité des actions

1.3 Contexte Technologique

Le projet utilise les technologies Microsoft les plus récentes, notamment .NET 10, Entity Framework Core pour la persistance, et WPF pour l'interface graphique. L'architecture adopte une approche moderne avec séparation des préoccupations et respect des principes SOLID.

2. Architecture Générale

2.1 Vue d'Ensemble

NetAdmin Pro adopte une architecture client-serveur à trois couches avec une séparation stricte entre la logique métier, la présentation et la persistance des données. L'application est divisée en trois projets distincts permettant une maintenance et une évolution facilitées.

2.2 Structure des Projets

2.2.1 NetAdmin.Client

Le projet client est responsable de la collecte des informations système et de leur transmission au serveur. Il implémente les fonctionnalités suivantes :

- Collecte des métriques système (CPU, RAM, disques)
- Énumération et surveillance des processus
- Communication TCP avec le serveur
- Gestion de l'authentification et des tokens JWT
- Heartbeat automatique pour maintenir la connexion

2.2.2 NetAdmin.Server

Le serveur constitue le cœur de l'application avec l'interface graphique et la logique métier :

- Serveur TCP pour gérer les connexions clientes
- Interface WPF avec MVVM (Model-View-ViewModel)
- Services d'authentification et d'autorisation
- Gestion de la base de données via Entity Framework Core
- Système de notifications Discord
- Audit et traçabilité des actions

2.2.3 NetAdmin.Shared

Le projet partagé contient les définitions communes utilisées par le client et le serveur :

- Modèles de données (DTOs)
- Protocoles de communication réseau
- Constantes et énumérations
- Utilitaires partagés

2.3 Flux de Communication

Le système implémente un protocole de communication bidirectionnel basé sur TCP avec les étapes suivantes :

1. Le client initie une connexion TCP vers le serveur sur le port configuré (par défaut 5000)
2. Authentification du client avec envoi des identifiants et réception du token JWT
3. Enregistrement du client auprès du serveur avec transmission des informations système
4. Envoi périodique de paquets de données (métriques, processus, heartbeat)
5. Le serveur traite, persiste et affiche les données en temps réel
6. Des commandes peuvent être envoyées du serveur vers le client pour exécuter des actions

3. Conception Détaillée

3.1 Architecture Client

3.1.1 Collecte des Informations Système

Le client utilise plusieurs APIs Windows et bibliothèques .NET pour collecter les informations système. Les composants principaux incluent des services de monitoring qui interrogent périodiquement le système d'exploitation pour récupérer les métriques CPU, RAM, et disque. La collecte est optimisée pour minimiser l'impact sur les performances de la machine surveillée.

3.1.2 Communication Réseau

La communication entre le client et le serveur est établie via des sockets TCP. Le client maintient une connexion persistante avec un mécanisme de reconnexion automatique en cas de perte de connexion. Les données sont sérialisées au format JSON avant transmission pour faciliter l'interopérabilité et le débogage.

3.1.3 Gestion de Configuration

Le client lit sa configuration depuis un fichier appsettings.json qui contient les paramètres de connexion au serveur, les intervalles de collecte de données, et les options d'authentification. Cette approche permet une configuration flexible sans recompilation.

3.2 Architecture Serveur

3.2.1 Serveur TCP

Le serveur TCP écoute sur un port configurable et accepte les connexions entrantes des clients. Il utilise un modèle asynchrone pour gérer simultanément plusieurs clients sans bloquer le thread principal. Chaque connexion cliente est gérée dans une tâche distincte avec sa propre session.

3.2.2 Pattern MVVM

L'interface graphique WPF implémente le pattern Model-View-ViewModel pour séparer la logique de présentation de l'interface utilisateur. Les ViewModels exposent des commandes et des propriétés observables que les vues consomment via le data binding. Cette architecture facilite les tests unitaires et la maintenabilité.

3.2.3 Services Métier

Le serveur expose plusieurs services métier injectés via le conteneur de dépendances :

- AuthenticationService : Gestion de l'authentification et génération des tokens JWT
- SessionManager : Suivi des sessions actives et gestion de l'état des clients
- AuditService : Enregistrement des actions pour la traçabilité
- DiscordNotificationService : Envoi de notifications via webhook

4. Sécurité et Authentification

4.1 Système d'Authentification

4.1.1 Hachage des Mots de Passe

Les mots de passe utilisateurs sont hachés avec l'algorithme BCrypt qui intègre automatiquement un salt unique pour chaque mot de passe. BCrypt est spécifiquement conçu pour être résistant aux attaques par force brute grâce à son facteur de coût ajustable qui ralentit le calcul. Le système n'enregistre jamais les mots de passe en clair et les compare uniquement via la fonction de vérification BCrypt.

4.1.2 Tokens JWT

Après authentification réussie, le serveur génère un token JWT (JSON Web Token) signé avec l'algorithme HMAC-SHA256. Le token contient les claims suivants : identifiant utilisateur, nom d'utilisateur, email, rôle, et nom complet. Les tokens ont une durée de vie configurable (par défaut 60 minutes) et sont validés à chaque requête. La signature cryptographique garantit que les tokens ne peuvent pas être falsifiés sans la clé secrète du serveur.

4.1.3 Refresh Tokens

Pour améliorer l'expérience utilisateur, le système implémente des refresh tokens qui permettent de renouveler les tokens JWT expirés sans redemander les identifiants. Les refresh tokens sont stockés en base de données avec une durée de vie plus longue (par défaut 7 jours) et peuvent être révoqués en cas de besoin. Le client peut automatiquement rafraîchir son token avant expiration si la configuration le permet.

4.2 Gestion des Rôles et Autorisations

4.2.1 Hiérarchie des Rôles

Le système définit quatre rôles avec des permissions croissantes :

- **Viewer :** Consultation en lecture seule des informations système
- **Operator :** Peut effectuer des actions basiques sur les clients
- **Supervisor :** Peut gérer les clients et configurer certains paramètres
- **Admin :** Accès complet incluant la gestion des utilisateurs et la configuration système

4.2.2 Comptes par Défaut

Le système initialise quatre comptes par défaut lors du premier démarrage :

Nom d'utilisateur	Mot de passe	Rôle
admin	Admin@123!	Administrateur
supervisor	Supervisor@123!	Superviseur
operator	Operator@123!	Opérateur
viewer	Viewer@123!	Observateur

4.3 Protection Contre les Attaques

4.3.1 Limitation des Tentatives de Connexion

Pour se protéger contre les attaques par force brute, le système implémente un délai progressif après chaque échec de connexion. Après plusieurs tentatives échouées depuis la même adresse IP, des délais croissants sont imposés avant d'autoriser une nouvelle tentative. Cette mesure décourage les attaques automatisées sans impacter significativement les utilisateurs légitimes.

4.3.2 Audit et Traçabilité

Toutes les actions sensibles sont enregistrées dans un journal d'audit persistant en base de données. Chaque entrée d'audit contient l'identifiant utilisateur, l'action effectuée, l'horodatage, et l'adresse IP source. Ces logs permettent de détecter des comportements suspects et de répondre aux exigences de conformité.

5. Persistance et Base de Données

5.1 Technologie de Persistance

NetAdmin Pro utilise SQLite comme système de gestion de base de données pour sa simplicité de déploiement et son efficacité sur les petites et moyennes installations. Entity Framework Core agit comme ORM (Object-Relational Mapping) pour abstraire les interactions avec la base de données et faciliter la maintenance du code.

5.2 Modèle de Données

5.2.1 Table Users

Stocke les informations des utilisateurs autorisés à se connecter au serveur :

- Id : Identifiant unique (clé primaire)
- Username : Nom d'utilisateur (unique)
- Email : Adresse email (unique)
- PasswordHash : Hash BCrypt du mot de passe
- Role : Rôle attribué (Admin, Supervisor, Operator, Viewer)
- FullName : Nom complet de l'utilisateur
- CreatedAt : Date de création du compte
- IsActive : Indicateur d'activation du compte

5.2.2 Table AuthTokens

Gère les refresh tokens pour le renouvellement automatique de l'authentification :

- Id : Identifiant unique
- UserId : Référence vers l'utilisateur
- Token : Token de rafraîchissement
- ExpiresAt : Date d'expiration
- CreatedAt : Date de création
- IsRevoked : Indicateur de révocation

5.2.3 Table ClientHosts

Enregistre les machines clientes connectées au serveur :

- Id : Identifiant unique

- HostName : Nom de la machine
- IpAddress : Adresse IP du client
- OperatingSystem : Système d'exploitation
- LastSeen : Dernière activité enregistrée
- Status : État de connexion (Online, Offline)

5.2.4 Table MetricLogs

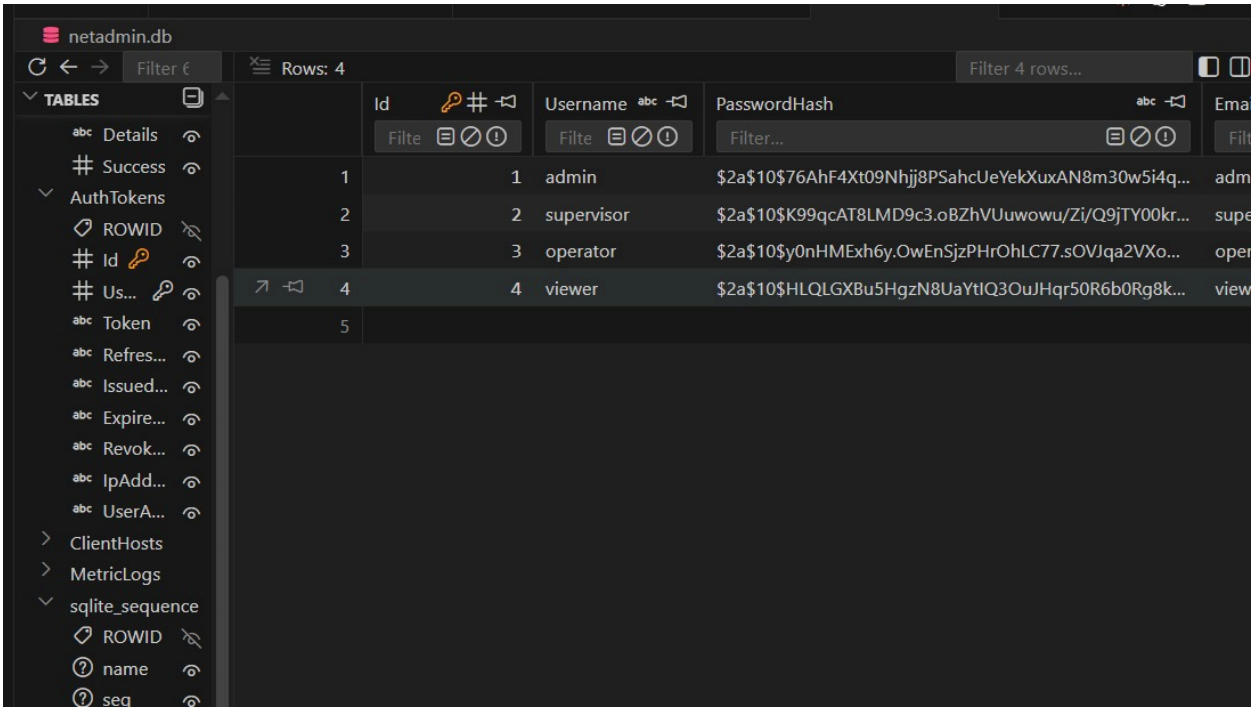
Conserve l'historique des métriques système collectées :

- Id : Identifiant unique
- ClientHostId : Référence vers le client
- Timestamp : Horodatage de la collecte
- CpuUsage : Utilisation CPU en pourcentage
- MemoryUsage : Utilisation RAM en pourcentage
- DiskUsage : Utilisation disque en pourcentage

5.2.5 Table AuditLogs

Enregistre toutes les actions effectuées dans le système pour la traçabilité et la conformité :

- Id : Identifiant unique
- UserId : Utilisateur ayant effectué l'action
- Action : Description de l'action
- Timestamp : Date et heure de l'action
- IpAddress : Adresse IP source
- Details : Détails supplémentaires au format JSON



5.3 Migrations et Évolution du Schéma

Entity Framework Core gère les migrations de base de données, permettant d'évoluer le schéma de manière contrôlée au fil des versions. Chaque modification du modèle de données génère une migration qui peut être appliquée automatiquement au démarrage ou manuellement via des commandes CLI. Cette approche garantit la cohérence entre le code et la structure de la base de données.

6. Interface Utilisateur

6.1 Architecture WPF

L'interface graphique du serveur est développée avec Windows Presentation Foundation (WPF), une technologie Microsoft permettant de créer des applications desktop riches et performantes. L'architecture suit strictement le pattern MVVM pour séparer la logique métier de la présentation et faciliter les tests automatisés.

6.2 Composants Principaux

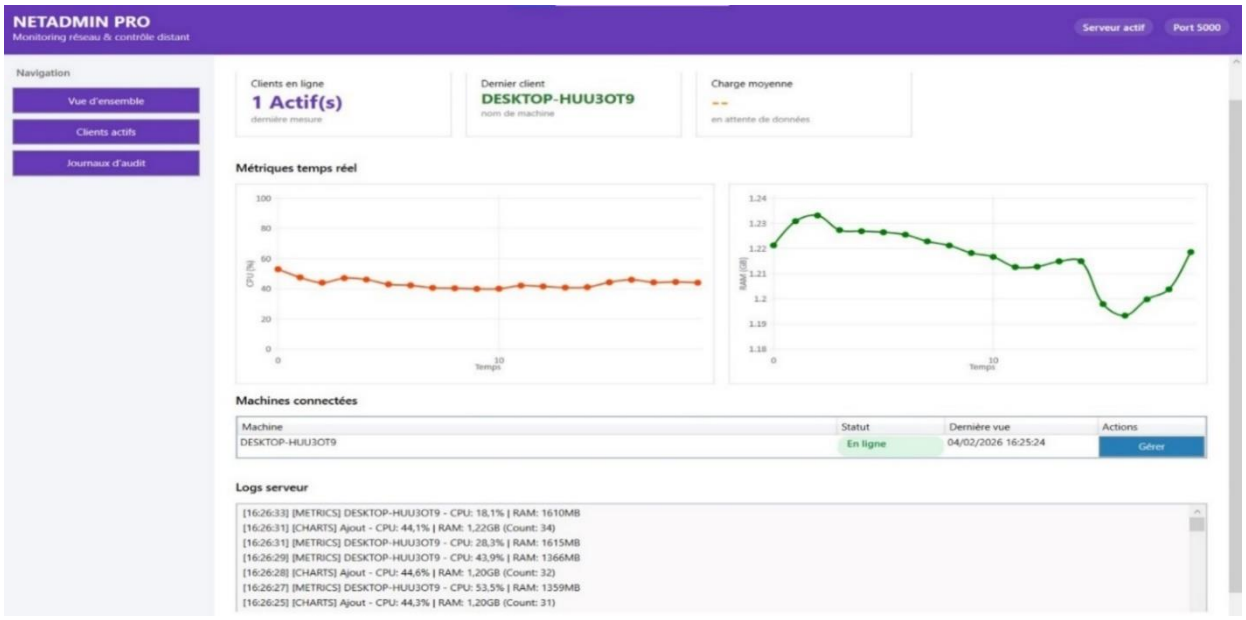
6.2.1 Page de connexion :

Pour avoir accès à l'application nous avons mis sur pied une interface de connexion solide pour éviter que n'importe qui ne puisse avoir accès au tableau de bord.

6.2.1 Tableau de Bord

Le tableau de bord principal affiche une vue d'ensemble de l'état du système :

- Nombre de clients connectés en temps réel
- Dernière machine connectée
- Statistiques globales d'utilisation des ressources
- Alertes et notifications importantes



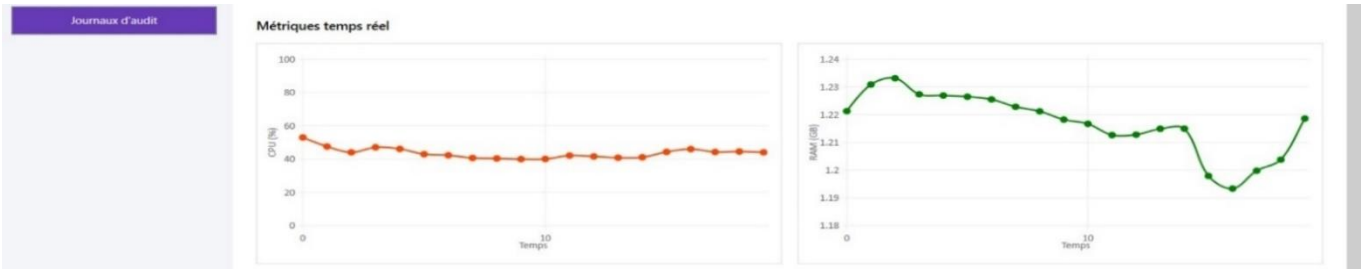
6.2.2 Liste des Clients

Un tableau affiche tous les clients enregistrés avec les informations suivantes :

- Nom de la machine
- Adresse IP
- Système d'exploitation
- État de connexion (en ligne/hors ligne)

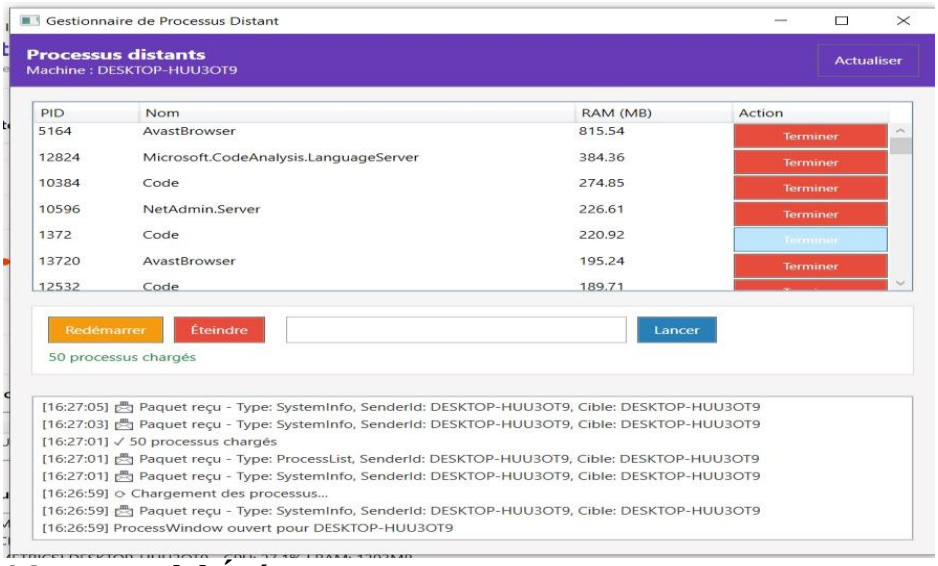
6.2.3 Graphiques de Performance

Des graphiques en temps réel visualisent l'évolution des métriques système pour chaque client sélectionné. Les graphiques affichent l'historique des 60 dernières minutes pour les métriques CPU et RAM, permettant d'identifier rapidement les tendances et les anomalies.



6.2.4 Liste des processus systèmes

Notre application a été conçu pour afficher uniquement 50 processus actifs de la machine distante.



6.2.5 Journal d'Événements

Un journal en temps réel affiche tous les événements du serveur incluant les connexions, déconnexions, erreurs, et actions administratives. Les entrées sont horodatées et peuvent être filtrées par type ou niveau de sévérité pour faciliter le diagnostic.

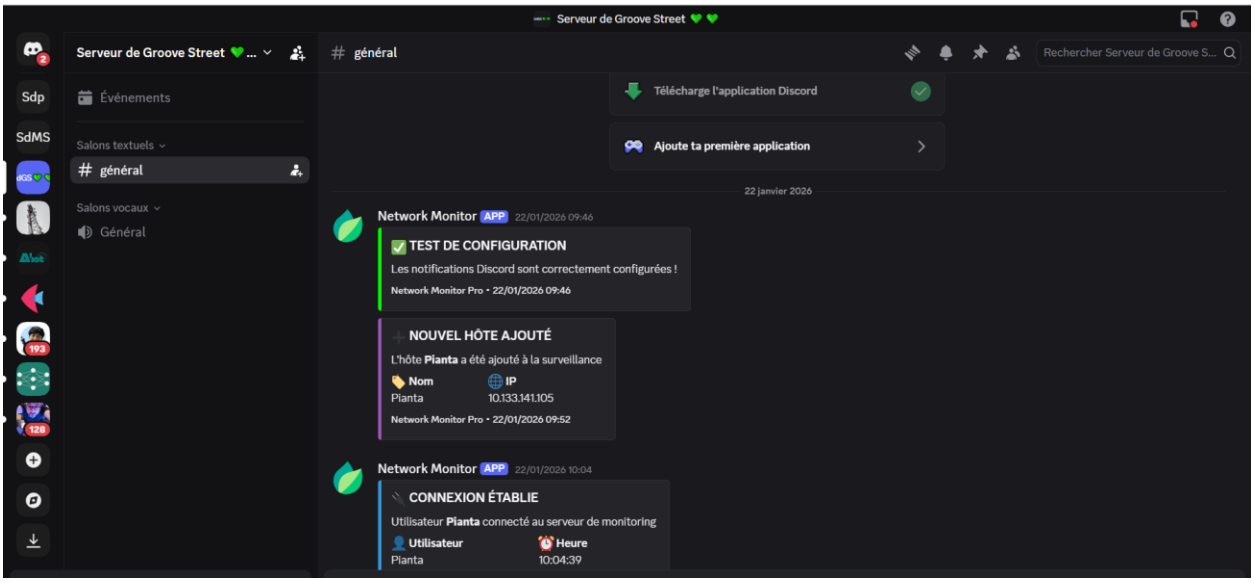
6.3 Data Binding et Observables

L'interface utilise intensivement le data binding WPF pour synchroniser automatiquement l'affichage avec les données. Les ViewModels implémentent l'interface INotifyPropertyChanged pour notifier l'interface des changements d'état. Les collections observables (ObservableCollection) permettent de mettre à jour automatiquement les listes affichées lors de l'ajout ou de la suppression d'éléments.

7. Notifications Discord

7.1 Intégration Webhook

NetAdmin Pro peut envoyer des notifications en temps réel vers un canal Discord via un webhook. Cette fonctionnalité permet aux administrateurs de recevoir des alertes importantes sans avoir à surveiller constamment l'interface graphique. Le système utilise l'API REST de Discord pour poster des messages formatés avec des embeds riches.



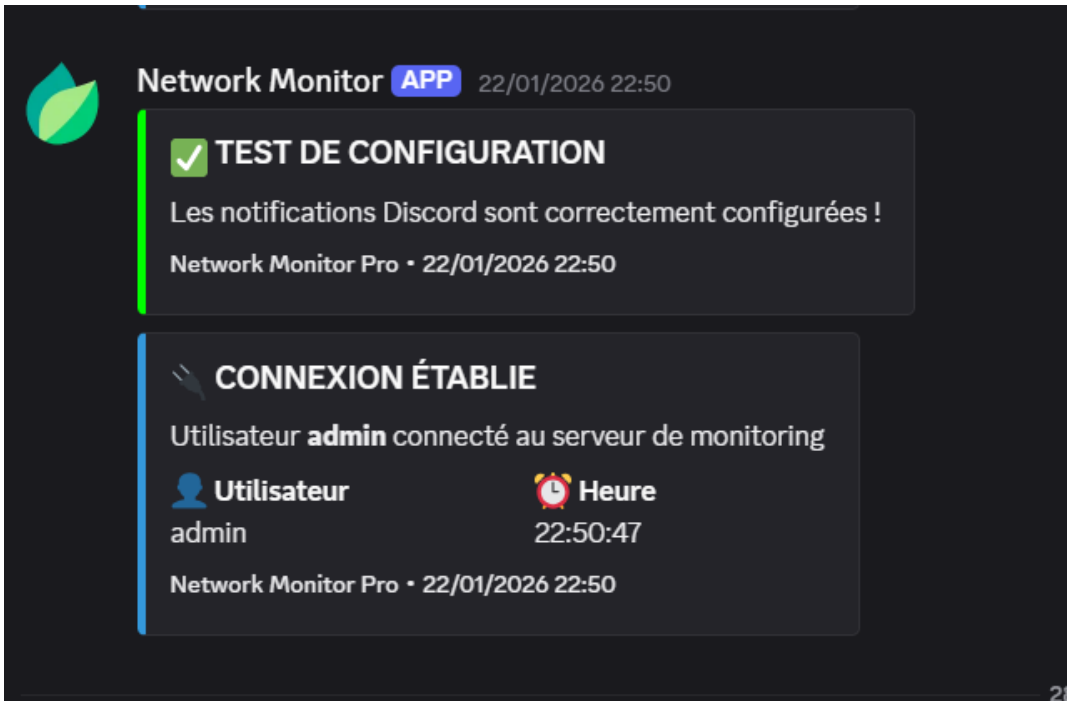
7.2 Types de Notifications

7.2.1 Démarrage du Serveur

Une notification est envoyée automatiquement lors du démarrage du serveur, incluant la date et l'heure de démarrage, le nom du serveur configuré, et le port d'écoute TCP. Cette notification confirme que le système est opérationnel et prêt à accepter des connexions.

7.2.2 Connexion de Client

Chaque nouvelle connexion cliente génère une notification contenant le nom de la machine, son adresse IP, et le système d'exploitation détecté. Ces alertes permettent de détecter rapidement des connexions inattendues ou non autorisées.



7.2.3 Enregistrement de Client

Lorsqu'un nouveau client s'enregistre pour la première fois, une notification détaillée est envoyée avec toutes les informations système collectées. Cela permet de documenter l'ajout de nouvelles machines au parc surveillé.

7.3 Configuration

Les notifications Discord se configurent dans le fichier appsettings.json du serveur :

- **Enabled :** Active ou désactive les notifications (true/false)
- **WebhookUrl :** URL du webhook Discord à utiliser
- **Username :** Nom affiché pour le bot (par défaut NetAdmin Pro)
- **AvatarUrl :** URL de l'avatar du bot (optionnel)
- **ServerName :** Nom du serveur à inclure dans les messages

8. Installation et Déploiement

8.1 Prérequis Système

8.1.1 Serveur

- Windows 10 ou Windows 11 (pour l'interface WPF)
- .NET 10 SDK installé
- 4 Go de RAM minimum (8 Go recommandés)
- Port TCP disponible (par défaut 5000)

8.1.2 Client

- Windows 10 ou supérieur
- .NET 10 Runtime installé
- Connexion réseau vers le serveur

8.2 Procédure d'Installation

8.2.1 Clonage du Dépôt

Récupérer le code source depuis le dépôt GitHub :

```
git clone https://github.com/simsbm/Projet_ProgRx.git

cd Projet_ProgRx
```

8.2.2 Restauration des Dépendances

```
dotnet restore
```

8.2.3 Compilation du Projet

```
dotnet build
```

8.3 Configuration Initiale

8.3.1 Configuration Serveur

Éditer le fichier NetAdmin.Server/appsettings.json et adapter les paramètres :

- Modifier le secret JWT (minimum 32 caractères)
- Configurer le port d'écoute TCP si nécessaire
- Activer et configurer les notifications Discord si souhaité

8.3.2 Configuration Client

Éditer le fichier NetAdmin.Client/appsettings.json avec l'adresse du serveur et le port correspondant. Configurer les intervalles de collecte selon les besoins.

8.4 Démarrage

8.4.1 Lancement du Serveur

```
cd NetAdmin.Server

dotnet run
```

Le serveur initialise la base de données au premier démarrage et crée les comptes utilisateurs par défaut. L'interface graphique s'ouvre automatiquement.

8.4.2 Lancement du Client

```
cd NetAdmin.Client

dotnet run
```

Le client se connecte automatiquement au serveur configuré. Si les paramètres sont absents, ils peuvent être fournis en ligne de commande ou saisis interactivement.

9. Tests et Validation

9.1 Stratégie de Tests

Le projet suit une approche de tests à plusieurs niveaux pour garantir la qualité et la fiabilité du système. Les tests couvrent les composants individuels, l'intégration entre modules, et les scénarios de bout en bout.

9.2 Tests Unitaires

Les tests unitaires vérifient le comportement des composants isolés :

- Services d'authentification (génération et validation des tokens)
- Hachage et vérification des mots de passe
- Sérialization et désérialisation des paquets réseau
- Logique métier des ViewModels

9.3 Tests d'Intégration

Les tests d'intégration valident l'interaction entre composants :

- Communication client-serveur via TCP
- Persistance en base de données avec Entity Framework
- Flux d'authentification complet
- Notifications Discord

9.4 Tests Fonctionnels

Les tests fonctionnels vérifient les scénarios utilisateur de bout en bout :

- Connexion d'un nouveau client et collecte des données
- Visualisation des métriques dans l'interface graphique
- Gestion des sessions utilisateur
- Reconnexion automatique après déconnexion

9.5 Tests de Sécurité

Des tests spécifiques valident les aspects sécurité :

- Résistance aux tentatives de connexion par force brute
- Impossibilité de falsifier les tokens JWT

- Validation des autorisations basées sur les rôles
- Sécurité des mots de passe stockés

10. Maintenance et Support

10.1 Journalisation

10.1.1 Logs Serveur

Le serveur génère automatiquement un fichier de log au démarrage nommé netadmin.startup.log qui contient toutes les informations de diagnostic. Ce fichier est essentiel pour identifier les problèmes de démarrage, les erreurs de configuration, ou les exceptions non gérées.

10.1.2 Logs d'Audit

Toutes les actions sensibles sont enregistrées dans la table AuditLogs de la base de données. Ces logs peuvent être consultés directement en base ou via une interface d'administration future. Ils constituent une piste d'audit complète pour la conformité et l'investigation.

10.2 Diagnostic et Dépannage

10.2.1 Problèmes de Connexion

Si un client ne parvient pas à se connecter au serveur :

- Vérifier que le serveur est démarré et écoute sur le bon port
- Confirmer que le pare-feu autorise les connexions TCP entrantes
- Valider l'adresse IP et le port dans la configuration client
- Vérifier la connectivité réseau avec un ping ou telnet

10.2.2 Problèmes d'Authentification

En cas d'échec d'authentification :

- Vérifier que les identifiants fournis sont corrects
- Confirmer que le compte n'est pas désactivé (IsActive = true)
- Vérifier le secret JWT dans appsettings.json (doit correspondre)
- Consulter les logs pour identifier la cause exacte du rejet

10.2.3 Problèmes de Performance

Si le système devient lent ou consomme trop de ressources, plusieurs actions sont possibles : réduire la fréquence de collecte des métriques dans la configuration client, augmenter les ressources allouées au serveur, nettoyer la base de données des anciennes entrées MetricLogs, ou limiter le nombre de clients connectés simultanément.

10.3 Sauvegarde et Restauration

10.3.1 Sauvegarde de la Base de Données

La base de données SQLite est contenue dans un seul fichier (netadmin.db). Il suffit de copier ce fichier pour effectuer une sauvegarde complète. Il est recommandé de sauvegarder régulièrement ce fichier, particulièrement avant les mises à jour majeures.

10.3.2 Restauration

Pour restaurer une sauvegarde, arrêter le serveur et remplacer le fichier netadmin.db par la version sauvegardée. Au redémarrage, le serveur utilisera automatiquement les données restaurées. S'assurer que la version de la base correspond à la version de l'application pour éviter les problèmes de compatibilité.

11. Évolutions Futures

11.1 Fonctionnalités Planifiées

11.1.2 Gestion des Processus

Ajouter la possibilité de démarrer, arrêter ou redémarrer des processus sur les clients depuis le serveur. Cette fonction sera utile pour la gestion des services et des applications critiques.

11.1.3 Alertes Configurables

Implémenter un système d'alertes avec des seuils personnalisables. Les administrateurs pourront définir des limites (CPU > 90%, RAM > 85%) et recevoir des notifications lorsque ces seuils sont dépassés.

11.1.4 Rapports et Statistiques

Développer un module de génération de rapports permettant d'exporter des statistiques d'utilisation sur des périodes définies. Les rapports pourront être au format PDF ou Excel pour faciliter l'analyse et la présentation.

11.1.5 Support Multi-Plateforme

Étendre le support du client aux systèmes Linux et macOS en utilisant .NET multi-plateforme. Cela permettra de surveiller des environnements hétérogènes depuis une seule interface.

11.2 Améliorations Techniques

11.2.1 Migration vers WebSocket

Remplacer le protocole TCP actuel par WebSocket pour bénéficier d'une meilleure gestion de la connexion et d'une compatibilité accrue avec les proxies et pare-feu. WebSocket offre également une API standardisée facilitant l'intégration.

11.2.2 Interface Web

Développer une interface web avec Blazor ou ASP.NET MVC en complément de l'application WPF. Cela permettrait l'accès à distance depuis n'importe quel navigateur sans installation spécifique.

Conclusion

NetAdmin Pro représente une solution complète et moderne pour la surveillance et l'administration de machines en réseau. L'architecture client-serveur offre une séparation claire des responsabilités et facilite l'évolution du système. L'utilisation des technologies .NET 10 et Entity Framework Core garantit des performances optimales et une maintenance simplifiée.

La sécurité a été placée au cœur de la conception avec un système d'authentification robuste basé sur JWT et BCrypt. Le système de rôles permet une gestion fine des autorisations selon le principe du moindre privilège. Les mécanismes de traçabilité et d'audit assurent la conformité et facilitent les investigations en cas d'incident.

L'interface graphique WPF offre une expérience utilisateur intuitive avec des visualisations en temps réel des métriques système. Les notifications Discord permettent aux administrateurs de rester informés des événements importants sans surveillance constante de l'interface.

Le projet pose des bases solides pour de nombreuses évolutions futures. Les fonctionnalités planifiées telles que l'exécution de commandes à distance, les alertes configurables, et le support multi-plateforme élargiront considérablement les capacités du système. L'architecture modulaire facilitera l'ajout de ces nouvelles fonctionnalités sans nécessiter de refonte majeure.

NetAdmin Pro démontre qu'il est possible de créer une application d'administration réseau professionnelle avec les outils et frameworks Microsoft. Le code source ouvert permet à la communauté de contribuer et d'adapter le système à des besoins spécifiques. L'équipe de développement reste engagée dans l'amélioration continue du produit et le support des utilisateurs.

Annexes :

Annexe A : Configuration Complète du Serveur

Voici un exemple complet de fichier appsettings.json pour le serveur :

{ "JwtSettings": { "Secret": "your-super-secret-key-min-32-characters-for-security", "TokenExpirationMinutes": 60, "RefreshTokenExpirationDays": 7 }, "Database": { "ConnectionString": "Data Source=netadmin.db" }, "Server": { "Port": 5000, "MaxConnections": 100 }, "Discord": { "Enabled": false, "WebhookUrl": "", "Username": "NetAdmin Pro", "AvatarUrl": "", "ServerName": "" } }

Annexe B : Configuration Complète du Client

Voici un exemple complet de fichier appsettings.json pour le client :

{ "Server": { "Host": "127.0.0.1", "Port": 5000 }, "Authentication": { "AutoRefreshToken": true, "RefreshIntervalMinutes": 55 }, "Client": { "HeartbeatIntervalSeconds": 30, "ConnectTimeoutSeconds": 10 } }

Annexe C : Liens Utiles

- **Dépôt GitHub :** https://github.com/simsbm/Projet_ProgRx
- **Documentation .NET :** https://docs.microsoft.com/dotnet
- **Entity Framework Core :** https://docs.microsoft.com/ef/core
- **Discord Developer Portal :** https://discord.com/developers

Annexe D : Glossaire

Terme	Définition
JWT	JSON Web Token - Standard pour créer des tokens d'accès signés cryptographiquement
BCrypt	Fonction de hachage sécurisée spécialement conçue pour les mots de passe
MVVM	Model-View-ViewModel - Pattern architectural pour séparer logique et présentation
DTO	Data Transfer Object - Objet utilisé pour transférer des données entre couches
ORM	Object-Relational Mapping - Technique pour convertir données entre systèmes incompatibles
WPF	Windows Presentation Foundation - Framework Microsoft pour applications desktop
TCP	Transmission Control Protocol - Protocole de communication réseau fiable
EF Core	Entity Framework Core - ORM moderne pour .NET

Annexe E : Technologies et dépendances clés

Catégorie	Technologie / Dépendance	Description
Langage	C#	Langage de programmation principal.
Framework	.NET 10	Environnement d'exécution et bibliothèques de classes.
UI Serveur	WPF	Framework pour l'interface utilisateur graphique du serveur.
Base de Données	SQLite	Base de données relationnelle légère et embarquée.
ORM	Entity Framework Core	Mappage objet-relationnel pour l'accès aux données.
Authentification	JWT	Jetons d'authentification sécurisés.
Hachage Mots de Passe	BCrypt.Net-Next	Bibliothèque pour le hachage sécurisé des mots de passe.
Graphiques UI	LiveCharts.Wpf	Composants graphiques pour la visualisation des données.
Design UI	MaterialDesignThemes	Thème Material Design pour l'interface WPF.
Collecte Matériel	System.Diagnostics.PerformanceCounter, System.Management	API Windows pour la collecte de métriques système.
Sérialisation	System.Text.Json	Sérialisation/désérialisation JSON des paquets réseau.