420-W1T-SW - Projet 2 Grille

Catégorie	Critère spécifique	Pts
Interprétation des spécifications *(15 pts)*		
Compréhension du sujet (IMDb, médiathèque)	Le modèle respecte le thème imposé (gestion de médias personnalisés). L'équipe a bien compris le rôle des utilisateurs, médias, genres, statuts, etc.	5
Choix des entités et relations pertinentes	Les entités choisies sont logiques et suffisantes. Les relations (1:N, N:N) sont bien modélisées. Les noms sont parlants.	5
Respect des requis fonctionnels	Toutes les fonctionnalités demandées (recherches, marquage, statistiques) sont présentes, même de façon simple.	5
Modèle de données & seed *(20 pts)*		
Modélisation (entités, relations, types, conventions)	Le schéma respecte les bonnes pratiques : noms clairs, types cohérents, normalisation (ou structure documentaire logique), clés bien définies.	6
Données importées depuis IMDb	L'équipe a sélectionné et extrait un sous-ensemble pertinent des fichiers IMDb, avec preuve dans le dépôt.	6
Implémentation du seed	Les données sont bien injectées dans la BD : via code (`HasData` , `InsertMany`) ou fichier. Le seed est fonctionnel.	4
Réalisme des données	Les champs conservés sont utiles. Les données sont propres (pas de doublons, pas de placeholder inutiles).	4
Requêtes et logique métier *(30 pts)*		
Implémentation complète du CRUD	Chaque entité principale (User, MediaItem, etc.) peut être créée, lue, modifiée et supprimée via DAO ou l'API.	5
Recherches avec filtres et mots-clés	L'application permet des recherches textuelles ou par genre. Utilisation judicieuse de `LIKE` , `Contains` , regex, etc.	5
Agrégations (statistiques)	Deux requêtes avec `GroupBy` , `Average` , `Count` , etc. Produisent des résultats clairs et utiles (ex. : genres les plus fréquents).	6
Requêtes avec projection	L'application renvoie des données ciblées (`select new { Title, Year }` ou DTO), pas des entités complètes non filtrées.	4
Requête impliquant une transaction	Une opération combine plusieurs actions liées (ex. : ajouter un film et ses genres). Exécutée de manière atomique (`TransactionScope` , EF `SaveChanges()` groupé, etc.).	5
Respect des bonnes pratiques dans les requêtes	Les requêtes sont claires, bien structurées, optimisées (pas de `ToList()` prématuré), respectent LINQ ou les pipelines Mongo.	5
DAO, architecture, ORM/ODM *(20 pts)*		
DAO correctement séparés et injectés	DAO créés pour chaque entité. Injectés via DI. Pas de logique métier dans les DAO. Séparation nette des responsabilités.	5
ORM ou ODM bien configuré	Utilisation correcte de EF Core ou MongoDB.Driver : `DbContext` , mapping (`Fluent API` , `[Key]` , etc.), collections et relations bien reliées.	5
Architecture générale saine	Projet organisé proprement : entités, DAO, modèles séparés. Dossier `DAL` ou `Data` , solution structurée.	5

Catégorie	Critère spécifique	Pts
Validation de modèle et logique métier	Attributs `[Required]`, `[MaxLength]`, `[Range]`, etc. utilisés. La validation métier est claire (ex. : interdire un film vide ou doublon).	5
Tests / API d'accès *(10 pts)*		
API REST minimale fonctionnelle	Les routes sont disponibles et testables (`GET`, `POST`, etc.). L'API n'est pas complexe mais bien structurée.	3
Suite Postman couvrant les cas importants	Une collection Postman est livrée. Elle contient au moins 5 requêtes pertinentes, chacune avec des assertions (`pm.test`).	4
README avec instructions de test	Contient les routes, exemples d'appel, format de réponse, configuration du projet, DB utilisée.	3
Qualité technique globale *(5 pts)*		
Qualité du code, lisibilité, nomenclature, structure	Indentation, nommage, constance (fr/en), absence de répétition ou de code inutilisé.	3
Utilisation appropriée de Git	Minimum 7 commits par équipier, progression visible, dépôt bien structuré, commits nommés correctement.	2