	Proc#179 - Développement : règles et bonnes pratiques
Sujets connexes	

Règles sur les classes

Suivi des modifications majeures	04/2026 - Nicolas MARCHAND - Création 05-2026 - Evolutions majeurs
Suivi des approbations	Cartographie fonctionnelle
Objet	Normaliser les classes servant à mapper dans Angular les objets du back (/ de la BDD)
Destinataires	**- Validation des modifications :** Chef de projet **- Approbation du document** : Tous développeurs

Nommage des classes de donnée (modele)

Classe correspondant à

Correspondance	Règle de nommage
une table des BDDs	Dans ce cas le nom doit être celui de la classe pascal. Soit TSQL + nom de la table dans la base. Exemples : TSQLPersonne, TSQLFamille ... Pour facilité le fichier s'appelle record + nom de la classe (a l'image du back). Il est stocké dans `ngx-logeasweb-ui\src\lib\models\BDD\Nom bdd\`
à une classe interne au front	On préfixe le nom de la lettre T suivi d'un nom parlant. NB : on ne crée pas une classe copie d'une classe TSQL, si on a besoin d'ajouter des éléments on utilise l'héritage (extends) Exemple : `TPersonnePlus extends TSQLPersonne { nomDeJeuneFille : string …}` ou on ajoute une propriété à la classe de base calculée Exemple : `getNomComplet(){return `${this.titre} ${this.prenom} ${this.nom}`}` Si la classe est à usage unique du composant on pourra utiliser une Interface
classes dérivées pour affichage	En fait il ne s'agit pas de classe mais d'interface (pas de fonction de traitement). Le nommage se fait sur le nom de la table plus Grid Exemple : `IPersonneGrid` corresponds au type `TSQLPersonne` mis à plat pour usage dans la grille

Les classes issues des BDDs (TSQL...)

Quelques règles génériques

- On ne **charge/enregistre/efface** les données depuis le back directement depuis le code mais uniquement au travers des fonctions des classes **extends SynopseClientService** afin d'avoir des comportements homogènes
- Une classe **NE DOIT APPELER** qu'une autre classe de typage, jamais DGService ou extends SynopseClientService. Si c'est le cas c'est le service qui doit le faire.
  Exemple : pour effacer une famille il faut vérifier les personnes donc on devrait importer

TSQPersonne dans recordFamille et réciproquement — on aurait donc TSQLFamille qui appelle une méthode de TSQLPersonne et réciproquement ⇒ à mettre dans TBaseLoGeAs.
- Vu les problématiques à respecter la syntaxe ou privilégiera TSQLFamille[] à créer y=une classe TSQLFamilles

Les fonctions des classes "Serveurs"

Chargement de données (chargeFromBDD$)

```
async chargeFromBDD$<T extends TSQLRecord>( classe:
TSQLRecordConstructor<T>, filtre: Partial<T>|null = null): Promise<T[]>
```

Entrée	**Fonction**
classe: TSQLRecordConstructor<T>	Indique la table qui doit être chargée au travers de la classe (exemple : TSQLPersonne, TSQLHistoriqueDon...)
filtre: Partial<T>|null = null	Paramètre optionnel Si il est indiquer le filtrage demandé à la BDD pends en compte ce filtrage Si il est omis la fonction **calculFiltre_chargeFromBDD** de la classe (indiqué si dessus) et appelée pour créer si pertinent un filtre sur secteur, exercice, utilisateur)

Sortie	**Fonction**
Promise<T[]>	Un tableau contenant des éléments de la classe indiqué. Les éléments du tableau doivent être typé/traduit correctement au travers de la fonction de la classe **corrigeDataFromBDD** qui est appeler pour chaque élement viaa la procedure **mapArrayTo**. NB: Celle-ci prends aussi en charge les problématique de case

Exemple d'appel :

```
********** CLASSIQUE **********
  detailDon!:TSQLHistoriqueDon[];
  listeCerfa!:TSQLHistoriqueDonSynthese[];
  [...]
  this.detailDon = await
this.DG.BaseCourante.chargeFromBDD$(TSQLHistoriqueDon,filtre);
  this.listeCerfa = await
this.DG.BaseCourante.chargeFromBDD$(TSQLHistoriqueDonSynthese,{personne:this
.personne.ID});
********** SIGNAL**********
  public personnes = signal<TSQLPersonne[]>([]);
  [...]
  const [titres, liens, data] = await Promise.all([
     await this.DG.BaseCourante.GetFichierTexte$('/PersonneTitre.Txt'),
     await
this.DG.BaseCourante.GetFichierTexte$('/PersonneLienFamille.Txt'),
     await this.DG.BaseCourante.chargeFromBDD$(TSQLPersonne)
```

```
  ]);
  this.listeTitres = titres.lignes;
  this.listeLiensFamille = liens.lignes;
  this.personnes.set(data);
```

Les fonctions « normées » correspondant au CRUD

Opération	Fonction	Description

READ	`chargeFromBDD$`	`static async chargeFromBDD$(filtre:TSQLPersonne=new TSQLPersonne(), base?:TBaseLoGeAs) : Promise<TSQLPersonne[]>`
		C'est la procédure qui doit se charger de toutes les transformations nécessaires après la réception (appel de la fonction générique `mapArrayTo` qui appelle `corrigeDataFromBDD`).
		`static` permet d'appeler la fonction sans instance de la classe.
		Exemple : `this.personnes = await TSQLPersonne.chargeFromBDD$();`
		Le filtre permet de demander à la BDD un sous-ensemble de la table.
		Exemple : `const filtre = new TSQLPersonne({Codepostal: “31800”}) ;`
		`this.personnes = await TSQLPersonne.chargeFromBDD$(filtre);`
		→ on ne devrait avoir que les personnes ayant un codepostal = 31800 (A TESTER)
		`base` peut être laissé vide si on utilise la base chargée dans DG, sinon il faut bien sûr l'indiquer (possibilité de charger une deuxième base)

CREATE / UPDATE	`sauveToBDD$`	`async sauveToBDD$(base?:TbaseLoGeAs)` Permet de créer (ID=0) ou de mettre à jour l'objet instancié dans la base de données. Aucune préparation ne doit être faite avant, c'est la procédure qui s'en charge. `base` peut être laissé vide si on utilise la base chargée dans DG, sinon il faut bien sûr l'indiquer (possibilité de charger une deuxième base)
DELETE	`effaceToBDD$`	`async effaceToBDD$(base?:TbaseLoGeAs)`

Les fonctions « normées » à usage interne de la classe ("protected")

Nom de la fonction	Usage
protected override corrigeDataFromBDD()	Appelé sur chaque enregistrement lors du chargement depuis le back, via la procédure MapUtils.mapArrayTo. Elle permet de corriger, mapper.. les données sur le modèle de la classe. Elle permet en outre de s'affranchir des problèmes de case entre back et front
protected override async prepareDataToBDD()	Appelé par la fonction sauveToBDD$ elle permet de mettre les format angular au format attendu par le back

Les fonctions « normées » à usage public

Nom de la fonction	Usage
override toGrid():IFamilleGrid	Met à plat la structure parente (champ personnalisé, sous structure...) afin de faciliter leur affichage dans les grilles Peut permettre au passage à ajouter des champs calculé **Attention :** la structure résultante est une Interface elle ne peut/doit pas servir pour interfacer avec le back ou autre
static toGrids(liste:TSQLFamille[]):IFamilleGrid[]	Boucle sur les éléments du tableau pour appelé togrid

Les fonctions spécifique à la classe

Dépends de chaque classe voir la classe

From:
https://wiki-logeas.fr/certif/ - **dokuwiki-certif**

Permanent link:
https://wiki-logeas.fr/certif/doku.php?id=certif:procedure:usageclasseinterface&rev=1778060798

Last update: **2026/05/06 11:46**