

|   |  |
|---|--|
|                  | <a href="#">ANGULAR : Installation, Bonne pratique, Astuces...</a> |
|  Sujets connexes |  |

## Règles sur les classes

|   |  |
|---|--|
| <b>Suivi des modifications majeures</b> | 04/2065 - Nicolas MARCHAND - Création  |
| <b>Suivi des approbations</b>           | Cartographie fonctionnelle   |
| <b>Objet</b>                            | Normaliser les classes servant à mapper dans Angular les objets du back (/ de la BDD)                          |
| <b>Destinataires</b>                    | - <b>Validation des modifications</b> : Chef de projet<br>- <b>Approbation du document</b> : 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)<br>Exemple : TPersonnePlus extends TSQLPersonne {<br>nomDeJeuneFille : string ...}<br>ou on ajoute une propriété à la classe de base calculée<br>Exemple : getNomCompleet(){return `\${this.titre}<br>\${this.prenom} \${this.nom}`}<br>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<br>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 les données depuis le back directement depuis le code mais uniquement au travers des fonctions de la classe afin d'avoir des comportements homogènes
- Une classe **NE DOIT PAS APPELER** une autre classe de typage, 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 TSQLPersonne dans recordFamille et réciproquement — on aurait donc TSQLFamille qui

appelle une méthode de TSQLPersonne et réciproquement ⇒ à mettre dans un service.

- Vu les problématiques à respecter la syntaxe ou privilégiera TSQLFamille[] à créer y=une classe TSQLFamilles

## Les fonctions « normées » correspondant au CRUD

| Opération       | Fonction        | Description  |
|-----------------|-----------------|--|
| READ            | chargeFromBDD\$ | <p>static async chargeFromBDD\$(filtre:TSQLPersonne=new TSQLPersonne(), base?:TBaseLoGeAs) : Promise&lt;TSQLPersonne[]&gt;</p> <p>C'est la procédure qui doit se charger de toutes les transformations nécessaires après la réception (appel de la fonction générique <code>mapArrayTo</code> qui appelle <code>corrigeDataFromBDD</code>).</p> <p>static permet d'appeler la fonction sans instance de la classe.</p> <p>Exemple : <code>this.personnes = await TSQLPersonne.chargeFromBDD\$()</code> ;</p> <p>Le filtre permet de demander à la BDD un sous-ensemble de la table.</p> <p>Exemple : <code>const filtre = new TSQLPersonne({Codepostal: "31800"}) ;</code><br/> <code>this.personnes = await TSQLPersonne.chargeFromBDD\$(filtre)</code> ;</p> <p>→ on ne devrait avoir que les personnes ayant un codepostal = 31800 (A TESTER)</p> <p>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)</p> |
| CREATE / UPDATE | sauveToBDD\$    | <p>async sauveToBDD\$(base?:TbaseLoGeAs)</p> <p>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.</p> <p>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)</p>   |
| DELETE          | effaceToBDD\$   | <p>async effaceToBDD\$(base?:TbaseLoGeAs)</p>  |

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

| Nom de la fonction                                       | Usage  |
|--|--|
| protected override <code>corrigeDataFromBDD()</code>     | Appelé sur chaque enregistrement lors du chargement depuis le back, via la procédure <code>MapUtils.mapArrayTo</code> . 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 <code>prepareDataToBDD()</code> | Appelé par la fonction <code>sauveToBDD\$</code> 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<br>Peut permettre au passage à ajouter des champs calculé<br><b>Attention :</b> la structure résultante est une Interface elle ne peut/doit pas servir pour interfacier 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

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>

Last update: **2026/04/24 17:04**

