RAPPORT DE PROJET

Gestion Matos 13

Application de gestion de matériel et de clients

C# WinForms · API PHP · MySQL

1. Introduction et contexte

Le projet « Gestion Matos 13 » consiste à développer une application de gestion de parc de matériel et de clients, destinée à un contexte de location d'équipements (matériel audiovisuel, image, son, éclairage). L'objectif est de fournir un outil simple permettant d'enregistrer les fiches clients et les fiches matériel, de suivre l'état et la disponibilité des équipements, et de gérer ces informations depuis une interface de bureau connectée à une base de données centralisée.

La problématique adressée est celle d'un suivi de matériel souvent réalisé de façon manuelle (tableur, papier), peu fiable dès que le volume d'équipements ou de clients augmente. L'application propose une solution structurée : une interface WinForms pour la saisie et la consultation, une API PHP faisant l'intermédiaire avec la base de données, et une base MySQL assurant la persistance des données.

Ce projet a été réalisé individuellement, dans un cadre académique, avec pour but de mettre en pratique une architecture client-serveur complète : interface graphique, couche d'accès aux données via une API REST, et base de données relationnelle.

2. Analyse des besoins

2.1 Fonctionnalités attendues

- Gérer les clients : créer, consulter, modifier et supprimer une fiche client (nom, prénom, email, téléphone, adresse).
- Gérer le matériel : créer, consulter, modifier et supprimer une fiche matérielle (référence, désignation, catégorie, quantité, prix par jour, état).
- Disposer d'un menu principal permettant d'accéder rapidement aux deux modules (Clients / Matériel).
- Centraliser les données dans une base MySQL accessible via une API, plutôt que de les stocker localement.
- Fournir un retour visuel clair sur l'état du matériel (disponible, en maintenance, etc.).

2.2 Utilisateurs cibles

L'application s'adresse à un gestionnaire de parc matériel, qui a besoin d'un outil rapide pour enregistrer les demandes de location, suivre l'état des équipements et retrouver les informations d'un client. Le périmètre reste volontairement simple (CRUD) afin de se concentrer sur la mise en place d'une architecture propre plutôt que sur des fonctionnalités métier avancées (facturation, réservation par créneau, etc.), qui constituent des pistes d'évolution.

3. Architecture technique

L'application repose sur une architecture à trois couches, séparant clairement l'interface, la logique d'accès aux données et le stockage :

Couche	Technologie	Rôle
Interface	C# WinForms (.NET Framework 4.8)	Saisie et affichage des données
API	PHP (PDO)	Traitement des requêtes et accès à la base

Base de données	MySQL / MariaDB (XAMPP)	Stockage persistant des données
Communication	HTTP / JSON	Échange de données entre couches

Le principe de communication est le suivant : l'application WinForms envoie des requêtes HTTP (GET, POST, PUT, DELETE) vers les scripts PHP exposés par le serveur Apache. Chaque script PHP interroge la base MySQL via PDO, puis retourne une réponse au format JSON. Côté C#, une classe dédiée (ApiService) est chargée d'effectuer ces appels HTTP et de désérialiser les réponses en objets métier (Client, Matériel) utilisés par les formulaires.

Ce choix d'architecture plutôt qu'une connexion directe de l'application C# à la base MySQL permet de découpler l'interface de la base de données : l'API constitue un point d'accès unique, réutilisable par d'autres clients (application web, mobile) et plus facile à sécuriser. Il permet aussi de contourner l'absence de connecteur MySQL natif et stable pour .NET Framework, difficulté rencontrée en cours de projet et détaillée en section 6.

4. Conception

4.1 Modèle de données

Deux entités principales structurent l'application :

Entité	Attributs
Client	id, nom, prénom, email, téléphone, adresse
Matériel	id, référence, désignation, catégorie, quantité, prix / jour, état

4.2 Structure du projet

Le projet est organisé en trois grands répertoires : « api » regroupe les scripts PHP exposant les endpoints REST (clients.php, materiel.php, config.php, setup.php) ; « database » contient le script SQL d'initialisation (matos.sql) créant les tables et insérant des données de démonstration ; « WindowsFormsAppGMmatos » contient l'application cliente, elle-même divisée en Models (classes Client et Materiel), Data (classe ApiService gérant les appels HTTP) et les formulaires (FormGenerale pour le menu, Formclient et FormMateriel pour la gestion de chaque entité).

Figure 1 : Menu principal de l'application (FormGenerale)

4.3 Endpoints de l'API

Méthode	URL	Description
GET	/api/clients.php	Liste des clients
POST	/api/clients.php	Créer un client
PUT	/api/clients.php?id=	Modifier un client
DELETE	/api/clients.php?id=	Supprimer un client
GET / POST / PUT / DELETE	/api/materiel.php	Mêmes opérations pour le matériel

5. Réalisation

La réalisation s'est déroulée en plusieurs étapes : mise en place de la base de données et du script d'initialisation, développement des endpoints PHP pour chaque entité, puis construction de l'application WinForms consommant ces endpoints.

5.1 Communication HTTP / JSON

La classe ApiService centralise les appels vers l'API : elle construit les requêtes HTTP (GET pour lister, POST pour créer, PUT pour modifier, DELETE pour supprimer), envoie les données sous forme de JSON, et désérialise les réponses en objets Client ou Materiel exploités par les formulaires. L'URL de base de l'API est paramétrée dans le fichier App.config, ce qui permet de changer d'environnement sans recompiler le code.

5.2 Interface WinForms et opérations CRUD

Chaque formulaire (Formclient, FormMateriel) affiche les données dans une grille et propose des champs de saisie ainsi que des boutons Nouveau, Enregistrer, Supprimer et Actualiser, permettant de réaliser les quatre opérations CRUD directement depuis l'interface.

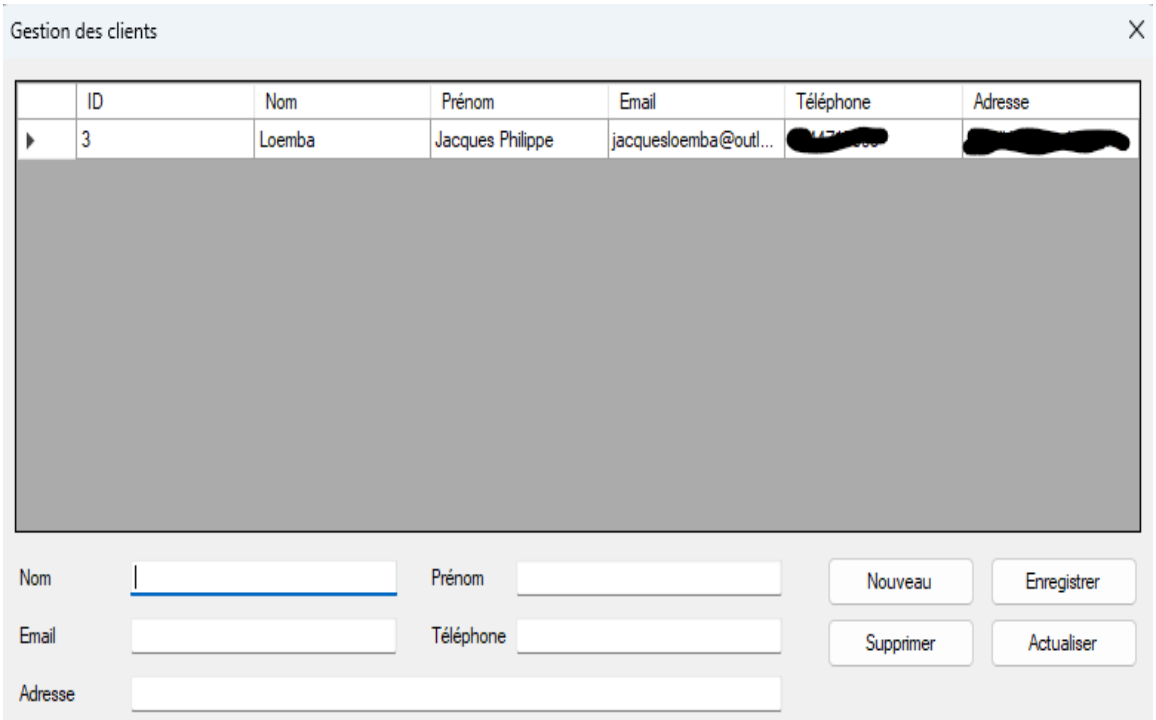

Figure 2 : Formulaire de gestion des clients

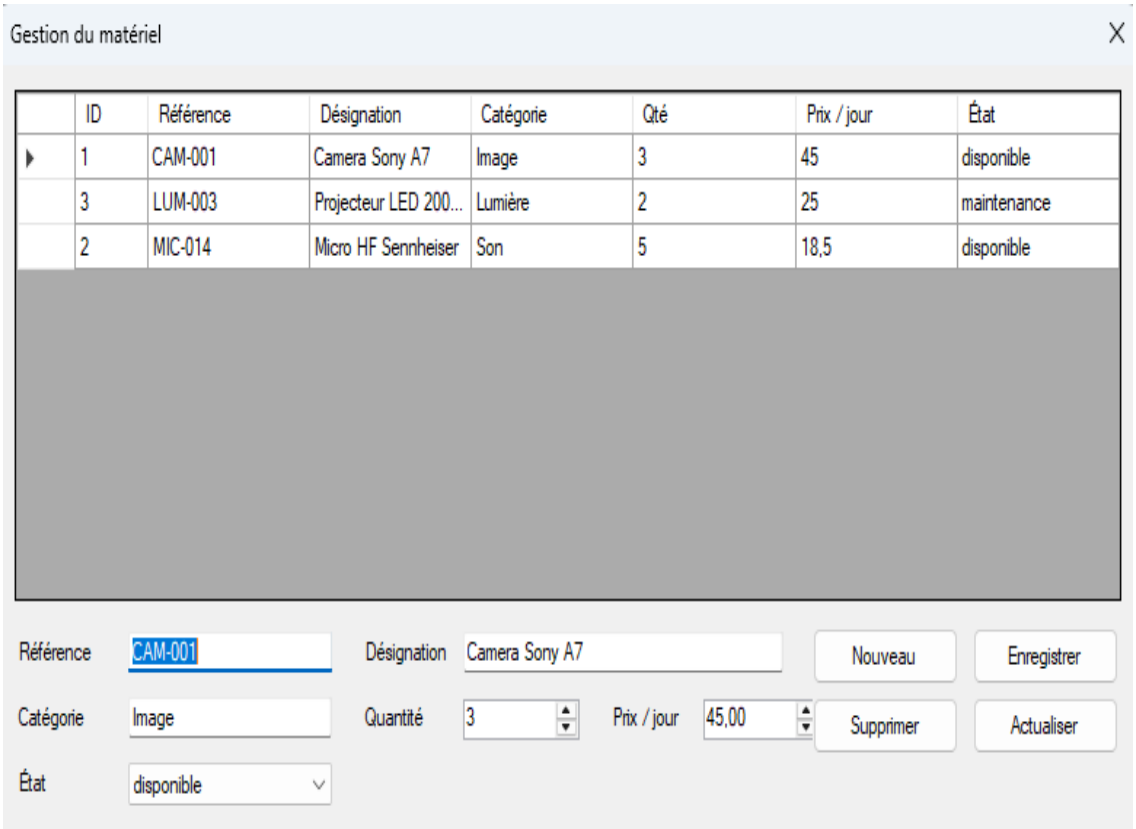

Figure 3 : Formulaire de gestion du matériel, avec suivi de l'état (disponible / maintenance)

Le formulaire Matériel illustre bien l'utilité du champ « état » : il permet de distinguer en un coup d'œil les équipements disponibles de ceux actuellement en maintenance, information essentielle pour la gestion du parc.

6. Difficultés rencontrées et solutions

Plusieurs difficultés techniques ont jalonné le développement du projet :

- Événements WinForms mal reliés : certains boutons (Enregistrer, Supprimer) ne déclenchaient pas le comportement attendu en raison d'un mauvais rattachement des gestionnaires d'événements dans le concepteur de formulaires. La vérification systématique des associations événement / méthode dans le fichier Designer.cs a permis de corriger le problème.
- Exécution d'une version non recompilée : après modification du code, l'application testée restait parfois l'ancienne version compilée, ce qui a conduit à des comportements incohérents. La solution a consisté à systématiquement nettoyer et reconstruire la solution (Clean puis Rebuild) avant chaque test.
- Disponibilité de MySQL : le service MySQL du panneau XAMPP ne démarrait pas toujours correctement (port déjà utilisé, service arrêté), ce qui bloquait l'accès à l'API. Un contrôle systématique de l'état des services Apache et MySQL avant chaque session de travail a été mis en place.
- Absence de connecteur .NET stable pour MySQL : plutôt que d'intégrer un connecteur MySQL directement dans l'application C# (source d'instabilité et de dépendances lourdes), le choix a été fait de passer par une API PHP intermédiaire, qui gère seule l'accès à la base via PDO. Cette solution a aussi simplifié l'architecture globale du projet.
- Synchronisation Git avec historiques divergents : des conflits sont apparus lors de la synchronisation du dépôt local avec le dépôt distant, les deux ayant évolué séparément. La résolution a nécessité une fusion manuelle des historiques (rebase / merge) et une vérification attentive des fichiers en conflit avant validation.

7. Tests et validation

Les tests ont été réalisés manuellement, directement depuis l'interface de l'application, en vérifiant pour chaque module (Clients et Matériel) le bon fonctionnement des quatre opérations CRUD : création d'une nouvelle fiche, modification d'une fiche existante, suppression, et actualisation de la liste après chaque action.

L'API a également été testée indépendamment via le navigateur et l'appel direct des scripts PHP, afin de vérifier que les réponses JSON retournées étaient correctement formées avant d'être consommées par l'application C#. Le script setup.php a permis de valider l'initialisation de la base de données avec des données de démonstration cohérentes, utilisées comme jeu de test tout au long du développement.

8. Conclusion et perspectives d'amélioration

Ce projet a permis de mettre en œuvre une application complète reposant sur une architecture client-serveur à trois couches: interface WinForms, API REST en PHP et base de données MySQL. Les fonctionnalités essentielles de gestion des clients et du matériel ont été implémentées avec succès, et les difficultés techniques rencontrées (événements, compilation, disponibilité des services, connecteur MySQL, gestion Git) ont été résolues, ce qui a permis d'approfondir la compréhension de chaque couche de l'architecture.

Plusieurs pistes d'amélioration peuvent être envisagées pour la suite du projet :

- Ajouter une gestion des locations reliant un client à un matériel sur une période donnée, avec calcul automatique du prix.

- Sécuriser l'API (authentification, validation des entrées côté serveur) avant tout déploiement au-delà d'un environnement local.
- Améliorer l'ergonomie de l'interface (recherche, filtres, tri des colonnes).
- Ajouter des tests automatisés afin de fiabiliser les évolutions futures de l'application.