# Application de gestion de client et collaborateur

Table des matières

[Application de gestion de client et collaborateur 1](#_Toc486326035)

[But 2](#_Toc486326036)

[Utilisation 2](#_Toc486326037)

[Arborescence 2](#_Toc486326038)

[Fichiers du dossier JS 3](#_Toc486326039)

[Itop de recette 3](#_Toc486326040)

[Technologies utilisées 4](#_Toc486326041)

[Ajax 4](#_Toc486326042)

[Animation de chargement 5](#_Toc486326043)

[Datepicker 5](#_Toc486326044)

[Tooltip 6](#_Toc486326045)

[Cookie et localStorage 6](#_Toc486326046)

[Fonctions par fichier 7](#_Toc486326047)

[load\_backlog.js (appli gestion client et collabo) 7](#_Toc486326048)

[load\_backlog.js (appli backlog version éditable) 10](#_Toc486326049)

[load\_CI (appli gestion client) 11](#_Toc486326050)

[load\_CI (appli gestion collabo) 13](#_Toc486326051)

## But

L’application de gestion des clients a pour but de permettre de visualiser différentes informations sur un client en particulier tel que ses CIs, ses contacts, son backlog et son document de connexion. Il est aussi possible d’effectuer des modifications sur certains tickets du backlog.

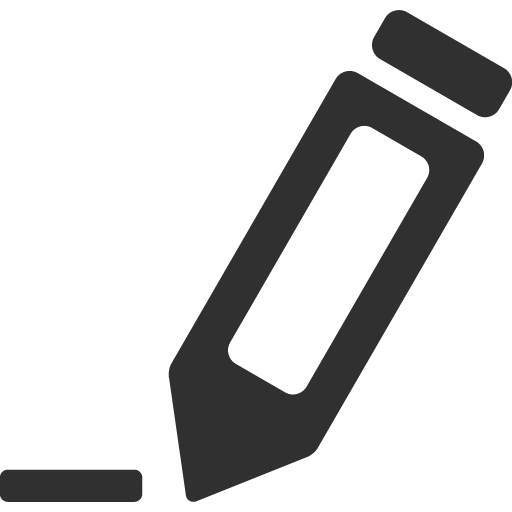
L’application de gestion des collaborateurs a pour but de permettre de visualiser différentes informations sur un collaborateur en particulier tel que ses informations personnelles (fonction, équipe, numéro de téléphone etc), ses clients, les CIS de ses clients et son backlog et de générer un rapport de ces modifications.

## Utilisation

Les deux applications nécessite d’être connecté sur **iTop**.

Sur l’application de gestion des clients, il est possible de sélectionner un client et/ou un CI à partir d’un formulaire. Si seul le client est renseigné, tous ces CIs seront présentés. Si seul le CIs est renseigné, seul le CI concerné sera présenté.

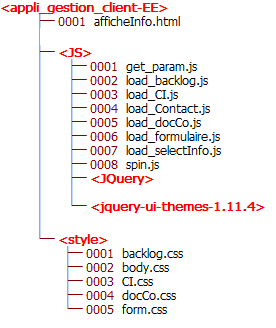
Pour modifier un ticket, il faut cliquer sur l’image suivante 🡪 , un pop-up apparait et permettra d’effectuer des modifications sur le statut et renseigner les champs supplémentaires liés au statut, l’agent et l’équipe et d’ajouter des commentaires privés et publics.



Pour générer le rapport, il faut cliquer sur le bouton « commencer le rapport », effectuer les modifications, puis cliquer sur le bouton « terminer et télécharger le rapport » qui déclenchera le téléchargement d’un fichier Excel. Le rapport restera en génération même en cas de fermeture de l’onglet ou du navigateur.

Sur l’application de gestion des collaborateurs, on va sélectionner le nom/prénom d’un collaborateur. Si le collaborateur sélectionné n’a pas de clients, l’onglet client ne sera pas disponible.

## Arborescence

Voici l’arborescence de l’application de gestion des clients:

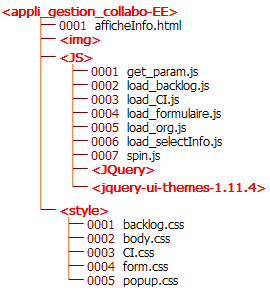
Le fichier qui est exécuté en premier est afficheInfo.html, il contient la structure de base de l’application et quelques initialisations de variables.

Dans le dossier JS on retrouve les fichiers JavaScript qui vont permettre à l’application de fonctionner.

Dans Le dossier JQuery et jquery-ui-themes-1.11.4, il y a les fichiers utiles pour pouvoir se servir de JQuery et des thèmes spéciaux.

Enfin, le dossier style contient les fichier CSS pour l’apparence de l’application.

Voici l’arborescence de l’application de gestion des collaborateurs:

La structure est quasiment la même que pour l’application de gestion des clients.

### Fichiers du dossier JS

get\_param.js : contient une fonction permettant de récupérer les paramètres de l’url de la manière suivante 🡪 $\_GET(‘NOMDUPARAMETRE’) ;

load\_backlog/CI/org/Contact/DocCo.js :contient des fonctions pour récupérer et afficher les informations sur le backlog du client ou du collaborateur/CI du ou des clients /client du collaborateur/contact du client/document de connexion du client.

**load\_backlog :** contient aussi des fonctions pour générer le pop-up de modification des tickets

load\_formulaire.js :contient des fonctions pour initialiser et vérifier le formulaire ainsi que pour initialiser des variables telles que le nom du client ou une liste de clients. Sert aussi pour l’auto-complétion. Ce fichier va appeler la fonction de chargement du contenu du premier onglet affiché (« CIs » dans le cas de l’application de gestion des clients et « client du collaborateur » dans le cas de l’application de gestion des collaborateurs).

load\_formulaire\_backlog.js *(appli gestion client, pas présent sur le schéma)* : contient des fonctions pour effectuer les modifications sur les tickets.

load\_rapport.js *(appli gestion client, pas présent sur le schéma)* : contient des fonctions pour générer le rapport des modifications de ticket.

load\_selectInfo.js :premier fichier JavaScript à être exécuté. Sert à vérifier si l’utilisateur est connecté sur iTop et charger les affichages en fonction.

spin.js :contient des fonctions pour permettre d’afficher une animation de chargement.

## Itop de recette

Pour faire des tests sans risquer d’endommager les données d’iTop de production, il faut mettre à jour les variables suivantes dans le fichier JS :

var ITOP\_URL = 'https://itoprec.hardis.fr';

var ITOP\_WS\_URL = ITOP\_URL + "/itop/webservices/rest.php?version=1.3";

Pour utiliser iTop de production il faut mettre à jour ces mêmes variables de la manière suivante :

var ITOP\_URL = 'https://itop.hardis.fr';

var ITOP\_WS\_URL = ITOP\_URL + "/webservices/rest.php?version=1.3";

## Technologies utilisées

L’application a été réalisée en HTML, CSS et JavaScript.

### Ajax

Dans cette application, Ajax permet d’effectuer des requêtes pour récupérer des informations ou effectuer des mises à jour par le biais d’un service informatique. Voilà comment on va procéder pour effectuer une requête Ajax :

* Définition d’une variable JSon

JSON = {

operation: 'core/get', //core/get = select

'class': 'DatabaseCluster', //classe de l’objet que l’on va récupérer

key: 'SELECT ci FROM FunctionalCI AS ci', //requête

output\_fields: 'id, name' //champs qui nous intéressent

};

* Requête Ajax

$.ajax({

type: "GET",

url: ITOP\_WS\_URL, // url du service informatique

dataType: 'jsonp',

data: { auth\_user: login, auth\_pwd: pwd, json\_data: JSON.stringify(JSON)}, //paramètres de l’url

crossDomain: 'true',

success: function (data) {

//traitement en cas de requête aboutie

// data = résultat de la requête (peut être une erreur)

},

error: function (data) {

//traitement en cas de requête échouée

}

});

Documentation pour la variable JSON: <https://wiki.openitop.org/doku.php?id=2_3_0:advancedtopics:rest_json>

### Animation de chargement

Une animation de chargement se déclenche lors de l’exécution des fonctions. Pour l’utiliser il faut :

* l’initialiser

spinner\_div = $('#spinner').get(0);

if(spinner == null) {

spinner = new Spinner(opts).spin(spinner\_div);

}else {

spinner.spin(spinner\_div);

}

* l’arrêter

spinner.stop(spinner\_div);

### Datepicker

Lorsque l’on clique sur l’image de modification du ticket un pop-up apparait avec la possibilité de sélectionner une date de planification pour le ticket en statut « en attente planifiée ». Voici comment on l’initialise :

Dans le fichier HTML :

<input type='text' id='datepicker' onclick='$(this).datepicker();’/>

Pour faire apparaitre le datepicker lorsque l’on clique sur le champ texte correspondant il suffit d’appeler la fonction datepicker sur l’élément dans un évènement onclick.

Dans le fichier JS :

$("#datepicker").datepicker({

altField: "#datepicker",

closeText: 'Fermer',

firstDay: 1,

dateFormat: 'dd/mm/yy',

monthNames: ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin', 'Juillet', 'Août', 'Septembre', 'Octobre', 'Novembre', 'Décembre'],

monthNamesShort: ['Janv.', 'Févr.', 'Mars', 'Avril', 'Mai', 'Juin', 'Juil.', 'Août', 'Sept.', 'Oct.', 'Nov.', 'Déc.'],

dayNames: ['Dimanche', 'Lundi', 'Mardi', 'Mercredi', 'Jeudi', 'Vendredi', 'Samedi'],

dayNamesShort: ['Dim.', 'Lun.', 'Mar.', 'Mer.', 'Jeu.', 'Ven.', 'Sam.'],

dayNamesMin: ['D', 'L', 'M', 'M', 'J', 'V', 'S'],

weekHeader: 'Sem.',

});

Ces lignes servent à initialiser le datepicker avec les valeurs standards pour le français.

### Tooltip



Pour afficher les commentaires long dans un popup, on va utiliser la class ‘tooltip’ de la manière suivante 🡪

<div class="tooltip">Voir la suite

<span class="tooltiptext"> COMMENTAIRE </span>

</div>

### Cookie et localStorage

Pour pouvoir garder les données du rapport, on utilise un cookie et spécialement pour notre cher ami Google Chrome qui n’autorise pas l’initialisation de cookie en local…. On utilise un objet LocaStorage. Plus d’information sur cet élément ici 🡪 <https://www.alsacreations.com/article/lire/1402-web-storage-localstorage-sessionstorage.html>.

Dans le fichier HTML, on va initialiser un nom pour le cookie comme suit : « rapport\_DATEDUJOUR ». Ensuite, lorsque l’on va cliquer sur le bouton « commencer le rapport », on va initialiser le cookie ou un objet LocalStorage si on est sur Chrome avec les fonctions présentent dans le fichier *load\_rapport.js* avec le nom de cookie initialiser plus tôt. Le cookie expirera dans 1 jour après son initialisation, mais pour l’objet LocalStorage il n’y a pas de limite.

Concernant le contenu du rapport, il sera alimenté par la fonction du fichier *load\_formulaire\_backlog.js,* lorsqu’une modification est effectuée, on vérifie si un cookie ou un objet LocalStorage est initialisé, et si c’est le cas, on va ajouter du contenu dans les variables.

Voici un exemple d’ajout de contenu dans les variables :

var content="&SL&" + $("#popupTicket h2").text() +"\*Ajout d'un commentaire privé\*-\*" + valueC;

if(!is\_chrome)

setCookie(cookieName, getCookie(cookieName) + content, 1);

else

setItem(localStorage.getItem(cookieName) + content);

**/ ! \** Pour la mise en forme Excel : pour ne pas avoir de problème avec le cookie (pas de points virgules et de caractère d’échappement) les « \* » et « &SL& » sont transformés dans la fonction finishRapport() du fichier *load\_rapport.js*

## Fonctions par fichier

Seuls les fichiers les plus volumineux sont décrits ici.

### load\_backlog.js (appli gestion client et collabo)

#### AjaxTicketFermePeriode(x, y) :

Fonction appelée par *RefreshSuccessfull(x)*.

Paramètre x= nombre de mois*.*

*Y=type du tiquet (‘UserRequest’ ou ‘Problem’)*

Permet d’effectuer une requête avec Ajax pour avoir tous les tickets fermés de type userrequest ou problem (paramètre y) sur une période donnée (paramètre x). Si la requête aboutie, la fonction *remplirTableauTicketFerme (x)* est appelée avec comme paramètre x= objet du résultat de la requête effectuée.

#### AjaxTicketP1SLAFermePeriode(x, y) :

Fonction appelée par *RefreshSuccessfull(x)*.

Paramètre x= nombre de mois*.*

Y=type du tiquet (‘UserRequest’ ou ‘Problem’)

Permet d’effectuer une requête avec Ajax pour avoir tous les tickets P1 et SLA de type userrequest ou problem sur une période donnée (paramètre x). Si la requête aboutie, la fonction *remplirTableauTicketP1SLSA(x)* est appelée avec comme paramètre x= objet résultat de la requête effectuée.

**ajaxProblem()** :

Fonction appelée par *refreshSuccessfull (x)*.

Paramètre x= nombre de mois*.*

Y=type du tiquet (‘UserRequest’ ou ‘Problem’)

Permet d’effectuer une requête avec Ajax pour avoir tous les tickets problem. Si la requête aboutie, la fonction va appeler les fonctions suivantes :

* -remplirTableauATEA(x, y) avec comme paramètre x= objets du résultat de la requête et y=0 pour le non vidage de tableau

-remplirTableauPb(),

-ajaxTicketFermePeriode(x, y) avec comme paramètre x= nombre de mois et y=’problem’

-ajaxTicketP1SLAPeriode(x, y) avec comme paramètre x nombre de mois et y=’problem’

#### LoadPageAfficheBacklog () :

Fonction appelé par le fichier *afficheInfo.html*.

Permet d’effectuer une requête avec Ajax, si la requête aboutie, la fonction *refreshSuccessfull(x)* est appelée avec comme paramètre x= résultat de la requête effectuée.

#### RefreshSuccessfull(x) :

Fonction appelée par *LoadPageAfficheBacklog()*.

Paramètre x= résultat de la requête effectuée par *LoadPageAfficheBacklog ()*.

Permet de vérifier si l’utilisateur est connecté sur iTop et charger les affichages en fonction. Si l’utilisateur est connecté, la fonction va appeler les fonctions suivantes :

* *remplirTableauATEA(x, y)* avec comme paramètre x= objets du résultat de la requête passée en paramètre et y=1 pour le vidage de tableau
* *remplirTableauCountIncDem()*,
* *remplirTableauCountInc()*,
* *remplirTableauCountDem()*,
* ajaxTicketFermePeriode(x, y) avec comme paramètre x= nombre de mois et y=’userrequest’
* ajaxTicketP1SLAPeriode(x, y) avec comme paramètre x= nombre de mois et y=’userrequest’

#### RemplirTableauATEA(x, y) :

Fonction appelée par *RefreshSuccessfull(x)*.

Paramètre x= objets du résultat de la requête passée en paramètre de *RefreshSuccessfull(x)*

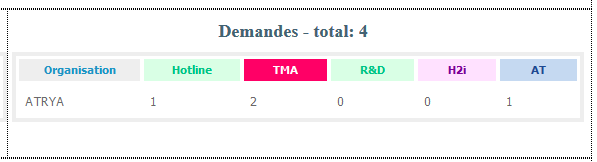
Y=1 ou 0 pour le vidage du tableau (1 = vider le tableau, 0= ne pas vider le tableau)

Permet de remplir le tableau des tickets à traiter et les tickets en attente.

#### RemplirTableauCountDem () :

Fonction appelée par *RefreshSuccessfull(x)*.

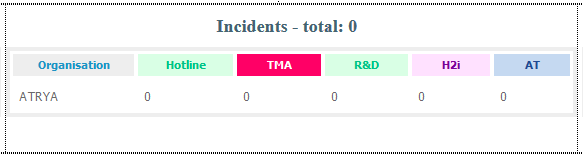
Permet de remplir le tableau des récapitulatifs du nombre de demandes.



#### RemplirTableauCountInc () :

Fonction appelée par *RefreshSuccessfull(x)*.

Permet de remplir le tableau des récapitulatifs du nombre de d’incidents.



#### RemplirTableauCountIncDem () :

Fonction appelée par *RefreshSuccessfull(x)*.

Permet de remplir le tableau des récapitulatifs du nombre des tickets.

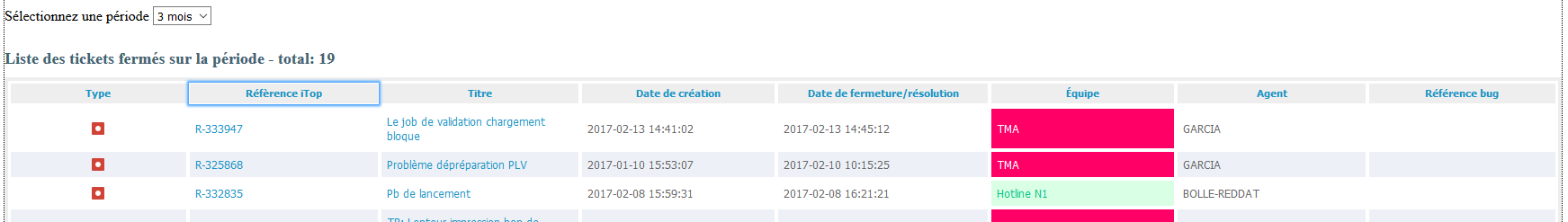


#### RemplirTableauTicketFerme(x, y) :

Fonction appelée par *RefreshSuccessfull(x)*.

Paramètre x= objet de la requête effectuée par *AjaxTicketFermePeriode(x, y)*

Y=1 ou 0 pour le vidage du tableau (1 = vider le tableau, 0= ne pas vider le tableau)

Permet de remplir de tableau des tickets fermés sur une période définie. 

#### RemplirTableauTicketP1SLSA(x, y) :

Fonction appelée par *RefreshSuccessfull(x)*.

Paramètre x= objet de la requête effectuée par *AjaxTicketFermePeriode(x, y)*

Y=1 ou 0 pour le vidage du tableau (1 = vider le tableau, 0= ne pas vider le tableau)

Permet de remplir de tableau des tickets P1 et SLA fermés sur une période définie.



### load\_backlog.js (appli backlog version éditable)

#### fadeInPopup (a, b, c, d, e, f, g, h, i, j, k, l) :

Fonction appelée lors d’un clic sur l’image de modification d’un ticket.

Paramètre a= type du ticket (userrquest ou problem)

b= id du ticket sur lequel on a cliqué

c= référence du ticket sur lequel on a cliqué

d= date de création du ticket sur lequel on a cliqué

e= titre du ticket sur lequel on a cliqué

f= date de mise à jour du ticket sur lequel on a cliqué

g= date de fermeture du ticket sur lequel on a cliqué

h= équipe assignée du ticket sur lequel on a cliqué

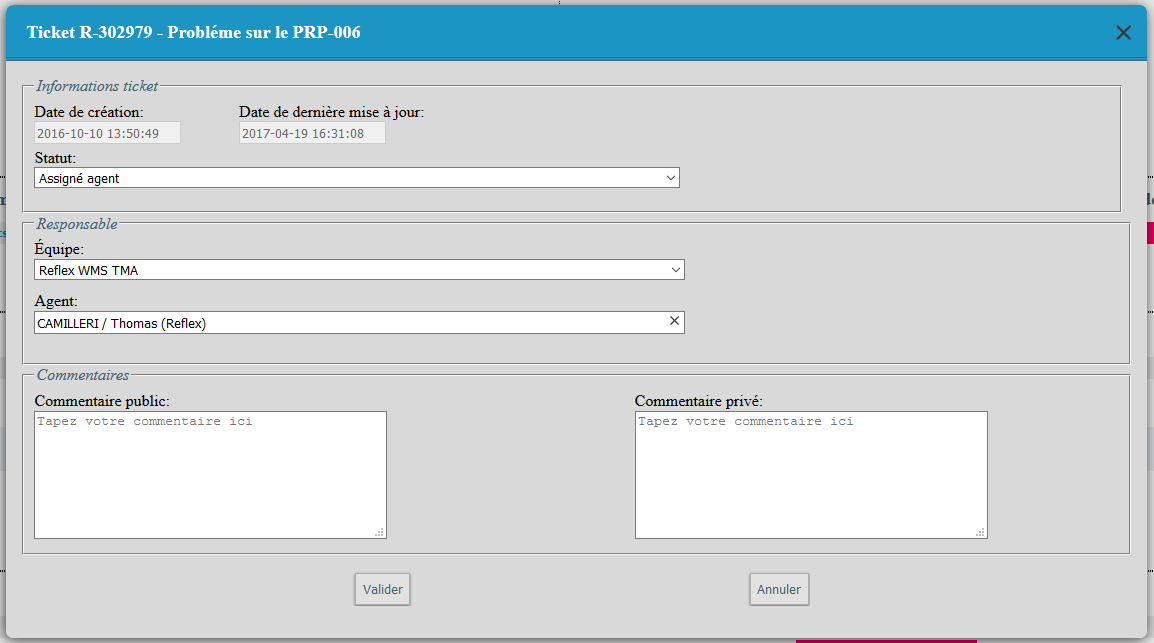
i= agent assigné du ticket sur lequel on a cliqué

j= statut du ticket sur lequel on a cliqué

k= service du ticket sur lequel on a cliqué

l= date planifiée du ticket sur lequel on a cliqué

Permet d’afficher le pop-up de modification en initialisant les champs avec les valeurs du ticket. Cette fonction sauvegarde aussi les valeurs de départ du ticket utile pour la génération du rapport.



### load\_CI (appli gestion client)

#### AjaxContactCI() :

Fonction appelée par AjaxlnkContactToContractCI*()*.

Permet d’effectuer une requête avec Ajax pour avoir tous les détails des contacts. Si la requête aboutie, le tableau des détails des contacts du client va être initialisé avec les objets du résultat de la requête et la fonction chargementPageCI(x, y)est appelée avec comme paramètre x= tableau de CIs et y= tableau de détails de CIs*.*

#### AjaxDBSCI () :

Fonction appelée par refreshSuccessful*(x)*.

Permet d’effectuer une requête avec Ajax pour avoir tous les schémas de bases de données. Si la requête aboutie, le tableau de schémas de base de données va être initialisé avec les objets du résultat de la requête et la fonction *ajaxVMCI()* est appelée*.*

AjaxFunctionalCICI():

Fonction appelé par LoadPageAfficheCI().

Permet d’effectuer une requête avec Ajax pour avoir tous les Cis. Si la requête aboutie, la fonction *refreshSuccessfull(x)* est appelée avec comme paramètre x= résultat de la requête effectuée.

#### AjaxlnkContactToContractCI() :

Fonction appelée par ajaxlnkSolutionToCICI *()*.

Permet d’effectuer une requête avec Ajax pour avoir tous les détails des CIs. Si la requête aboutie, le tableau des contacts du client va être initialisé avec les objets du résultat de la requête et la fonction ajaxContactCI*()* est appelée*.*

#### AjaxlnkSolutionToCICI() :

Fonction appelée par a*jaxVMCI()*.

Permet d’effectuer une requête avec Ajax pour avoir toute les détails des CIs. Si la requête aboutie, le tableau des détails de CIs va être initialisé avec les objets du résultat de la requête et la fonction *ajaxlnkContactToContractCI()* est appelée*.*

#### AjaxVMCI() :

Fonction appelée par a*jaxDBSCI()*.

Permet d’effectuer une requête avec Ajax pour avoir toute les machines virtuelles. Si la requête aboutie, le tableau des machines virtuelles va être initialisé avec les objets du résultat de la requête et la fonction *ajaxlnkSolutionToCICI()* est appelée*.*

#### ChargementPageCI(x, y) :

Fonction appelée par AjaxlnkContactToContractCI*()*.

Paramètre x= tableau de CIs alimenté par la fonction *refreshSuccessfullCI(x)*, y= tableau des détails des CIs alimenté par la fonction  *ajaxlnkSolutionToCICI().*

Permet d’alimenter les tableaux html avec les données de tableaux alimentés par les objets de résultat de requêtes.

#### LoadPageAfficheCI () :

Fonction appelé par le fichier *afficheInfo.html*.

Permet d’initialiser les variables Json et appelle la fonction *ajaxFunctionalCICI()*.

#### RefreshSuccessfullCI(x) :

Fonction appelée par AjaxFunctionalCICI*()*.

Paramètre x= résultat de la requête effectuée par AjaxFunctionalCICI*()*.

Permet de vérifier si l’utilisateur est connecté sur iTop et charger les affichages en fonction. Si l’utilisateur est connecté, la fonction va initialiser le tableau de CIs avec les object du paramètre et va appeler *ajaxDBSCI()*.

### load\_CI (appli gestion collabo)

Les fonctions du fichier load\_CI.js de l’application de gestion des collaborateurs est similaire à celles de l’application de gestion des clients excepté que toutes les fonctions ont un paramètre qui est un nom de client. Dans la dernière fonction appelée (*chargementPageCI(x, y, z)*) on va vérifier s’il y a encore des clients dans la liste et réappeler la première fonction (*loadPageCI(x)*) pour recommencer le traitement s’il y en a encore.