# Rapport d’Analyse des Algorithmes Utilisés

*Projet : ArtGallery – Application de gestion de galerie d’art*

## Table des matières

1. [Introduction](#introduction)
2. [Structure Générale et Principes](#structure-générale-et-principes)
   * [Principe MVC](#principe-mvc)
   * [Utilisation de LINQ](#utilisation-de-linq)
   * [CRUD et Transactions](#crud-et-transactions)
   * [Sécurité et Traitement Serveur](#sécurité-et-traitement-serveur)
3. [Algorithmes et Logiques Clés par Module](#algorithmes-et-logiques-clés-par-module)
   1. [Gestion des Œuvres (Artwork)](#gestion-des-œuvres-artwork)
   2. [Gestion des Catégories](#gestion-des-catégories)
   3. [Gestion des Expositions](#gestion-des-expositions)
   4. [Panier d’Achat et Commandes](#panier-dachat-et-commandes)
   5. [Gestion des Comptes et Authentification](#gestion-des-comptes-et-authentification)
   6. [Livre d’Or (Golden Book)](#livre-dor-golden-book)
   7. [Liens Externes](#liens-externes)
   8. [Administration et Sécurité](#administration-et-sécurité)
4. [Résumé des Structures de Données](#résumé-des-structures-de-données)
5. [Tableaux Récapitulatifs](#tableaux-récapitulatifs)
6. [Conclusion](#conclusion)

## Introduction

Ce rapport détaille l’ensemble des algorithmes et logiques métiers utilisés dans l’application ArtGallery. Il vise à fournir une compréhension approfondie des mécanismes internes du projet, en expliquant comment les données sont traitées, sécurisées et présentées à l’utilisateur. L’accent est mis sur la clarté, la maintenabilité et la robustesse du code, avec une utilisation optimale des outils offerts par le framework ASP.NET Core MVC.

## Structure Générale et Principes

### Principe MVC

L’architecture MVC (Modèle-Vue-Contrôleur) sépare la logique métier, la présentation et la gestion des requêtes. Les modèles représentent les données et leur structure, les vues affichent l’information à l’utilisateur, et les contrôleurs orchestrent les interactions, appliquant les algorithmes nécessaires pour répondre aux actions de l’utilisateur.

### Utilisation de LINQ

LINQ (Language Integrated Query) est utilisé pour interroger, filtrer, trier et transformer les données provenant de la base via Entity Framework Core. Cela permet d’écrire des requêtes expressives, sûres et performantes, tout en gardant le code lisible et maintenable.

### CRUD et Transactions

Les opérations de base (Créer, Lire, Mettre à jour, Supprimer) sont omniprésentes dans tous les modules. Elles sont réalisées via Entity Framework, qui gère également les transactions pour garantir la cohérence des données, en particulier lors des opérations critiques comme la validation de commandes ou la suppression en cascade.

### Sécurité et Traitement Serveur

La sécurité est assurée par ASP.NET Identity, qui gère l’authentification, les rôles et la protection des données sensibles. La majorité des traitements (calculs, validations, filtrages) sont réalisés côté serveur afin d’éviter toute manipulation malveillante côté client et de garantir l’intégrité des opérations.

## Algorithmes et Logiques Clés par Module

### 1. Gestion des Œuvres (Artwork)

Ce module permet la gestion complète des œuvres d’art : ajout, modification, suppression, affichage et recherche. Les algorithmes principaux incluent le filtrage dynamique (par catégorie, disponibilité, titre), la gestion des images (upload, suppression physique et logique), le tri par date ou popularité, et le comptage des vues. Les opérations sont optimisées via des requêtes LINQ et des méthodes asynchrones pour la performance.

| Fonctionnalité | Algorithme / Logique Utilisée |
| --- | --- |
| Affichage des œuvres | Requêtes LINQ avec filtres (Where, OrderBy, Include pour images et catégories). |
| Recherche/Filtrage | Filtrage dynamique sur titre, catégorie, disponibilité (LINQ + expressions lambda). |
| Ajout/Modification | Validation des champs, gestion des fichiers images (upload, suppression), mise à jour de la base de données. |
| Suppression | Suppression en cascade des images associées via Entity Framework. |
| Comptage des vues | Incrémentation d’un compteur à chaque affichage de détail (simple incrémentation + sauvegarde EF). |

**Exemple de requête LINQ** :

var oeuvres = \_context.Artworks  
 .Include(a => a.Categories)  
 .Where(a => a.IsAvailable && a.CategoryId == id)  
 .OrderByDescending(a => a.CreatedAt)  
 .ToList();

### 2. Gestion des Catégories

La gestion des catégories repose sur des opérations CRUD classiques, avec une attention particulière à l’intégrité référentielle : avant toute suppression, le système vérifie qu’aucune œuvre n’est rattachée à la catégorie. Cela évite les incohérences et garantit la stabilité des données.

### 3. Gestion des Expositions

Les expositions sont affichées et triées selon leurs dates de début et de fin. Les algorithmes valident la cohérence des périodes, gèrent l’upload et la suppression d’images, et assurent la suppression sécurisée des expositions sans laisser de données orphelines. Les requêtes LINQ permettent de filtrer les expositions en cours ou passées.

### 4. Panier d’Achat et Commandes

Ce module gère le cycle complet d’un achat : ajout au panier, calcul dynamique du total, validation et création de commande, et historique des achats. Les algorithmes assurent la gestion des quantités, la cohérence des stocks, et la sauvegarde transactionnelle des commandes. Les totaux sont calculés via LINQ et les historiques filtrés par utilisateur connecté.

| Fonctionnalité | Algorithme / Logique Utilisée |
| --- | --- |
| Ajout au panier | Recherche de l’article, ajout à une collection en session ou base, gestion des quantités. |
| Calcul du total | Somme des sous-totaux (foreach ou LINQ Sum). |
| Validation de commande | Création d’une commande, génération des détails, décrémentation du stock si applicable, sauvegarde transactionnelle. |
| Historique | Filtrage des commandes par utilisateur connecté. |

**Exemple de calcul du total** :

decimal total = cartItems.Sum(item => item.Quantity \* item.UnitPrice);

### 5. Gestion des Comptes et Authentification

L’authentification et la gestion des rôles sont assurées par ASP.NET Identity. Les algorithmes de hachage, de validation et de gestion des rôles sont robustes et éprouvés, garantissant la sécurité des accès et la confidentialité des données utilisateurs. Les contrôles d’accès sont appliqués via des attributs [Authorize] et des vérifications côté contrôleur.

**Détail des mécanismes :** - **Inscription et connexion :** Lorsqu’un utilisateur s’inscrit, ses informations sont validées (unicité de l’email, force du mot de passe, etc.) puis stockées avec un mot de passe haché. À la connexion, la comparaison se fait sur le hash du mot de passe fourni. - **Gestion des rôles :** Les utilisateurs peuvent se voir attribuer un ou plusieurs rôles (par exemple, “Admin”, “Client”). Les contrôles d’accès aux pages et actions sensibles utilisent l’attribut [Authorize(Roles = "Admin")] pour restreindre l’accès. - **Sécurité :** ASP.NET Identity protège contre les attaques courantes (brute force, injection, etc.) et permet la gestion de la réinitialisation de mot de passe, la confirmation d’email, et la double authentification si activée. - **Exemple d’utilisation :** - Un administrateur peut accéder à la gestion des œuvres, des commandes et des utilisateurs. - Un utilisateur classique ne peut accéder qu’à ses propres commandes et informations personnelles. - **Logique métier :** Toute modification de données sensibles (profil, mot de passe, etc.) déclenche une vérification d’identité pour garantir la sécurité.

### 6. Livre d’Or (Golden Book)

Le livre d’or permet aux utilisateurs de laisser des messages, qui peuvent être soumis à modération. Les entrées sont validées, stockées et affichées selon leur statut d’approbation. Le tri par date permet de mettre en avant les messages récents, et la modération assure la qualité du contenu affiché.

**Détail des mécanismes :** - **Ajout d’une entrée :** L’utilisateur saisit son nom et son message. Le système valide la présence des champs obligatoires et la longueur du message. - **Modération :** Selon la configuration, une entrée peut nécessiter une approbation manuelle par un administrateur avant d’être visible publiquement (champ booléen IsApproved). - **Affichage :** Les messages sont triés par date de soumission, du plus récent au plus ancien. Seuls les messages approuvés sont affichés sur la page publique. - **Exemple de logique métier :** - Si un message contient des mots interdits ou offensants, il peut être rejeté automatiquement ou signalé pour modération. - L’administrateur dispose d’une interface pour approuver, rejeter ou supprimer les messages. - **Utilité :** Ce module favorise l’interaction et la confiance des visiteurs, tout en garantissant la qualité et la pertinence des messages affichés.

### 7. Liens Externes

Les liens externes sont gérés via des opérations CRUD et affichés groupés par type. Le tri par ordre de priorité permet de mettre en avant les liens les plus importants. Les algorithmes de groupement et de tri sont réalisés via LINQ.

**Détail des mécanismes :** - **Ajout/Modification/Suppression :** Les administrateurs peuvent ajouter de nouveaux liens, les éditer ou les supprimer via une interface dédiée. Chaque lien possède des attributs comme le titre, l’URL, le type (partenaire, ressource, etc.) et l’ordre d’affichage. - **Affichage groupé :** Lors de l’affichage sur le site, les liens sont regroupés par type grâce à LINQ (GroupBy), puis triés selon le champ de priorité (OrderBy). - **Exemple d’utilisation :** - Un bloc “Partenaires” affiche en premier les partenaires institutionnels, suivis des ressources utiles. - L’administrateur peut réorganiser l’ordre d’apparition pour mettre en avant certains liens lors d’événements ou de promotions. - **Logique métier :** Seuls les liens actifs (IsActive == true) sont affichés au public, ce qui permet de préparer des campagnes ou de désactiver temporairement des liens sans les supprimer.

### 8. Administration et Sécurité

L’administration repose sur des algorithmes d’initialisation (seed) pour la création des rôles et de l’utilisateur administrateur. Les contrôles d’accès sont systématiques sur chaque action sensible. La gestion des erreurs redirige l’utilisateur vers des pages adaptées, améliorant la robustesse et la sécurité de l’application.

## Résumé des Structures de Données

Chaque entité principale du projet possède une structure adaptée à ses besoins fonctionnels. Les relations entre entités (un-à-plusieurs, plusieurs-à-plusieurs) sont gérées par Entity Framework, qui assure la cohérence et l’intégrité des données. Les algorithmes associés à chaque entité exploitent ces structures pour offrir des fonctionnalités avancées tout en restant performants.

| Entité | Principaux Attributs | Algorithmes associés |
| --- | --- | --- |
| Artwork | Id, Title, Description, Price, Images, … | Filtrage, Tri, CRUD, Comptage de vues |
| Category | Id, Name, Description, IsActive | CRUD, Vérification d’intégrité |
| Exhibition | Id, Title, Dates, Image, Description | Filtrage, Tri, CRUD |
| Order | Id, User, Items, Total, Status, Dates | Calcul total, Historique, Validation |
| User | Id, Email, Roles, PasswordHash | Authentification, Gestion des rôles |
| GoldenBook | Id, Name, Message, IsApproved, Date | Filtrage, Modération, CRUD |
| ExternalLink | Id, Title, Url, Type, IsActive, SortOrder | Groupement, Tri, CRUD |

## Tableaux Récapitulatifs

### Principaux Algorithmes par Module

| Module | Algorithmes principaux | Complexité estimée |
| --- | --- | --- |
| Artwork | LINQ filtrage, upload fichier, CRUD | Faible à moyenne |
| Category | CRUD, vérification d’intégrité | Faible |
| Exhibition | Filtrage date, upload image, CRUD | Faible à moyenne |
| ShoppingCart | Calcul total, gestion quantités | Faible |
| Order | Validation, calcul, historique | Faible à moyenne |
| Auth | Hash, rôles, validation | Gérée par Identity |
| GoldenBook | Filtrage, modération | Faible |
| ExternalLink | GroupBy, tri, CRUD | Faible |

Ce tableau synthétise la nature et la complexité des algorithmes utilisés dans chaque module du projet.

## Conclusion

L’application ArtGallery s’appuie sur des algorithmes éprouvés de manipulation de données, de filtrage, de validation et de sécurité. Le choix d’utiliser les outils natifs du framework (LINQ, Entity Framework, Identity) garantit la robustesse, la maintenabilité et la sécurité du code. Chaque module a été conçu pour offrir une expérience utilisateur optimale tout en assurant la cohérence et l’intégrité des données.

**Pour chaque module, la priorité est donnée à la clarté, la sécurité et la performance via l’utilisation des outils natifs du framework.**