

	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 : getNomComple() {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 correspond 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 SynopsisClientService** afin d'avoir des comportements homogènes
- Une classe **NE DOIT APPELER** qu'une autre classe de typage, jamais DGService ou extends SynopsisClientService. 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 TSQPersonne 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 TSQRecord)( classe:
TSQRecordConstructor<T>, filtre: Partial<T>|null = null): Promise<T[]>
```

Entrée	Fonction
classe: TSQRecordConstructor<T>	Indique la table qui doit être chargée au travers de la classe (exemple : TSQPersonne, TSQHistoriqueDon...)
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 élément via la procédure mapArrayTo . NB: Celle-ci prends aussi en charge les problématiques de case Si une erreur est détectée, l'utilisateur est informé et un tableau vide est rendu

Exemple d'appel :

```
***** CLASSIQUE *****
detailDon!:TSQHistoriqueDon[];
listeCerfa!:TSQHistoriqueDonSynthese[];
[...]
this.detailDon = await
this.DG.BaseCourante.chargeFromBDD$(TSQHistoriqueDon,filtre);
this.listeCerfa = await
this.DG.BaseCourante.chargeFromBDD$(TSQHistoriqueDonSynthese,{personne:this
.personne.ID});
***** SIGNAL*****
public personnes = signal<TSQPersonne[]>([]);
[...]
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);

```

Enregistrement de données (sauveToBDD\$)

```

  async sauveToBDD$<T extends TSQLError>(record:T, source: "FICHIER" |
  "COMPTA" | "SANSOBJET" = "SANSOBJET", forceCreate:boolean=false): Promise<T
  | null>

```

Entrée	Fonction
record:T	Il s'agit de l'objet à sauvegarder. Il doit bien sur être d'un type connu
source: "FICHIER" "COMPTA" "SANSOBJET" = "SANSOBJET"	Paramètre optionnel Utilisé pour la mise à jour des dates de fin de consentement (TSQLPersonne, TSQLFamille)
forceCreate:boolean=false	
Sortie	Fonction
Promise<T[]>	Rend l'élément tel qu'enregistré dans la BDD. L'élément doit être typé/traduit correctement au travers de la fonction de la classe corrigeDataFromBDD qui est appelé dessus. NB: Celle-ci prends aussi en charge les problématiques de case Si une erreur est détectée, l'utilisateur est informé et null est rendu

Exemple d'appel :

```
const res =await this.DG.BaseCourante.sauveToBDD$(personne, "FICHIER")
```

Fonction d'effacement (effaceFromBDD\$)

```

  async effaceFromBDD$<T extends TSQLError>(record:T): Promise<boolean>

```

Entrée	Fonction
record:T	Il s'agit de l'objet à effacer. Il doit bien sur être d'un type connu
Sortie	Fonction
Promise<boolean>	Indique si Ok

Exemple d'appel :

```
await this.DG.BaseCourante.effaceFromBDD$(famille);
```

Les fonctions « normées » à définir dans toutes les classes TSQL...

Fonction	Description
<pre>static override readonly sqlTableName: string = "personne";</pre>	En haut de la classe, doit indiqué (en minuscule) le nom de la table correspondante dans la BDD
<pre>constructor(init?: Partial<TSQLPersonne>) { super(init); if (init) { Object.assign(this, init); } }</pre>	

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

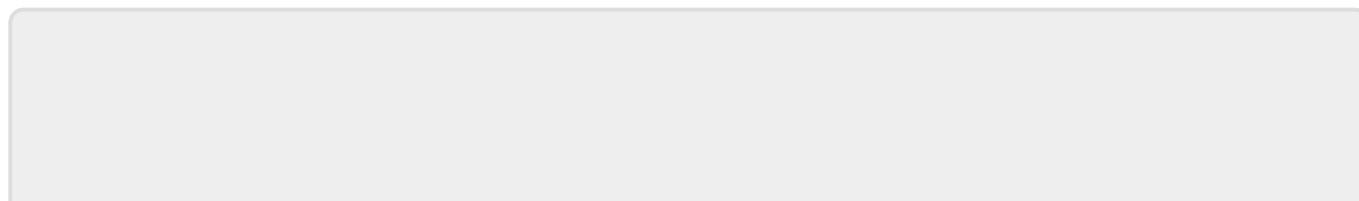
Nom de la fonction	Usage
<pre>protected override corrigeDataFromBDD()</pre>	Appelé sur chaque enregistrement lors du chargement depuis le back, via la procédure MapUtils.mapArrayTo. Elle permet de corriger, mapper.. les données sur le modèle de la classe. Elle permet en outre de s'affranchir des problèmes de case entre back et front
<pre>protected override async prepareDataToBDD()</pre>	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
<pre>override toGrid():IFamilleGrid</pre>	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
<pre>static toGrids(liste:TSQLFamille[]):IFamilleGrid[]</pre>	Boucle sur les éléments du tableau pour appelé togrid

Les fonctions spécifique à la classe

Dépend 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=1778061954>



Last update: **2026/05/06 12:05**