	ANGULAR : Installation, Bonne pratique, Astuces...
Sujets connexes	

Règles sur les classes

Suivi des modifications majeures	04/2065 - Nicolas MARCHAND - Création
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 developpeurs

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 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 TSQPersonne dans recordFamille et réciproquement — on aurait donc `TSQLFamille` qui

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

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$`	`async effaceToBDD$(ID:number, base?:TbaseLoGeAs)`