# Le service de traduction

Le service de traduction utilisé est celui d'Angular 2 : `import { TranslateService } from "@ngx-translate/core";`

Afin d'adapter son fonctionnement avec l'architecture de l'application et le patterne observable, ce service à été wrappé par le service `i18n.service.ts`.

# Ajout d'une traduction au dictionnaire

## 1 - Compléter le fichier dictionnaire situé : `<root_app>/tools/i18n/dico.json`

Ce fichier suit une syntaxe précise ; la racine du json prends deux attributs :

- **defaultLang** qui définit la langue par défaut à utiliser dans l'application via le code du pays à 2 lettres.
- **dico** qui contient tous les textes traduits.

à l'intérieur de dico, les textes peuvent être ranger dans des catégories. Pour définir une catégorie il suffit de mettre un point à la fin du nom de l'attribut.
EXEMPLE d'un dico contenant 2 catégories nommées respectivement 'categorie1' et 'categorie2' :

```
{
    "defaultLang": "en",
    "dico": {
        "categorie1.": {...},
        "categorie2.": {...}
    }
}
```

**NOTE :** Une catégorie peut contenir elle même une ou plusieurs catégories.

Une fois les catégories créées, on peut commencer à y mettre les attributs qui contiendrons les textes de traduction :

```
{
    "defaultLang": "en",
    "dico": {
        "categorie1.": {
            "texte_bouton_connexion": {
                "en": "log in",
                "fr": "Se connecter"
            }
        },
        "categorie2.": {
            "titre_principal": {
                "en": "My awsome APP",
                "fr": "Mon application d'enfer"
            },
            "un_autre_exemple": {
                "en": "Only english available for this text" <== On peut omettre la traduction, auquel cas, c'est la traduction dans
            }
        }
    }
}
```

**NOTE :** Il est préférable de toujours garder la langue par défaut en premier et de garder les autres langues dans le même ordre.

Il est également possible d'ajouter des emplacements disponible pour l'interpolation : EXEMPLE concret :

```
"impossible_to_fetch_planning": {
    "en": "Error when trying to fetch planning for {{ month }} {{ year }}",
}
```

Les variables "month" et "year" seront remplacées par les valeurs données au moment de la demande de traduction (voir chapitre suivant sur l'utilisation du dictionnaire".

## 2 - Générer les dictionnaires et la classe de chargement des dicos

A la racine du projet, taper la commande : `$ npm run dico`
Cette commande va créer un dictionnaire (un fichier JSON contenant les textes traduits) par langue, puis générer la classe permettant au service de traduction de savoir quel dictionnaires doivent être chargés.

Les dictionnaires sont créés dans le package `src/app/i18n/` .

# Utilisation du dictionnaire

## 1 - Importer le service **i18n** dans le composant puis déclarer un objet littéral public, destiné à recevoir les textes traduits :

```
export class MyComponent implements OnInit {
    textUI: object; // L'objet qui contiendra toutes les traductions

    constructor(
        private _L: i18nService // Injection du service i18n
    ) {
        this.textUI = {}; // Initialisation de l'objet (important pour éviter que la vue ne crash)
    }
   ...
}
```

## 2 - Invoquer le service i18n à l'initialisation du composant (hook ngOnInit)

La traduction se fait en utilisant la méthode translate() du service i18n.

DOC de la méthode translate() :

```
Récupère la traduction correspondante demandée par le 'keyword' de manière asynchrone, puis la passe à dest. Les points (.) contenus

EXEMPLE : demandé "ui_activity.display_error" aura comme résultat l'objet dest valorisé comme suist :<br> dest = { ui_activity_displ

@param keyWord le mot clé désignant le texte traduit
@param dest – Si c'est un objet, alors il sera valorisé avec le texte traduit mappé avec en guise de clé le 'keyword'.
Si c'est une fonction, cette dernière sera invoquée et recevra en paramètre le texte traduit.
@param params – (optionnel) Un objet littéral contenant les valeurs devant être interpolées dans le texte traduit.
```

## utilisation basique

Ci-dessous un exemple où on charge 4 textes contenus dans la catégorie **ui_login** :

- login_text_button
- loging_text_button
- error_login
- wrong_login

```
export class MyComponent implements OnInit {
    ...
    ngOnInit() {
        // Dans cet exemple on valorise this.textUI avec 4 textes traduits dans la langue actuellement sélectionné dans les paramètr
        this._L.translate("ui_login.login_text_button", this.textUI);
        this._L.translate("ui_login.loging_text_button", this.textUI);
        this._L.translate("ui_login.error_login", this.textUI);
        this._L.translate("ui_login.wrong_login", this.textUI);
    }
    ...
}
```

**NOTE :** Il est à noter que la méthode **translate** est **asynchrone**.

## traductions avec interpolations

Ci-dessous un exemple concret de l'utilisation d'une interpolation sur un message disant qu'il est impossible de récupérer une activité :

Définition dans le fichier dico.json :

```
...
"impossible_to_fetch_activity": {
    "en": "Something went wrong when trying to fetch the activity number {{ activityNumber }}",
    "fr": "Il a eu un problème lors de la tentative de récupération du détail de l'activité numéro {{ activityNumber }}"
},
...
```

Code typescript :

```
this._L.translate("ui_activity.impossible_to_fetch_activity", (translatedMessage: string) => {
    this.notificationService.pushAlert(translatedMessage, MessageType.ERROR, 5000);
}, {activityNumber: this.activityID});
```

## utilisation particulière

Il peut être nécessaire parfois de calculer la valeur d'un texte traduit pendant une action. C'est le cas par exemple lorsque l'on doit afficher un texte avec une valeur interpolée qui peut changer d'une action à une autre. Exemple concret lorsque l'on veut afficher un message de confirmation de téléchargement d'un fichier, avec le nom du fichier téléchargé.

Ci-dessous un exemple

```
import { NotificationService, MessageType } from '@src/app/services/Notification.service';

...

_this._L.translate("ui_export_pdf.success_export_pdf", (translatedMessage) => {
    _this.notificationService.pushAlert(translatedMessage, MessageType.SUCCESS, 5000);
}, { filePath });
```

## 3 - Exploitation d'un texte traduit

- Dans un composant directement

```
this.logInMessage = this.textUI['ui_login_wrong_login'];
```

- Dans la vue

```
<Button fontSize="16" [text]="loging ? textUI.ui_login_loging_text_button : textUI.ui_login_login_text_button" (tap)="onLogin()" [is
```