

Architecture de données réelle : EF Core, transactions, repositories et commandes

Guide de la leçon · Guide du lab / de l'examen

● LearnWisej

Guide de la leçon

Accès aux données en production, transactions, encapsulation par les services, séparation commandes/requêtes. Ce que cela signifie pour une équipe Wisej.NET avancée — et comment aborder ce module.



Enjeu de production

Accès aux données en production, transactions, encapsulation par les services, séparation commandes/requêtes.

1. Le problème en production

Les labs intermédiaires peuvent utiliser des services factices. Les applications avancées doivent se connecter à de vrais magasins de données sans transformer les écrans en clients SQL ni laisser les préoccupations de persistance fuir dans l'UI.

2. Concepts clés

Concept	Pourquoi c'est important
séparation EF Core	La ligne de séparation entre le code de l'UI et Entity Framework Core. Les écrans appellent des services et des commandes ; ils ne détiennent jamais de DbContext et n'écrivent jamais de LINQ sur les entités.
durée de vie du DbContext	Durée de vie d'une instance de DbContext. Des contextes courts, par opération, évitent le suivi obsolète, la croissance de la mémoire et les fuites entre sessions dans une application de longue durée.
repository ou service de requête	Les repositories chargent et enregistrent les agrégats pour les commandes, tandis que les services de requête renvoient des projections en lecture seule adaptées aux écrans. Les garder séparés permet à chacun de rester simple.
frontière transactionnelle	Unité de travail en base de données qui doit réussir ou échouer d'un seul bloc. Elle a sa place dans un service ou une commande, pas éparpillée dans les gestionnaires d'événements de l'UI.
objet de commande	Petite classe qui porte les entrées d'une opération métier et qui est exécutée par un gestionnaire, pour que l'opération puisse être validée, journalisée et testée indépendamment.
unité de travail	Coordonne toutes les modifications d'une opération et les valide ensemble, offrant un emplacement unique pour gérer les transactions et le suivi des modifications.
correspondance des erreurs	Traduire les exceptions de base de données, comme les violations de contrainte et les échecs de concurrence, en erreurs applicatives significatives que l'UI peut présenter sans risque.

3. Déroulé de la leçon

L'UI ne doit pas connaître la base de données

Un formulaire Wisej.NET peut être riche et avec état sans savoir comment les enregistrements sont stockés. Un code avancé traduit l'intention de l'UI en commandes et laisse les services applicatifs valider, autoriser, persister et auditer le résultat.

Choisir la frontière de données la plus simple qui résiste au changement

Les repositories, les services de requête et les gestionnaires de commande ne sont pas une cérémonie obligatoire. Ils sont utiles lorsqu'ils isolent une base de données, simplifient les tests ou empêchent le code de l'UI de disséminer des requêtes partout. Utilisez des frontières parce qu'elles résolvent un problème concret.

Une frontière de données à la mesure de l'application

Une application de gestion a besoin d'un endroit clair où commence la base de données. D'un côté se trouvent les formulaires, les pages et les services applicatifs qu'ils appellent. De l'autre, les contextes, les entités et les requêtes d'Entity Framework Core. Le diagramme de frontière du lab trace cette ligne et liste

ce qui la traverse : les commandes qui entrent, les résultats et les modèles de lecture qui sortent, et rien d'autre. Aucun formulaire ne détient de DbContext, et aucune entité n'est liée à une grille.

La forme de la frontière dépend de la taille de l'application. Une petite application peut placer les requêtes et les commandes dans une poignée de services applicatifs qui utilisent EF Core directement. Une application plus grande sépare les repositories, qui chargent et enregistrent les agrégats pour les commandes, des services de requête, qui renvoient des projections en lecture seule pour les écrans. La règle consiste à ajouter une couche lorsqu'elle isole quelque chose qui change, simplifie un test ou empêche les requêtes de se répandre dans le code, et pas avant.

Quelle que soit sa forme, la frontière est documentée une fois et appliquée en revue. Une pull request qui ajoute une requête LINQ à un formulaire est rejetée d'emblée.

Durée de vie du DbContext, transactions et commandes

Un DbContext est une unité de travail dotée d'un change tracker, et sa durée de vie doit correspondre à l'opération qu'il sert. Dans une session Wisej.NET de longue durée, un contexte stocké dans un champ de formulaire vit pendant des heures, accumule des entités suivies, sert des données obsolètes et ne peut pas être partagé sans risque avec une tâche en arrière-plan. La décision du lab sur la durée de vie est donc courte : un contexte par commande ou par requête, créé au début de l'opération et libéré à sa fin.

Une frontière transactionnelle englobe les écritures qui doivent réussir ou échouer ensemble. Approuver un ordre de travail met à jour l'enregistrement, écrit une entrée d'audit et met une notification en file d'attente ; les deux premières opérations appartiennent à une même transaction, et la troisième n'est mise en file qu'après sa validation. Cette frontière vit dans le gestionnaire de commande, là où toute l'opération est visible, ni dans le formulaire ni dans le repository.

Un objet de commande tel que `ApproveWorkOrderCommand` porte les entrées de cette opération et est exécuté par un gestionnaire qui renvoie un `CommandResult`. Comme la commande est un objet simple, le gestionnaire peut être testé avec une base de données en mémoire et sans aucun formulaire, ce qui correspond à la première question de révision du lab.

Des gestionnaires légers, de vrais services

Dans le lab, le gestionnaire de clic du bouton Approve tient en une ligne : il construit une commande à partir du contexte courant et de l'ordre de travail sélectionné, la transmet au service et affiche le résultat. C'est la forme visée pour chaque gestionnaire d'événements de l'UI. Le gestionnaire traduit un geste en intention et affiche un résultat. Il ne contient aucune validation au-delà de ce dont l'écran a besoin pour activer un bouton, aucune requête, aucune transaction et aucune connaissance de ce qu'est une base de données.

Tout le reste passe dans des services et des gestionnaires qui peuvent être revus sans ouvrir le concepteur. Un validateur vérifie la commande au regard des règles métier. Un repository charge l'agrégat. Le gestionnaire applique la transition et enregistre. Un auditeur consigne ce qui s'est passé. Chaque élément peut être testé isolément, réutilisé par un job d'import ou une WebMethod, et sécurisé une seule fois.

Le test d'un gestionnaire consiste à savoir s'il pourrait être supprimé et réécrit par quelqu'un qui n'a jamais vu la base de données. Si ce n'est pas le cas, la persistance a fui dans l'UI.

Traduire les erreurs de base de données en messages utiles

La base de données signale ses échecs dans son propre vocabulaire : violation de contrainte d'unicité, échec de clé étrangère, exception de concurrence, délai dépassé, perte de connexion. Aucun de ces messages ne doit atteindre l'utilisateur tel quel, et aucun ne doit atteindre le journal d'audit sous forme de stack trace brute. La table de correspondance des erreurs du lab liste chaque échec de base de données que l'application peut produire et l'associe à trois éléments : un code de résultat nommé, un message utilisateur et une entrée d'audit.

Un numéro d'ordre de travail en double devient un résultat indiquant que le numéro est déjà utilisé, affiché à côté du champ et journalisé comme échec de validation. Une exception de concurrence déclenche le flux de conflit du Module 3. Un délai dépassé devient un message indiquant que l'opération n'a pas pu aboutir, accompagné d'un ID de corrélation pour le support, et une entrée de log avec le nom de la requête et sa durée. L'utilisateur obtient quelque chose sur quoi agir ; le support obtient quelque chose à rechercher.

La correspondance se fait dans le gestionnaire de commande, à la frontière, pour que chaque appelant reçoive les mêmes codes de résultat, qu'il s'agisse d'un formulaire, d'un lot ou d'un job.



Réflexe de production avancé

Ne vous arrêtez pas quand l'écran paraît correct. Expliquez à qui appartient l'état, les implications de sécurité, les chemins d'échec et le comportement en production avant de considérer le module comme terminé.



Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Concevoir une séparation des accès aux données adaptée aux applications de gestion Wisej.NET.
- ✓ Utiliser la durée de vie du DbContext, les transactions et les objets de commande en toute sécurité.
- ✓ Tenir les gestionnaires d'événements de l'UI à l'écart de la logique de persistance.
- ✓ Traduire les erreurs de base de données en messages utiles pour l'utilisateur et pour l'audit.

Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.



Lab pratique

Remplacez la persistance factice de `WorkOrderService` par une implémentation reposant sur EF Core ou par une abstraction de repository de niveau production. Implémentez les opérations `Create`, `Update`, `Approve`, `Search` et `Audit` avec des frontières transactionnelles et des erreurs traduites.

Approche suggérée

- ✓ Lisez le problème en production et identifiez le risque d'architecture avant de toucher à l'UI.
- ✓ Créez ou mettez à jour d'abord l'encapsulation par un service, de commande, de modèle ou de composant concernée.
- ✓ Construisez l'écran ou le composant Wisej.NET en suivant les standards de projet établis.
- ✓ Ajoutez une démonstration du chemin d'échec, pas seulement du chemin nominal.
- ✓ Consignez ce qui a changé dans le journal d'apprentissage et mettez à jour la checklist concernée.

Livrables attendus

- ✓ Diagramme de la séparation des accès aux données
- ✓ Décision sur la durée de vie du `DbContext`
- ✓ Classes de commande et de résultat
- ✓ Exemple de transaction
- ✓ Table de correspondance des erreurs

Questions de révision

- ✓ Un enregistrement peut-il être testé sans créer de Form ?
- ✓ Que contient la transaction ?
- ✓ Comment les erreurs de concurrence et de clé unique sont-elles remontées ?



Réflexe de production avancé

Ne vous arrêtez pas quand l'écran paraît correct. Expliquez à qui appartient l'état, les implications de sécurité, les chemins d'échec et le comportement en production avant de considérer le module comme terminé.