

STD – TRANSVERSE

FRONT – ANGULAR | MATERIAL | NGRX | RXJS

BACK – .NETCORE | ENTITY FRAMEWORK

DATA – SQLITE

Sommaire

1. State Management
2. Communication – FRT / BCK
3. Autres sujets communs – FRT / BCK

Annexe – Questionnement

01

State Management

Storage
Data

State / Storage / Synchronisation

State Mngt

FRT / BCK ?	Type	Instance	Stable Pour :
FRT / Browser	Storage (localStorage)	1 / browser	App reinit (F5) / Browser Close
FRT / Browser	Storage (sessionStorage)	1 / tab	routeChange
FRT / App	globalState (Ngrx)	1 / tab	routeChange
FRT / App	localState (Component, Effect, Service)	1 / instance	Scrolling in same tab
BCK / DB	Applicative	1 / user ?	
?	Cache		

Sync - globalState / localStorage

- **Library** : ngrx-store-localstorage
- **globalState** réhydraté en partie grâce localStorage
- Donnée réhydraté :
 - Connexion Token, Etat des formulaires

Cas de @form

- Stocké dans « globalState » (configurable)
- Sync « localStorage » configurable manuellement
 - **(/!\ Pwd Field)**

02

Communication

FRONT / BACK

Synchronisation : « global state » - « backend »

- **Protocole** : HTTP / DTO : REST API
 - **DTO** : écrit manuellement (pourrait être généré par le BACK)
 - **Normalizer** : normalise les données (JSON) fourni par les API (cf. normalizr)
- **Basic Flow**
 1. Le user interagit avec un (ng)Component
 - Le (**ng**)Component dispatch une (**ng**)Action
 - Example : le user clique sur le bouton de soumission du formulaire d'ajout d'un produit
 2. Un (**ng**)Effect « <api-name>-api.effects.ts » réagit à la (**ng**)Action,
 1. Il appelle un (**ng**)Service « <api-name>.service.ts »
 2. Le (**ng**)Service utilise le HttpClient pour faire une requête http vers une API
 3. En fonction du retour du service, Le (**ng**)Effect renvoie une (**ng**)Action (Success / Failure)

Création d'une Entité (TO_THINK)

Comm FRT/BCK

- A quel moment l'entity state reducer doit ajouter la nouvelle entité créée ?

Après l'interaction user	Au retour de l'appel à l'API BACK
(+) : Délai maj l'UI indépendant du délai de réponse BCK => navigation + fluide (-) : Il faut potentiellement implémenté les règles métier coté Front (Ex si l'entité ne peut pas être créé car une avec le même non existe déjà) (?) : La responsabilité de génération de l'id est coté FRT (ou il faut implémenté un système de synchronisation de l'id de l'entité avec le retour du BCK)	(+) : on a l'id de l'entité, on sais que le back a bien l'info (-) : Affichage UI dépendant du délai de réponse BCK (?) : Id génération respo est coté BCK (?) : une route BCK de création d'entité doit retourner l'id de l'entité crée

Référence

- [id-generation-strategy](#)
- [using ngrx-entity](#)

- Comment répartir les responsabilités entre FRT / BCK :
 - Génération de l'ID de la nouvelle entité
 - Application des RG autorisant la création de l'entité (Ex : si on limite le nb d'entité dans une version gratuite d'un soft)
- Variable de temporalité : Tps réponse serveur, Capacité machine FRT, Tps Application RG

TO_THINK

- **Dans le FRT -> @ngrx/entity**
 - Utilisation des **Partial<>** pour mettre à jour des sous partie d'un objet
- **Dans le BCK**
 - Pour l'instant -> Route dédié à un besoin de MAJ
 - resetBoughtStatus
 - Trouver une solution C# pour utiliser un équivalent de Partial ?

03

Autres sujets communs

FRONT / BACK

Enum (=Dictionnaire de valeur)

Autres sujets

- **2 Types**
 - **Statique (dev-modifiable)** : défini dans le code FRT ou BCK
 - **Dynamique (user-modifiable)** : défini dans la BD (Table ENUM)
- **Statique**
 - **FRT** : Utilisation de Strings Enum de TS (Simplifie @form/MultipleOptionField)
 - **BCK** : enum C#
- **Dynamique**
 - **FRT** : module @enum qui appelle l'API Enum
 - **BCK** : API qui pour l'instant tape des énum codé en dur

Merci !

Principaux Paramètre

- **Temporalité**
- **Site Client**
- **Utilisateur Connecté**

Éléments Impactable

- Disponibilité du service
- Design Graphique UI
- Donné : Enum Type Partenaire
- Règle gestion métier
- Format d'un fichier d'export
- ...

- **Temporalité (Inspiré d'Applicam)**

- Dans la BD :

- **User connecté**

- Ajouter un module de conf user ?
- Avoir une instance de BD pour chaque utilisateur ? (pour l'instant 1 Account co présent dans le HttpContext) & 1 BD

- **Site Client (Inspiré d'Applicam/Seitra)**

- **BCK / WS :**

- Au sein du code source avec un BusinessService Transverse qui peut être surchargé selon le numéro de site qui est fournit dans la requête HTTP reçu en entrée
- En instanciant une version différente du WS pour chaque Site

- **BCK / BATCH :** fichier conf ds code source

- **FRT :** fichier conf ds code source

- **Model**
 - Chaque entité du model doivent avoir un Id
 - - Cet id doit être de type 'string' ? Utiliser un GUID ?
- **Code Organisation**
 - Regrouper les services (API call) et les effects associés dans un dossier dédié ?
- **State Organisation**
 - Séparer les états des "DBEntity" des états de "UIView"
 - **DBEntityState** : "shoppingList", "account", "enum", "category"...
 - **UIViewState** : "ShoppingList/Accordeon, editMode..."

- **Model / Révision**

- Chaque entité du model a un Id. Doit être de type 'string' / 'GUID' / 'number' ?

- **Permettre au User de Catégoriser (ou non) leurs items dans la shoppingList**

- Introduction des tables : « **Tree** », « **TreeNode** », « **Order** », « **Item** », « **Category** »
- Une shoppingList devient une liste d'item, ces items peuvent être lié à un « **TreeNode** »
- Revoir Cat/SubCat en Usage/SousUsage ? (Manger->Matin,Midi.../Sociabiliser/Hygiène/EntretienHome)

- **API/Enum** : Stocker les enums dans 1 table au lieu d'une enum C#

- **OldWebApp Vs SPA**

- L'API Back ne deviendrait pas simplement un moyen de persister un state ?
- State/save, get
- Avantage – simplification API
- Inconvenient – besoin de pouvoir synchro des sous-partie du state, encoder le state pour sécurité