Détail sur le code

Table des matières

[**Arborescence de l’application** 2](#_Toc73369509)

[**Scénario type : utilisation de l’application** 3](#_Toc73369510)

[**Aperçu du fichier « index.html »** 8](#_Toc73369511)

[**Aperçu des services** 8](#_Toc73369512)

[**auth-guard.services.ts** 8](#_Toc73369513)

[**auth.services.ts** 9](#_Toc73369514)

[**vizjs-view.services.ts** 10](#_Toc73369515)

[**Aperçu des components** 12](#_Toc73369516)

[**Component auth** 12](#_Toc73369517)

[**Component vizjs-view** 14](#_Toc73369518)

[**Code lié au component (variables, constructeur, ngOnInit, ngOnDestroy)** 16](#_Toc73369519)

[**Fonctions utilisées pour gérer le timeout d’une session utilisateur** 20](#_Toc73369520)

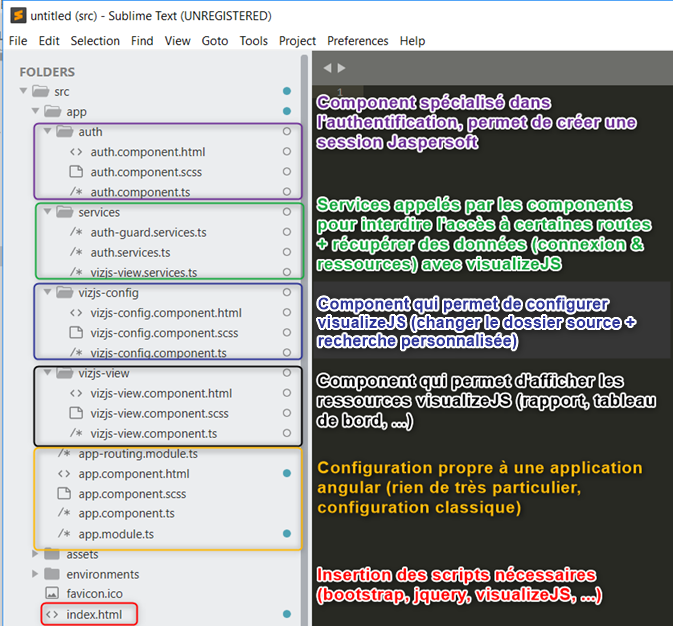
[**Fonctions utiles pour modifier la vue HTML du component** 21](#_Toc73369521)

[**Fonctions qui utilisent visualizeJS pour afficher / interagir avec des ressources Jaspersoft** 31](#_Toc73369522)

[**Component vizjs-config** 58](#_Toc73369523)

# **Arborescence de l’application**

Tout le code est contenu dans le dossier « src » de l’application, en voici un bref aperçu :

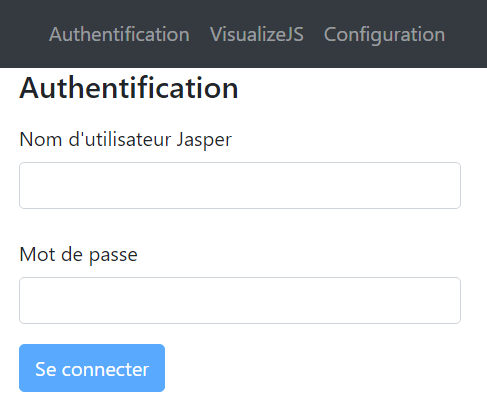


# **Scénario type : utilisation de l’application**

**1) Arrivée sur la page d’authentification**

Il faut se connecter en utilisant les identifiants Jaspersoft

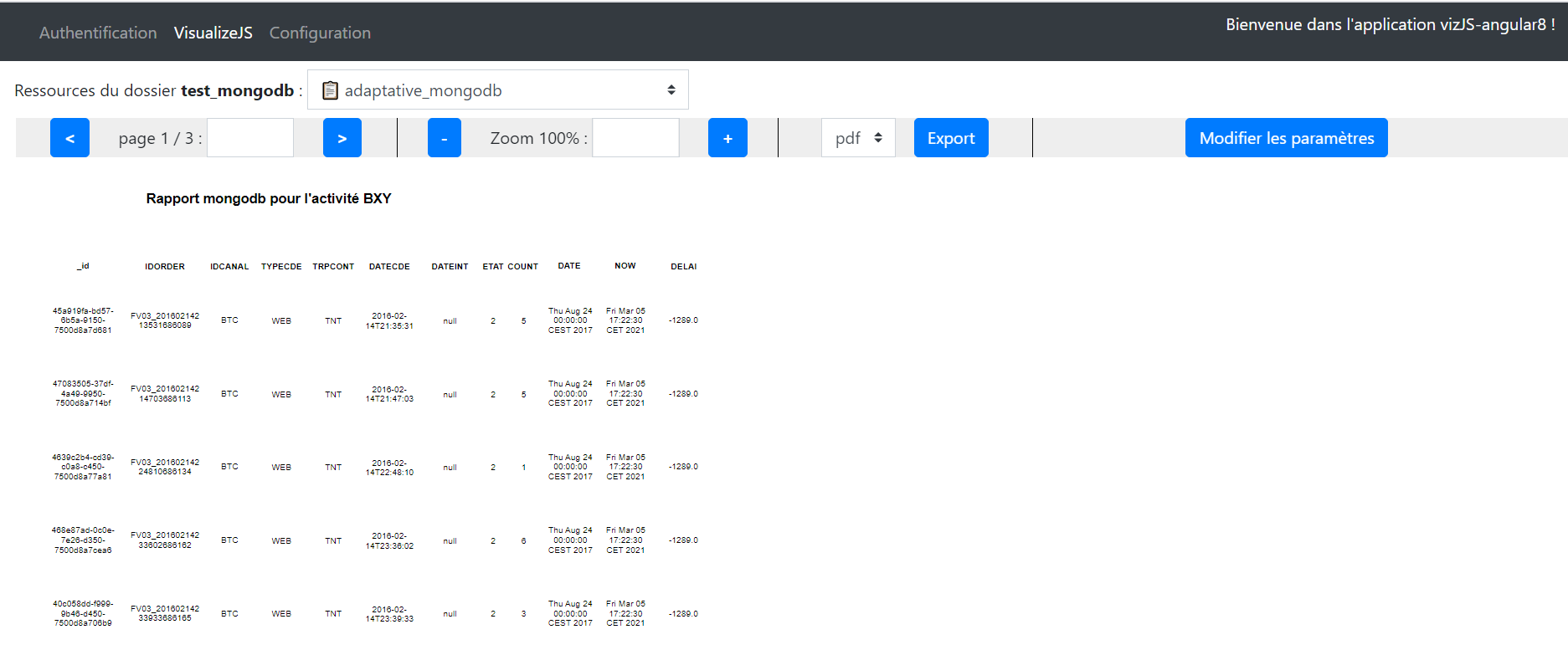
(coté code, il faudra se rendre dans le fichier auth.component.ts pour avoir tous les détails)



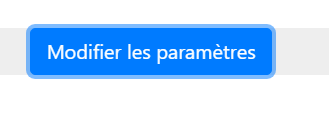
**2) Redirection vers vizjs-view**

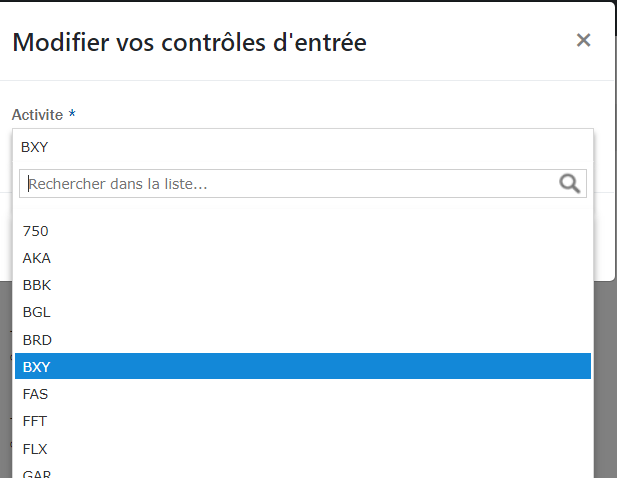
Si les identifiants sont corrects, l’utilisateur est redirigé vers la page vizjs-view qui contiendra les différentes ressources accessibles, il pourra modifier les paramètres / changer de ressource à afficher à tout moment.

(coté code, il faudra se rendre dans le fichier vizjs-view.component.ts, le mode « recherche personnalisé » sera désactivé)

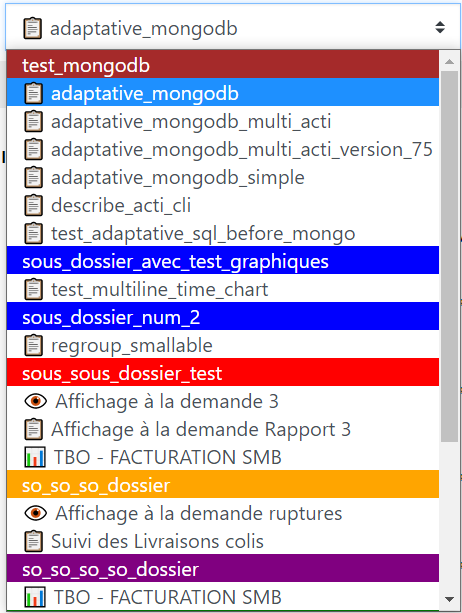


Modification de paramètres

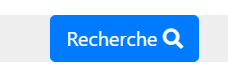
****

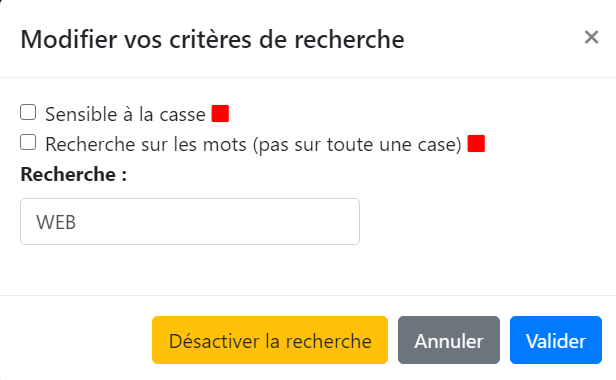
****

Changement de ressource

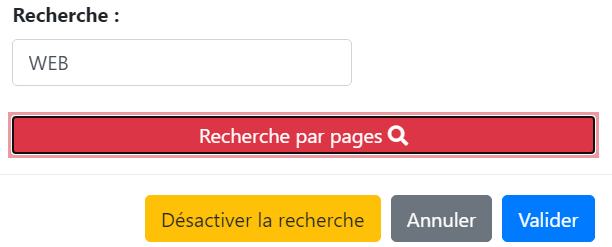
****

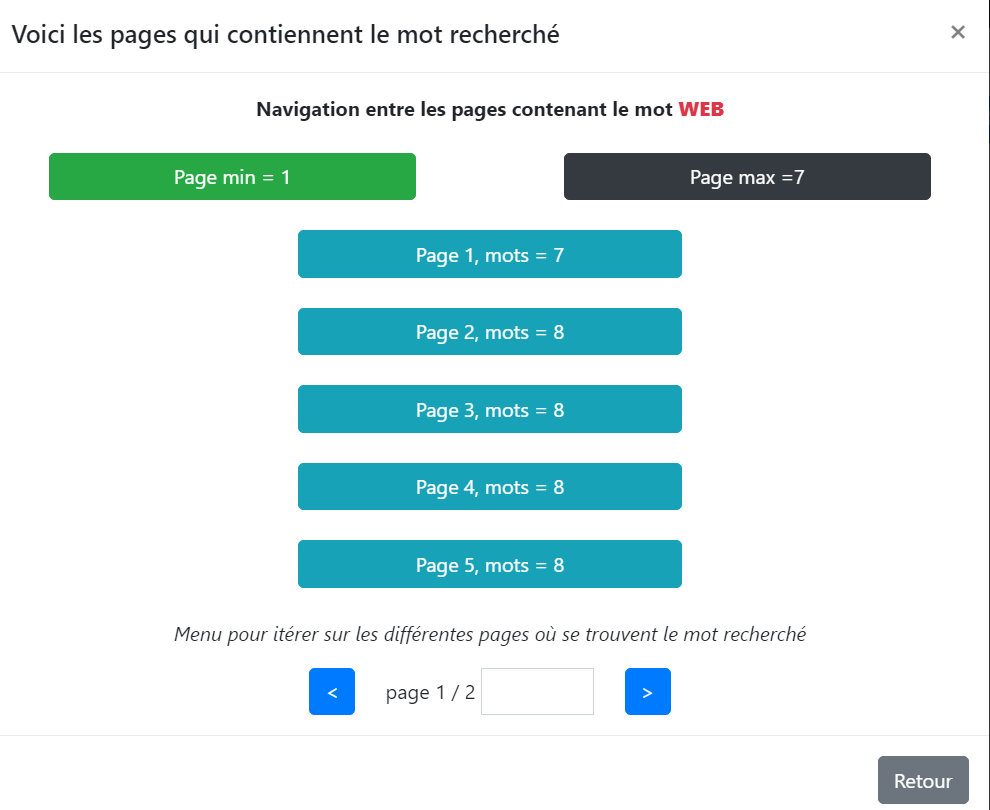
Recherche d’un mot clé dans le rapport





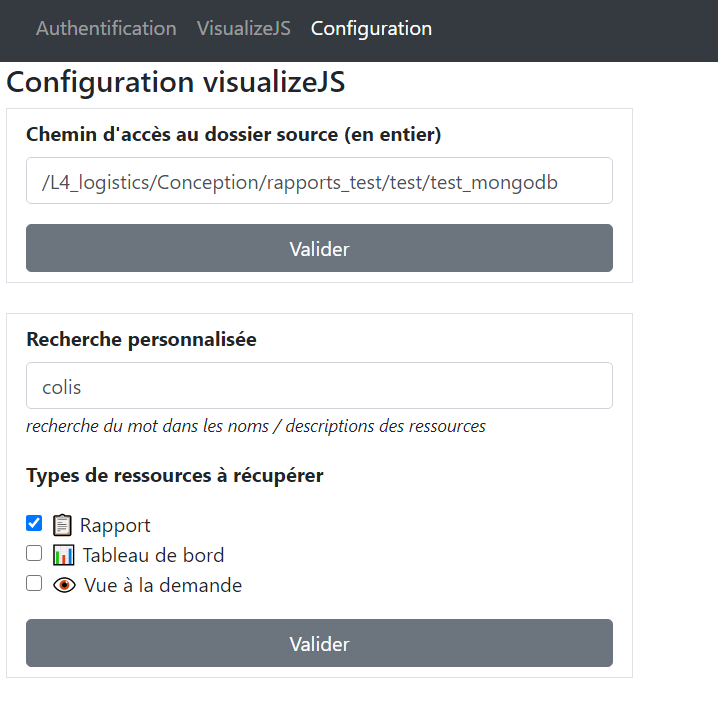
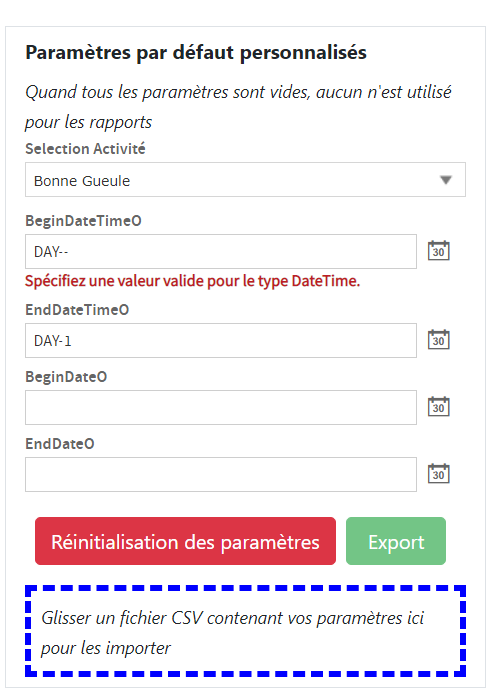






**3) Modifier les paramètres visualizeJS dans vjs-config**

En cliquant sur l’onglet « configuration », il est ensuite possible de réaliser une recherche personnalisée, de modifier le dossier source ou encore les paramètres utilisés par défaut côté vizjs-view (ex : tous les rapports s’exécutent automatiquement sur l’activité SMB).

** **

En modifiant le dossier source, l’utilisateur pourra accéder à de nouvelles ressources dans la page visualize JS.

Pour que la modification soit validée, il faut que le chemin entré soit valide et que l’utilisateur soit autorisé a y accéder. (pour l’instant : simple test de la correspondance avec une liste, il faudrait autoriser l’accès avec les droits Jasper)

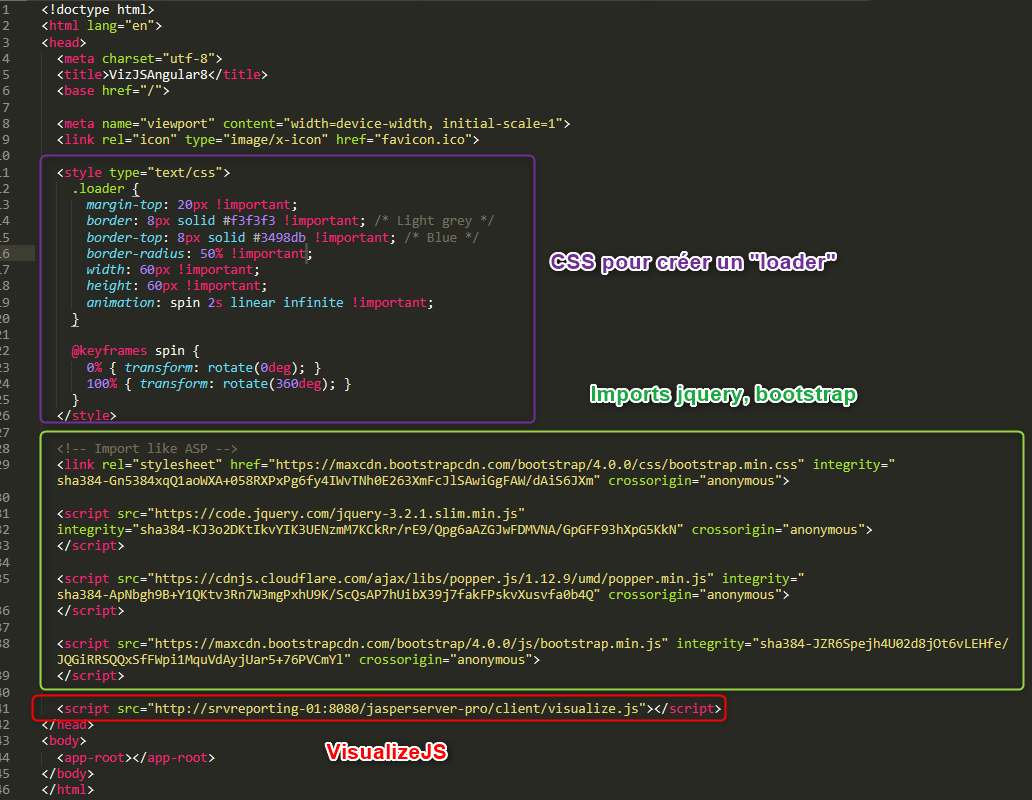
Avec le mode recherche personnalisée, seules les ressources contenant un certain terme (comme « colis » par exemple) et d’un certain type seront affichées dans la page visualize JS.

Les paramètres par défaut sont utilisés lors du premier chargement d’un rapport côté vizjs-view (page visualize JS).

Dès qu’un des paramètres est rempli et validé, il est automatiquement utilisé.

Pour désactiver ces paramètres par défaut, il faut qu’ils soient tous vides (c’est ce que permet de faire le bouton « Réinitialisation des paramètres »)

# **Aperçu du fichier « index.html »**

****

# **Aperçu des services**

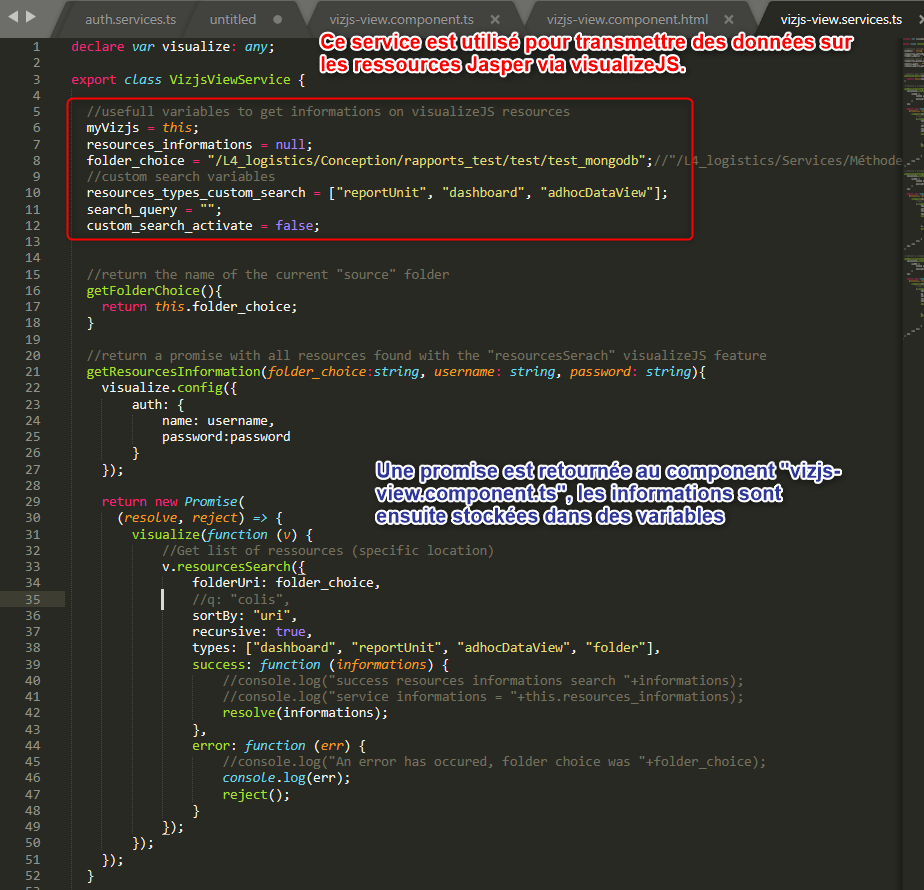
## **auth-guard.services.ts**



## **auth.services.ts**



## **vizjs-view.services.ts**







# **Aperçu des components**

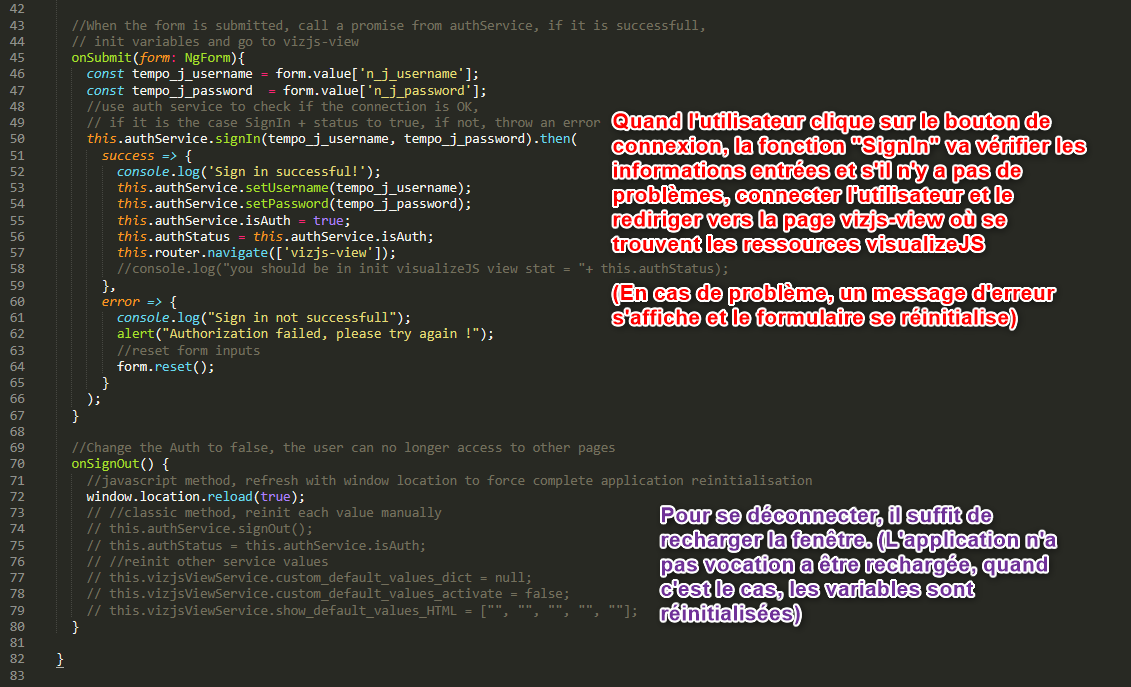
## **Component auth**

Partie HTML



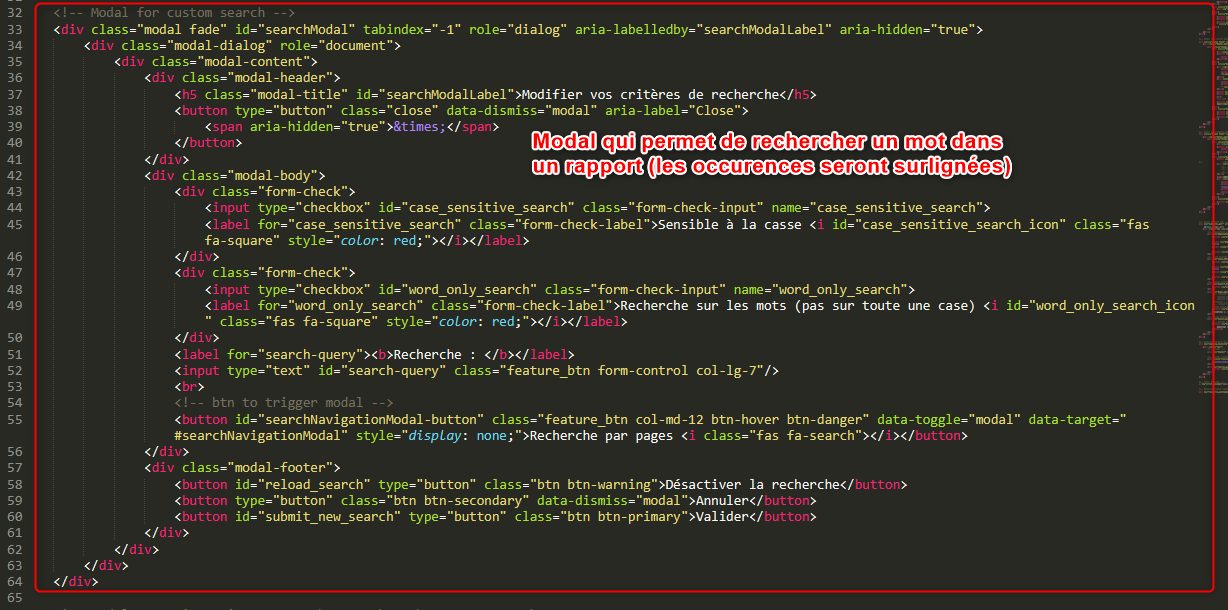
Partie Typescript

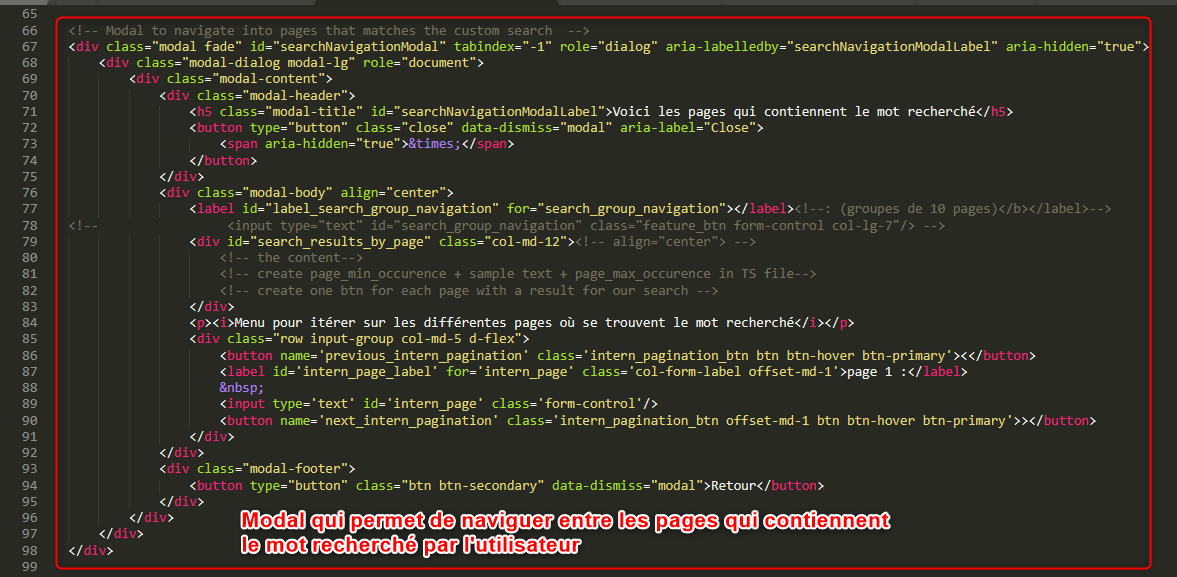
****

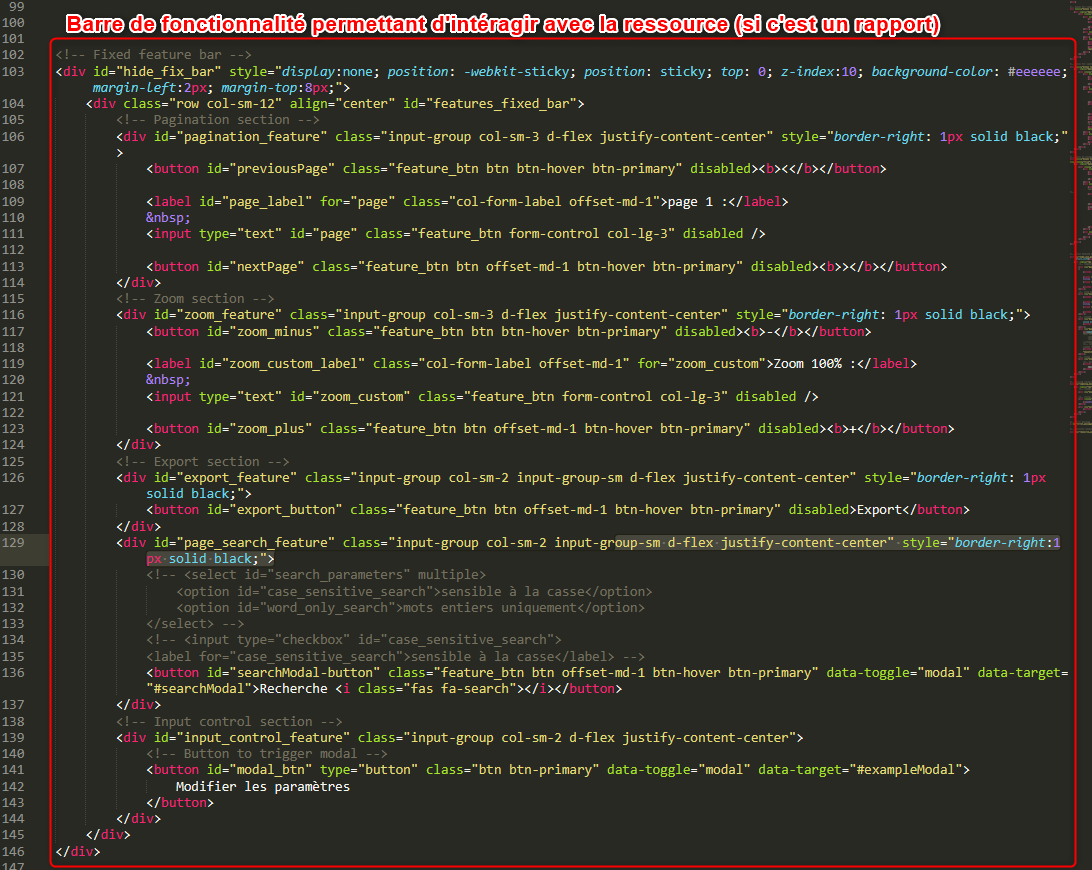
****

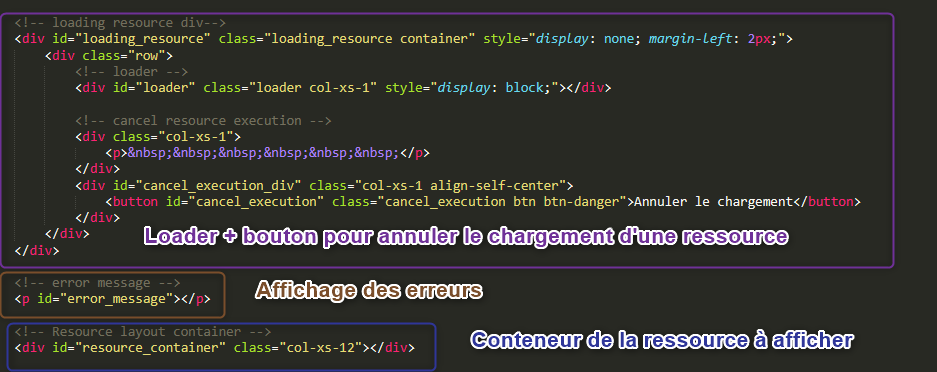
## **Component vizjs-view**

Partie HTML









Partie Typescript

(Vu que ce fichier Typescript est très volumineux et qu’il est modifié régulièrement, l’affichage sera différent pour cette partie de la documentation, tout le code sera séparé en portions et commenté dans des tableaux)

### **Code lié au component (variables, constructeur, ngOnInit, ngOnDestroy)**

*Imports, création des variables et constructeur*

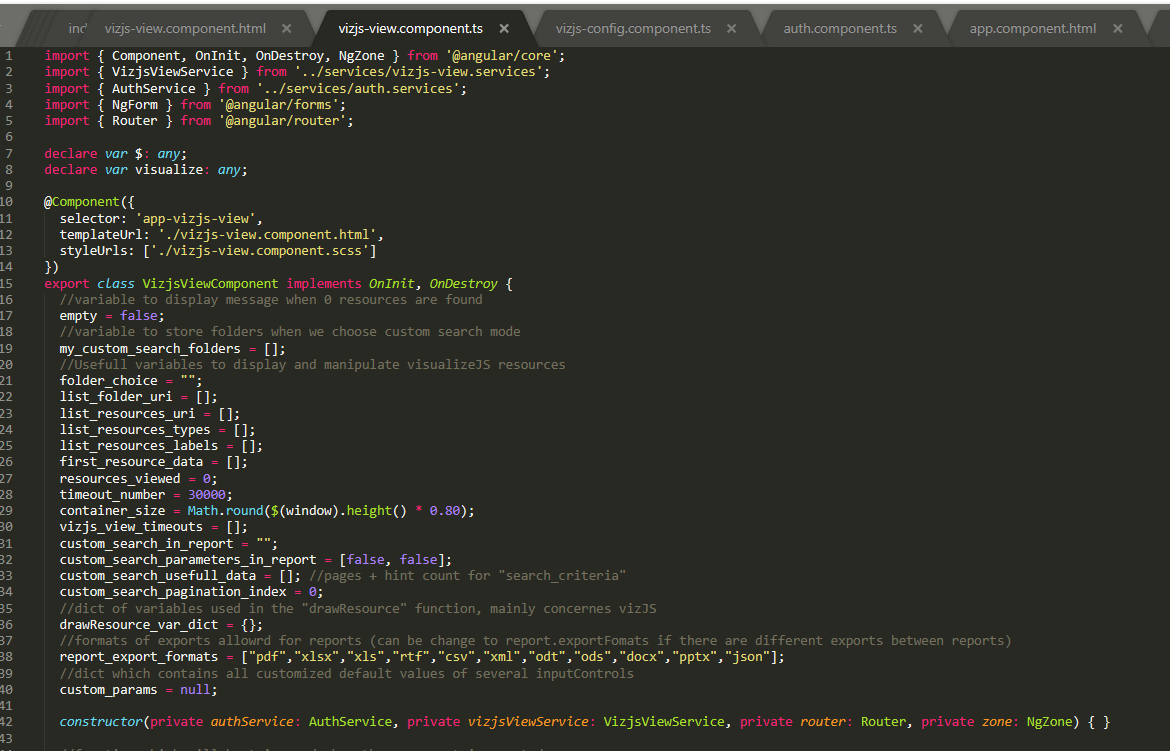
**Brève description :**

Pour que le component puisse fonctionner, il est nécessaire d’importer des librairies propres à Angular (NgForm, OnInit, NgZone, …), de définir des variables globales (modifiables facilement dans n’importe qu’elle fonction) ainsi qu’un constructeur implémentant les différents services que nous allons utiliser.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Les variables définies ici seront utilisées dans les fonctions de ce fichier TS. La plupart servent uniquement à conserver des valeurs utiles, quelques exemples :  - « Empty » permet d’afficher un message d’erreur quand aucune ressource n’est trouvée  - « timeout\_number » correspond au nombre de millisecondes maximum qu’une ressource peut prendre pour être chargée dans l’application  - « container\_size » adapte la taille du conteneur selon la hauteur de l’appareil  - « vizjs\_view\_timeouts » stocke l’ensemble des fonctions timeout en cours dans un tableau  - « drawResource\_var\_dict » est un dictionnaire stockant de nombreuses variables qui ont vocation à être utilisées dans la fonction « drawResource »  - « custom\_params » est un dictionnaire contenant les paramètres par défaut personnalisés entrés par l’utilisateur dans la page « vizjs-config », ils sont utilisés automatiquement lors de la première génération d’un rapport Jasper |

**1**



**1**

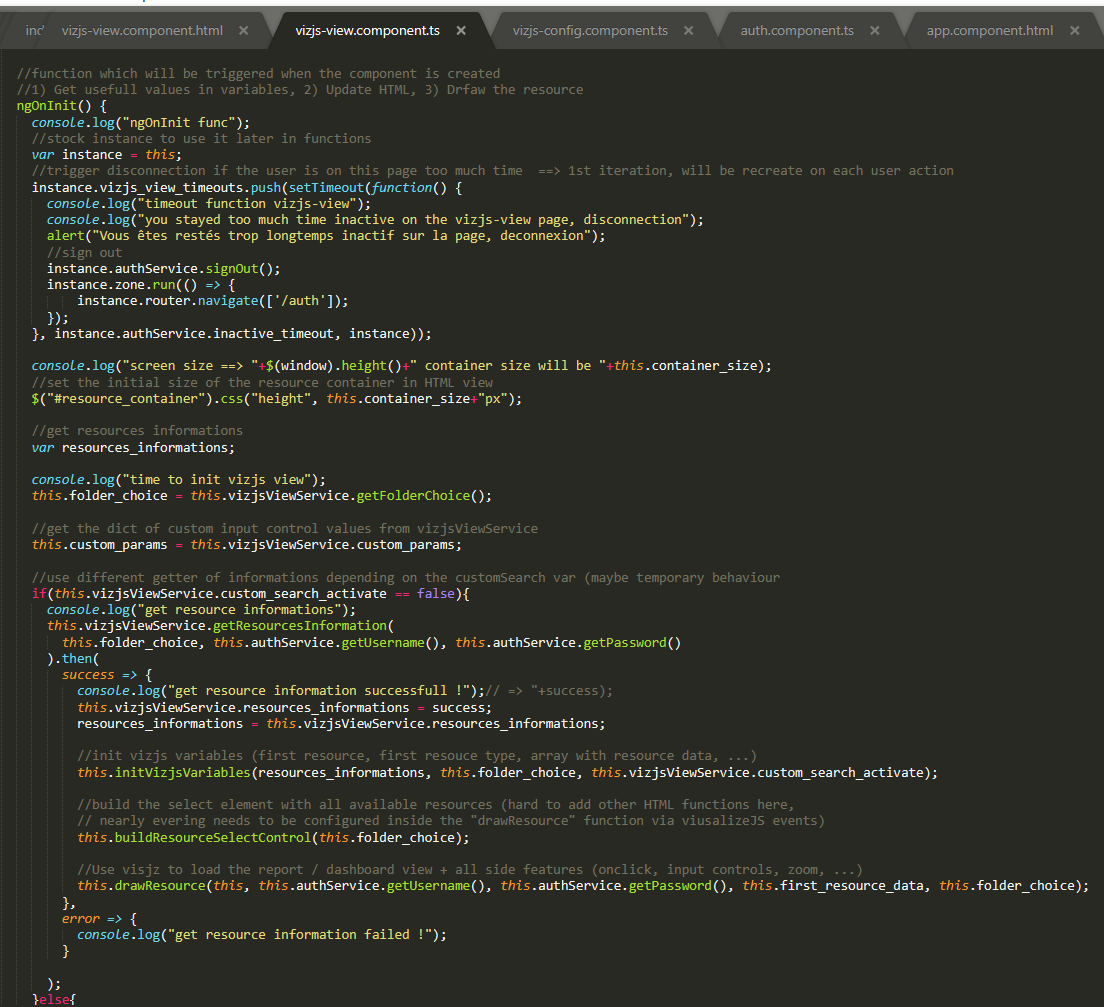
*Fonction ngOnInit (exécutée quand le component est créé)*

**Brève description :**

Ensemble de traitements à réaliser quand le component est créé. Le traitement sera légèrement différent si l’utilisateur a activé le mode « recherche personnalisé » avec le component « vizjs-config ».

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Si la recherche personnalisée n’est pas activée |
| 1.1 | Si aucune ressource n’a pu être récupérée, renvoyer l’utilisateur sur la page de config |
| 1.2 | Récupération des informations visualizeJS |
| 1.3 | Initialisation des variables |
| 1.4 | Construire le « select » qui liste les ressources |
| 1.5 | Charger la ressource à afficher (+fonctionnalités visualizeJS comme le changement de page par exemple) |
| 2 | Si la recherche personnalisée est activée, le processus va un peu changer |
| 2.1 | On récupère d’abord les sous-dossiers du dossier source |
| 2.2 | Puis les ressources qui correspondent aux critères de recherche |
| 2.3 | Si aucune ressource n’est disponible, on envoie un message à l’utilisateur |
| 2.4 | On construit ensuite le select qui liste les ressources et on charge la ressource à afficher |



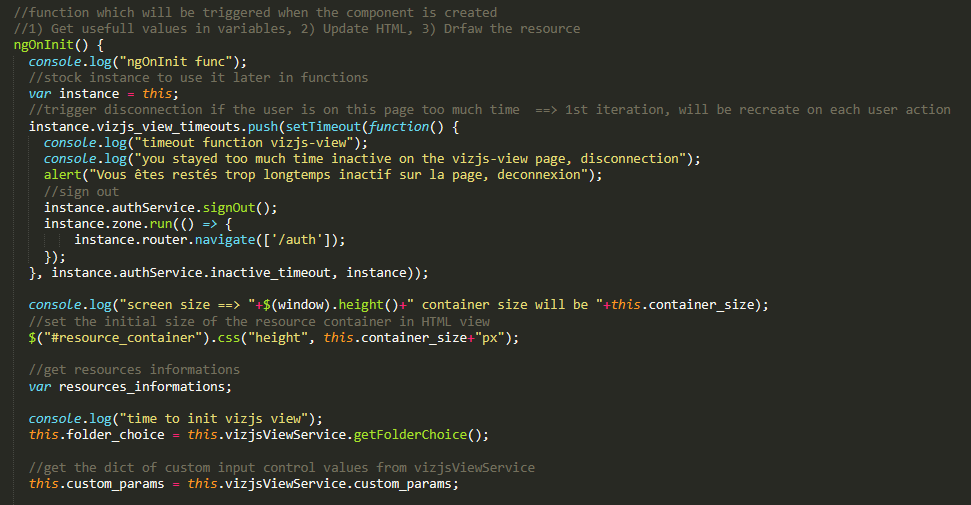
**1**

**1.2**

**1.1**

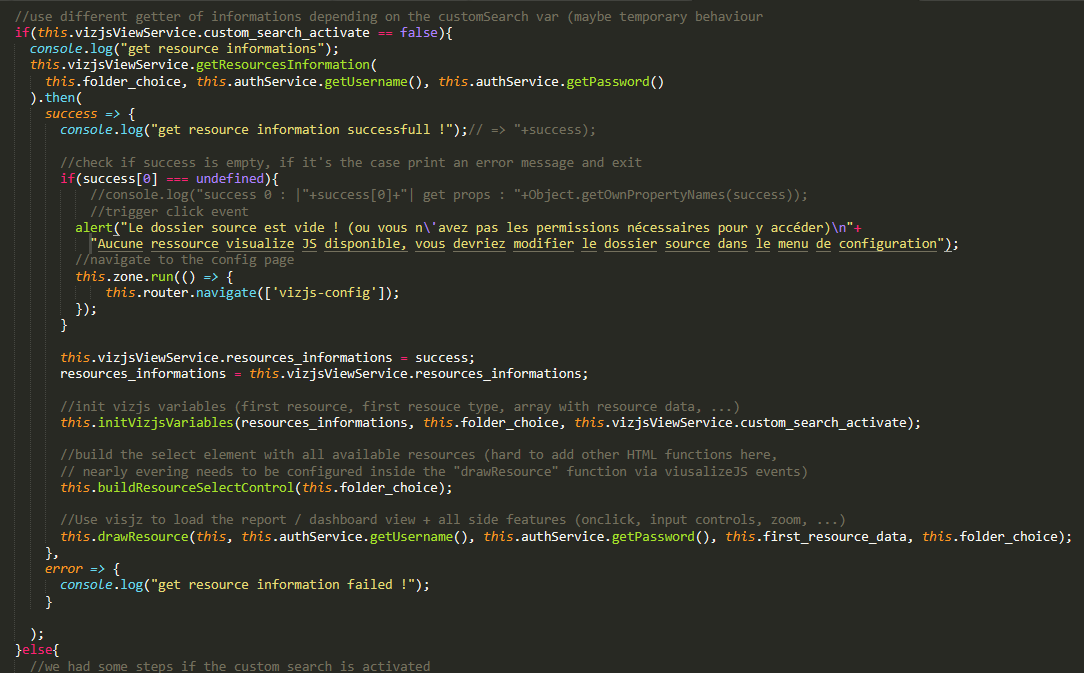
**1.3**

**1.4**



**Réinitialisation de la session utilisateur et récupération de quelques variables**

**(folder\_choice + custom\_params)**



**1.2**

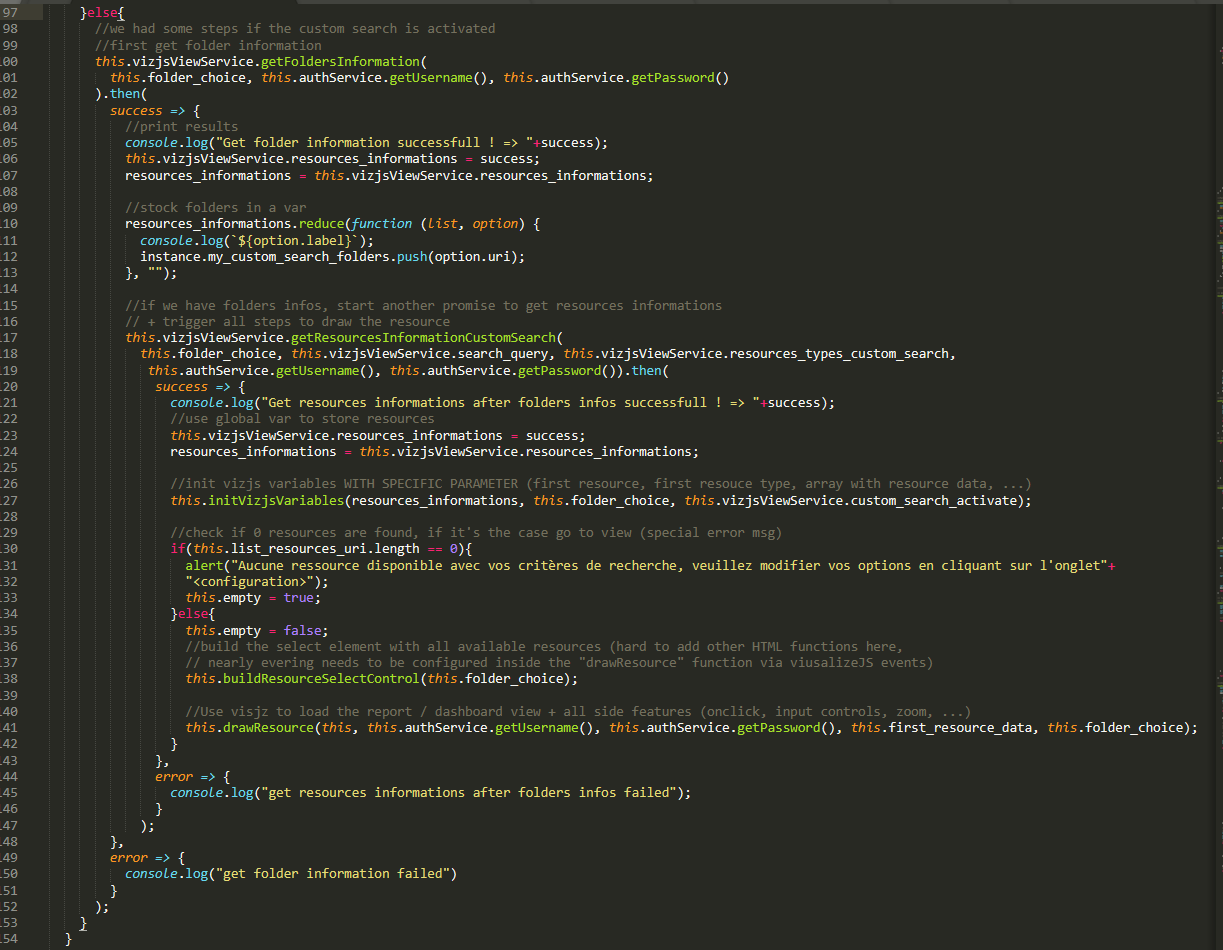
**1.1**

**1.3**

**1.5**

**1.4**

**1**



**2**

**2.4**

**2.3**

**2.2**

**2.1**

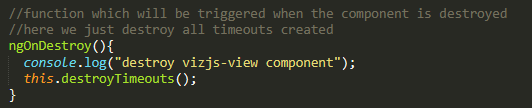
*Fonction\_ngOnDestroy*

**Brève description :**

Ensemble de traitements à réaliser quand le component est détruit. En l’occurrence, le seul traitement à réaliser ici est la suppression des différents « timeouts » créés (fonctions qui doivent s’exécuter X minutes après leur définition).

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Appel de la fonction « destroyTimeouts » pour détruire le tableau contenant l’ensemble des fonctions timeout |



**1**

### **Fonctions utilisées pour gérer le timeout d’une session utilisateur**

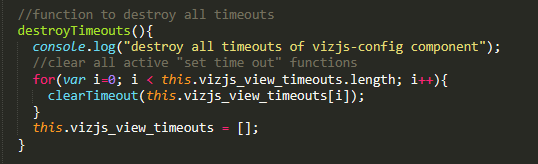
*Fonction destroyTimeouts*

**Brève description :**

Fonction permettant de détruire l’ensemble des fonctions timeout actuellement définies. Elle est utilisée pour éviter de réaliser des traitements inutiles. Par exemple, si un rapport a réussi à s’exécuter sans problème, il ne sert à rien d’exécuter une fonction timeout après 30 sec pour vérifier que tout s’est bien passé.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Appel de la fonction « clearTimeout » sur l’ensemble des fonctions « timeout » stockées dans le tableau « vizjs\_view\_timeouts » pour les désactiver.  A la fin du traitement, vider le tableau. |



**1**

*Fonction recreateInactiveTimeout*

**Brève description :**

Cette fonction permet recréer une fonction timeout qui se déclenchera si l’utilisateur reste inactif trop longtemps. Cette fonction est appelée très souvent (à chaque « action » de l’utilisateur) pour ne pas déconnecter l’utilisateur s’il est actif.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Destruction des fonctions timeout |
| 2 | Création d’une nouvelle fonction timeout qui se déclenchera dans X minutes si l’utilisateur n’a pas fait d’ « actions » (changement de ressource, modification de paramètres, …) |
| 3 | Déconnexion et redirection de l’utilisateur sur la page d’authentification si la fonction timeout s’exécute |



**3**

**2**

**1**

### **Fonctions utiles pour modifier la vue HTML du component**

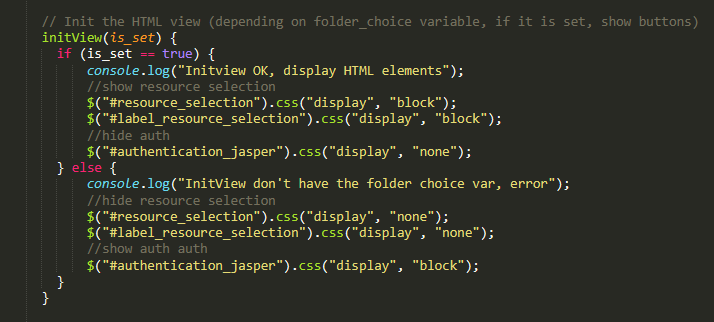
*Fonction initView*

**Brève description :**

Cette fonction affiche ou cache des éléments HTML selon la ressource à charger, si la ressource est un rapport, il faudra afficher une barre de fonctionnalités pour le manipuler. Dans le cas contraire, il faudra la cacher.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Afficher la barre de fonctionnalités si la ressource est un rapport (pages, zoom, recherche de mot, …) |
| 2 | Si la ressource n’est pas un rapport, cacher la barre de fonctionnalités |



**2**

**1**

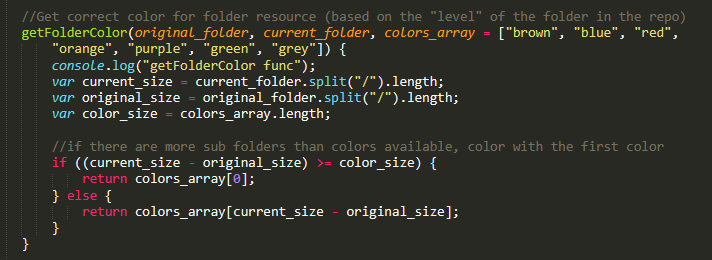
*Fonction getFolderColor*

**Brève description :**

Fonction qui permet de colorier les dossiers selon leur « emplacement » dans l’arborescence de fichiers. Exemple : les dossier « folder\_A » et « folder\_B » seront marrons et le sous-dossier « subfolder\_A » sera bleu

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Retour d’une couleur selon le nombre de « / » qui séparent le dossier source du dossier que l’on souhaite colorier dans notre élément « select » (liste des ressources disponibles) |



**1**

*Fonction buildResourceSelectControl*

**Brève description :**

Fonction qui va construire un élément « select » qui contiendra l’ensemble des ressources disponibles pour l’utilisateur (dépend du dossier source choisi).

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Pour chaque dossier, une couleur va être attribuée (fonction « getFolderColor ») + un élément (disabled) va être ajouté au « select » |
| 2 | Chaque ressource appartenant au dossier actuellement traité va ensuite être ajoutée dans le « select » (avec un affichage différencié selon son type) |
| 3 | Appliquer les modifications HTML |



**3**

**1**

**2**

*Fonction hideOrShowReportHTMLElements*

**Brève description :**

Fonction qui permet de modifier la vue HTML selon le type de ressource à afficher (similaire à initView dans le fonctionnement, les deux pourraient être fusionnées).

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Affichage de la barre de fonctionnalités si la ressource est un rapport |
| 2 | Cacher la barre de fonctionnalité et remettre à jour la taille du conteneur quand la ressource n’est pas un rapport |



**2**

**1**

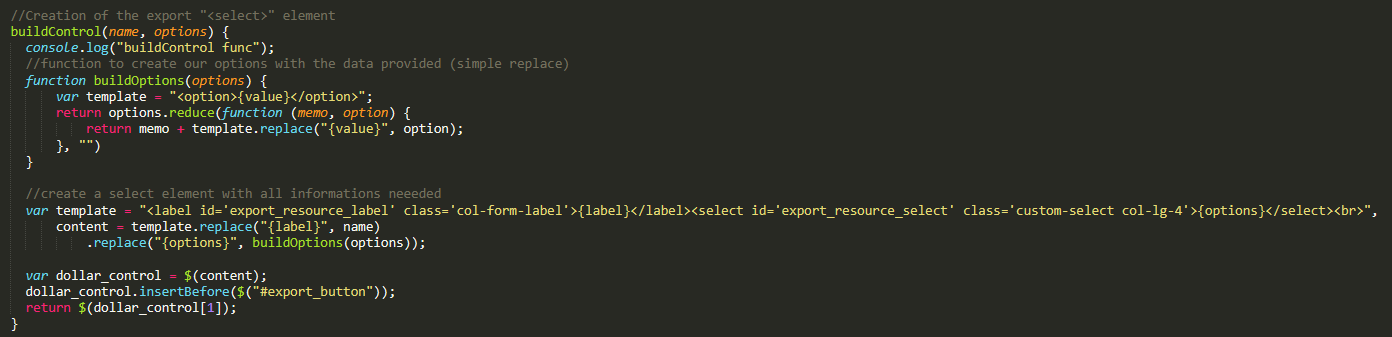
*Fonction buildControl*

**Brève description :**

Fonction qui permet de construire l’élément « select » à partir de formats d’exports de données fournis en paramètres. Ce « select » est dédié à l’exportation d’un rapport Jaspersoft.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Sous fonction permettant de créer les options de notre élément « select » |
| 2 | Création et insertion de l’élément « select » dans le code HTML |



**2**

**1**

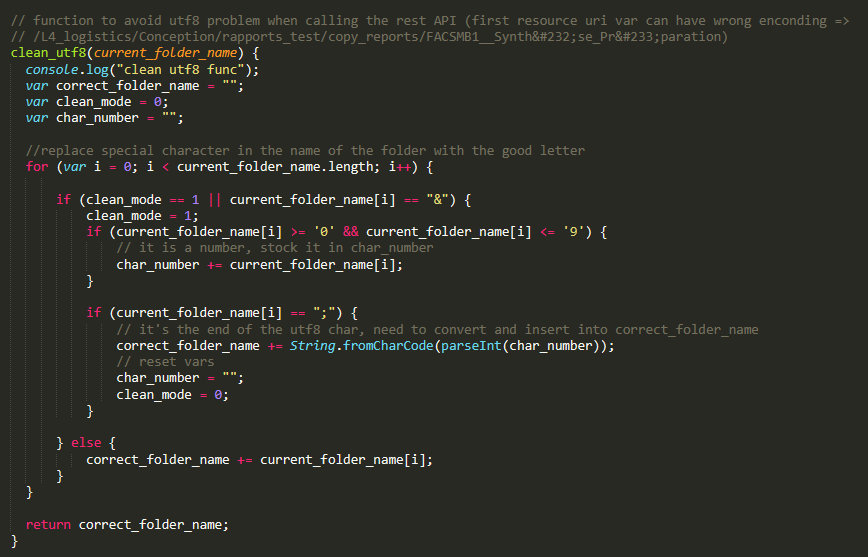
*Fonction clean\_utf8*

**Brève description :**

Fonction qui permet de modifier un « chemin d’accès à un dossier jasper » pour qu’il corresponde au format utf8.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | On itère sur les caractères, si un « & » est détecté, on change de mode et on va convertir le charCode donné en utf8 |



**1**

*Fonction customSearchInReport*

**Brève description :**

Fonction qui va rechercher un mot entré par l’utilisateur dans un rapport (page courante) selon certains critères (sensibilité à la casse ? / mots entiers uniquement ?). Quand le mot est trouvé, le code HTML est modifié pour surligner ses occurrences en bleu.

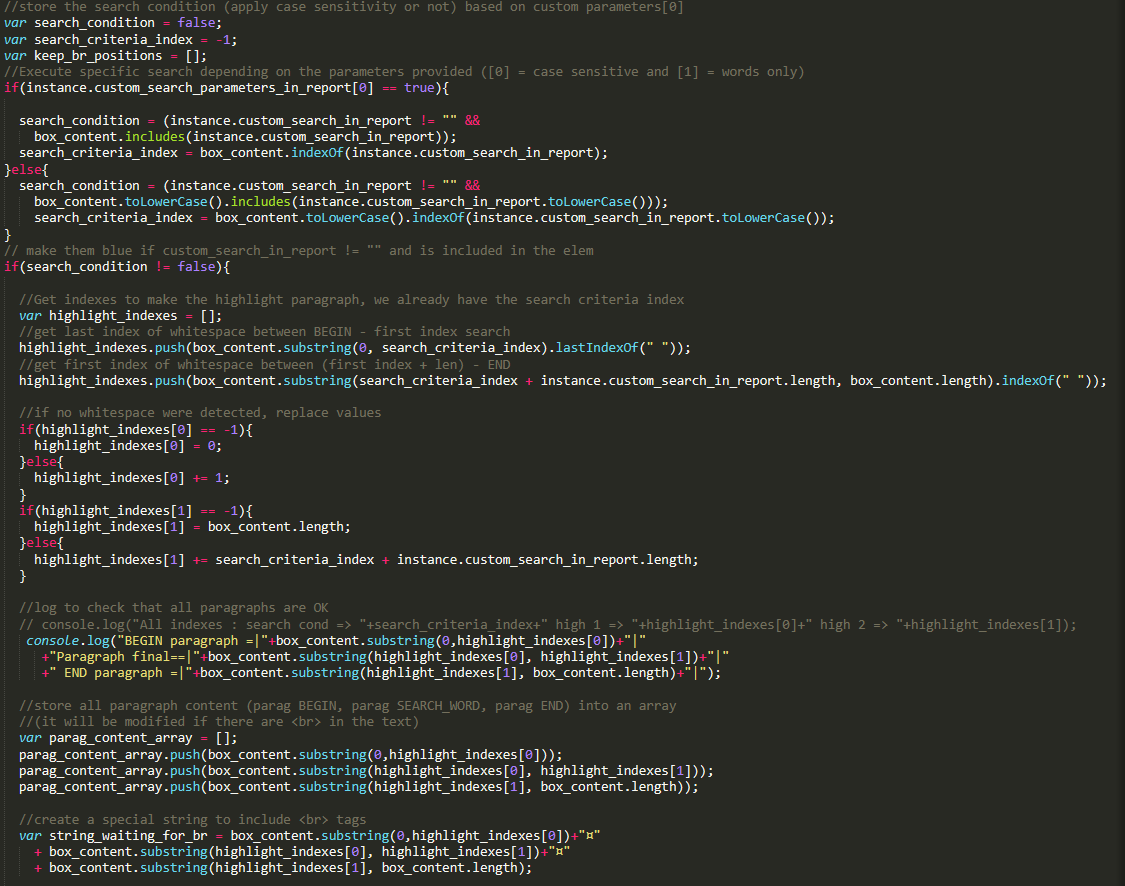
**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Itérer sur chaque élément HTML contenu dans la page du rapport, si un contenu est une balise « <span> », ne peut pas le récupérer pour éviter les problèmes. |
| 2 | Si le mode « mots entiers uniquement » a été activé, faire des vérifications pour s’assurer que le mot a été trouvé en respectant les critères. Si c’est le cas le traitement pour cet élément HTML peut se poursuivre. |
| 3 | Effectuer une recherche du mot donné par l’utilisateur dans l’élément HTML (en testant la sensibilité à la casse si nécessaire).  Si le mot est trouvé, récupérer les différents index permettant de délimiter le mot recherché du reste de l’élément HTML + trouver les positions des « <br> » dans l’élément originel pour les replacer + construction de l’élément HTML final (où le mot recherché est surligné en bleu) |

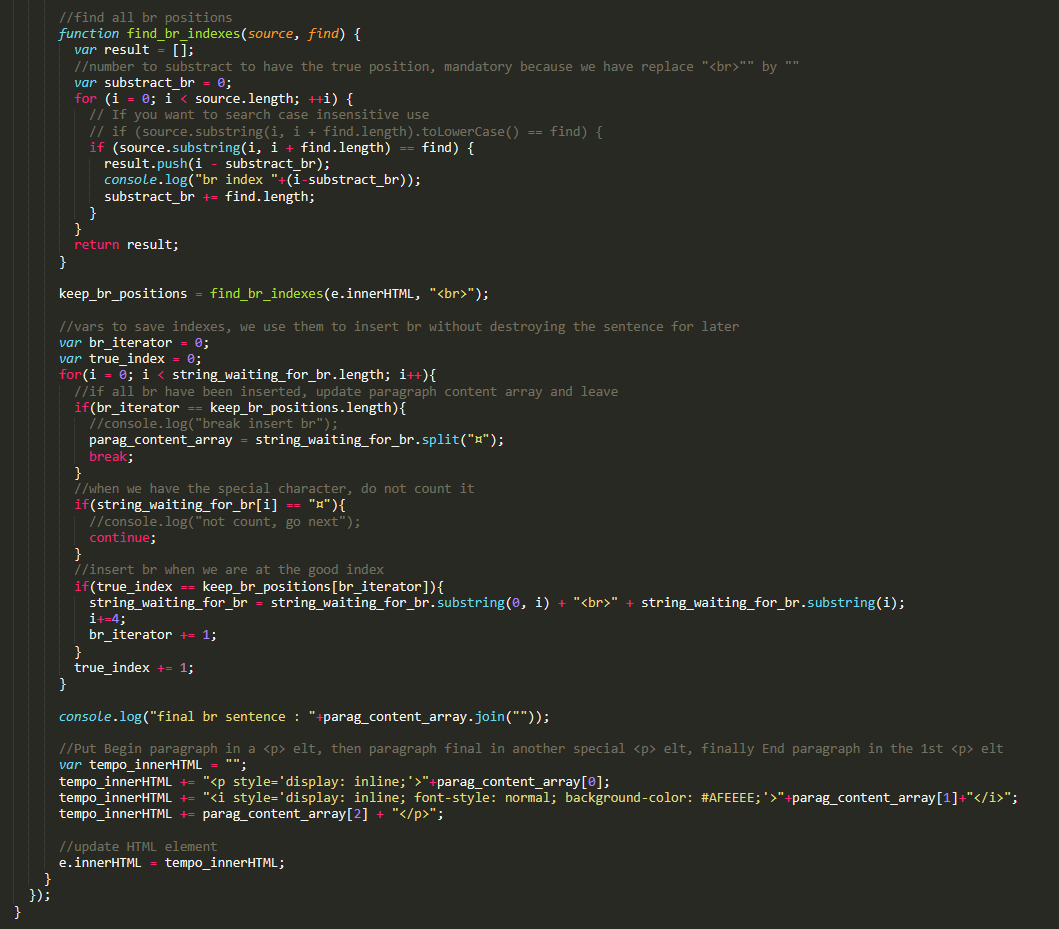
****

**1**

**2**

****

**3**

****

**3**

**Fonctions pour initialiser visualizeJS (obtenir les informations sur les ressources disponibles)**

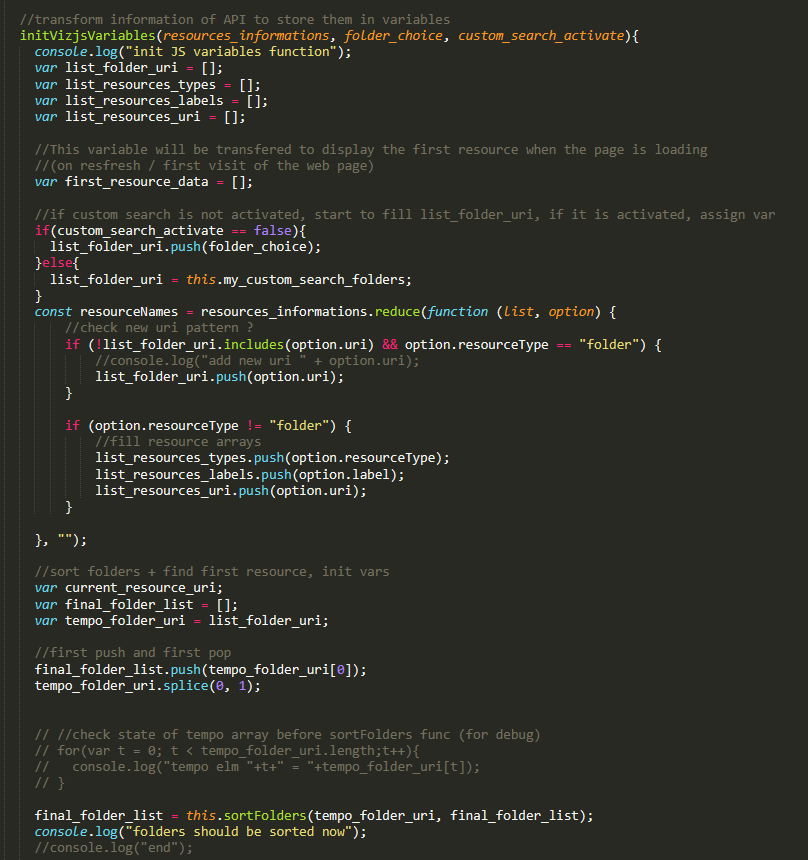
*Fonction initVizJsVariables*

**Brève description :**

Fonction qui permet d’initialiser les variables qui seront utilisées par visualizeJS.

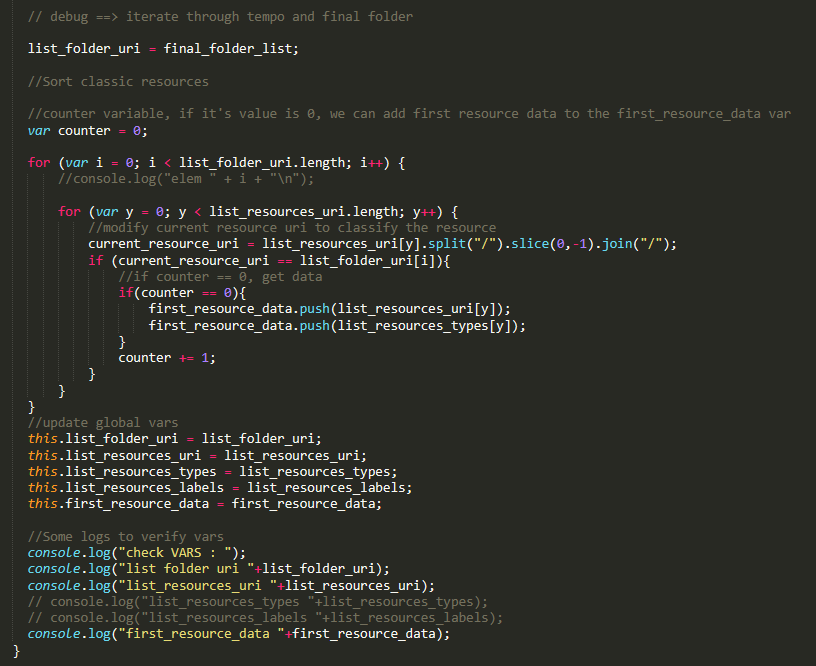
**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Récupération brute des noms des ressources et des dossiers |
| 2 | Tri des dossiers avec la fonction « sortFolders » |
| 3 | Récupération de la première ressource à afficher |
| 4 | Assignation des variables et logs de vérification |



**2**

**1**

****

**4**

**3**

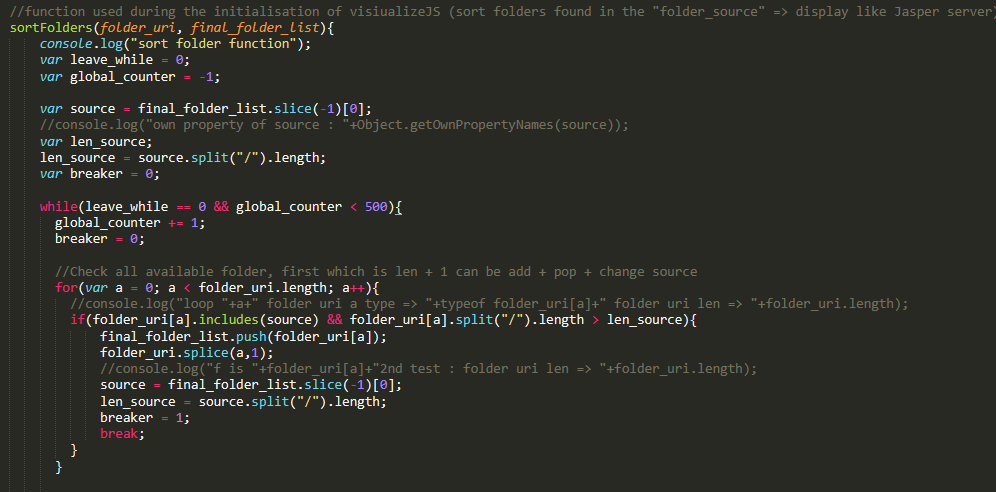
*Fonction sortFolders*

**Brève description :**

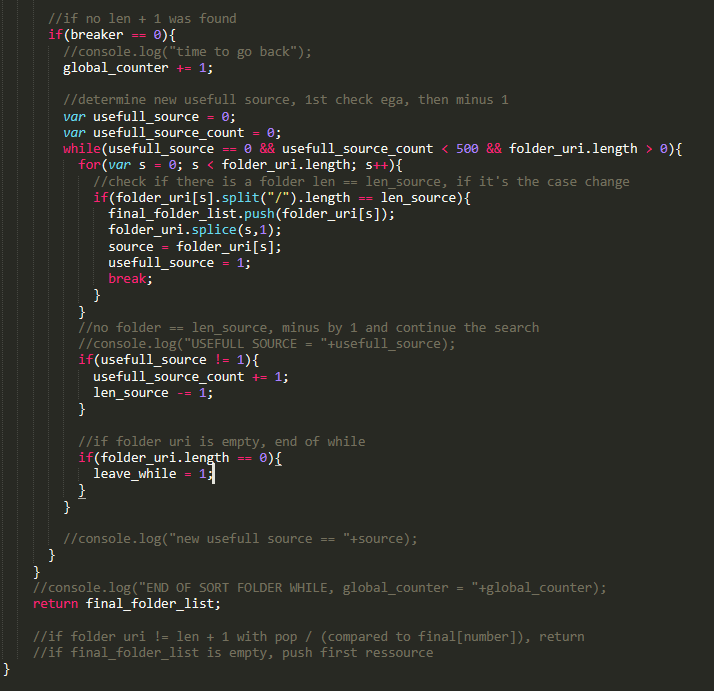
Fonction pour trier les dossiers qui s’affichent dans l’élément « select » listant les ressources. L’objectif étant d’obtenir un affichage similaire à celui du serveur Jasper en rajoutant au fur et à mesure les dossiers dans un tableau nommé « folder\_uri ».

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Si un dossier est un sous\_dossier du « dossier source » (variable locale), on le rajoute.  Si ce n’est pas le cas, « breaker » vaudra 0 et un traitement va être réalisé pour désigner un nouveau dossier source. |
| 2 | Quand aucun sous dossier n’est trouvé, on va déterminer un nouveau « dossier source » parmi les ressources restantes. |
| 3 | Si un dossier non récupéré se trouve au même niveau dans l’arborescence, il devient la nouvelle source. |
| 4 | Si ce n’est pas le cas, on remonte d’un niveau dans l’arborescence et on recommence. |



**1**



**2**

**4**

**3**

### **Fonctions qui utilisent visualizeJS pour afficher / interagir avec des ressources Jaspersoft**

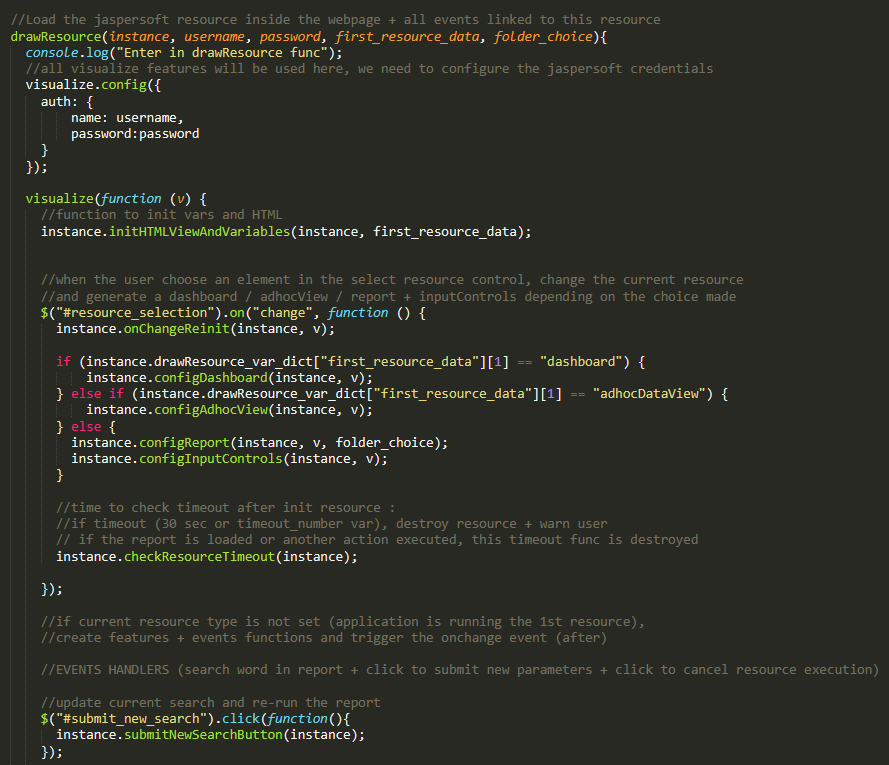
*Fonction drawResource (principale)*

**Brève description :**

La fonction **drawResource** contient l’ensemble du code et des fonctions qui utilisent visualizeJS afin de charger les ressources du serveur Jasper dans notre page HTML

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Connexion à visualizeJS et initialisation des variables ainsi que de la vue HTML |
| 2 | A chaque fois qu’une ressource est sélectionnée (dans l’élément « select » listant les ressources accessibles), réinitialiser les variables et configurer la ressource selon son type. (cet événement onchange est également déclenché lors de la génération de la première ressource) |
| 3 | Si la ressource générée met trop de temps à s’exécuter, annuler son exécution |
| 4 | Ensemble des événements « onclick » à gérer (recherche, pagination, modification de paramètres, …) |

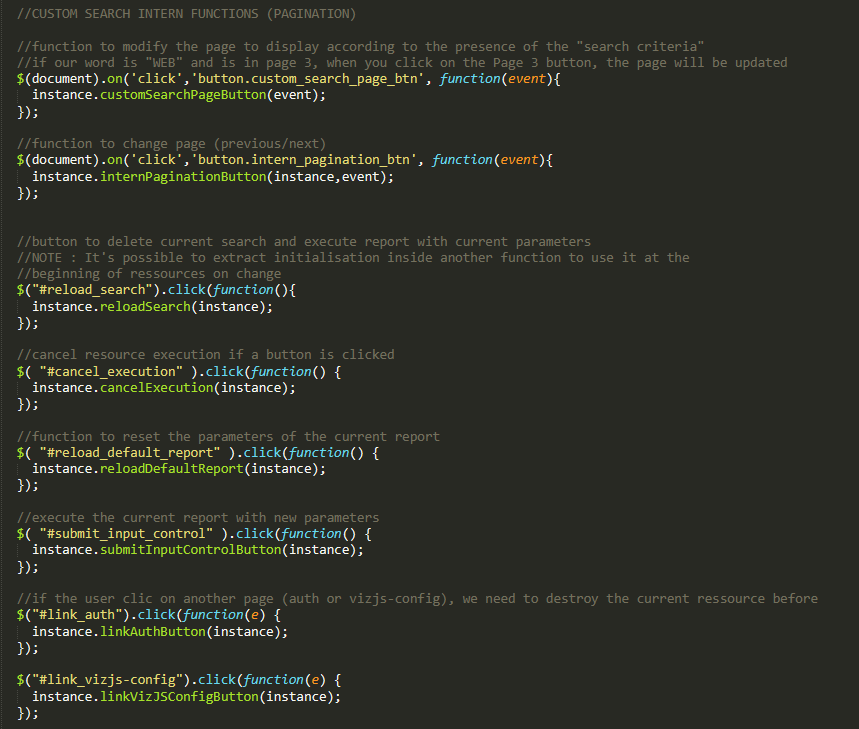


**2**

**4**

**3**

**1**



**4**



**2**

**4**

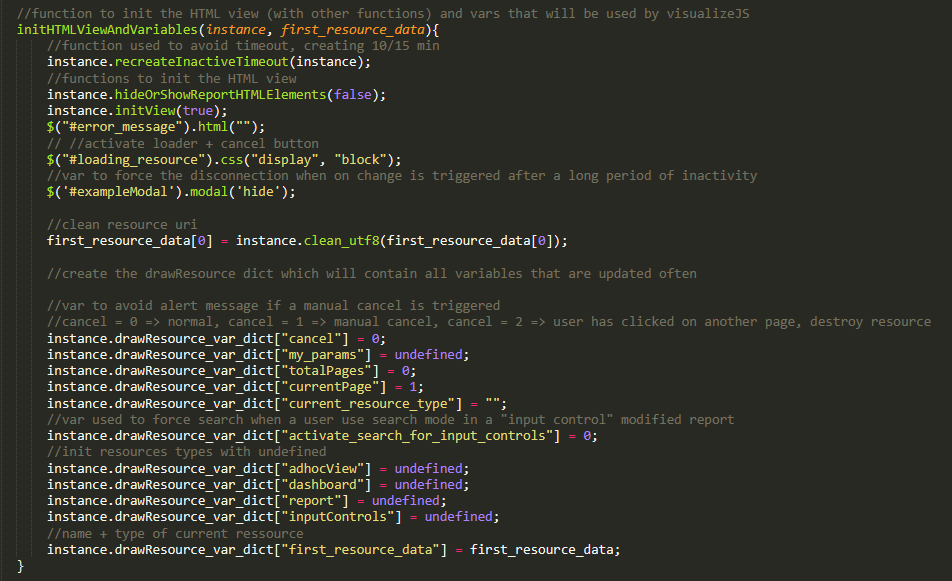
*Fonction initHTMLViewAndVariables*

**Brève description :**

Fonction appelée au début de la fonction « drawResource ». Son objectif est de modifier la vue HTML pour préparer l’affichage des ressources visualizeJS et d’instancier les variables qui seront utilisées pour manipuler ces ressources (surtout en peuplant un dictionnaire de variables).

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Appel de fonctions permettant d’initialiser la vue HTML afin de la préparer à l’affichage d’une ressource visualizeJS |
| 2 | Initialisation des variables qui seront utilisées pour bénéficier des fonctionnalités visualizeJS (surtout peuplement d’un dictionnaire de variables accessible facilement par n’importe quelle fonction puisque qu’étant un attribut du component) |



**2**

**1**

*Fonction onChangeReinit*

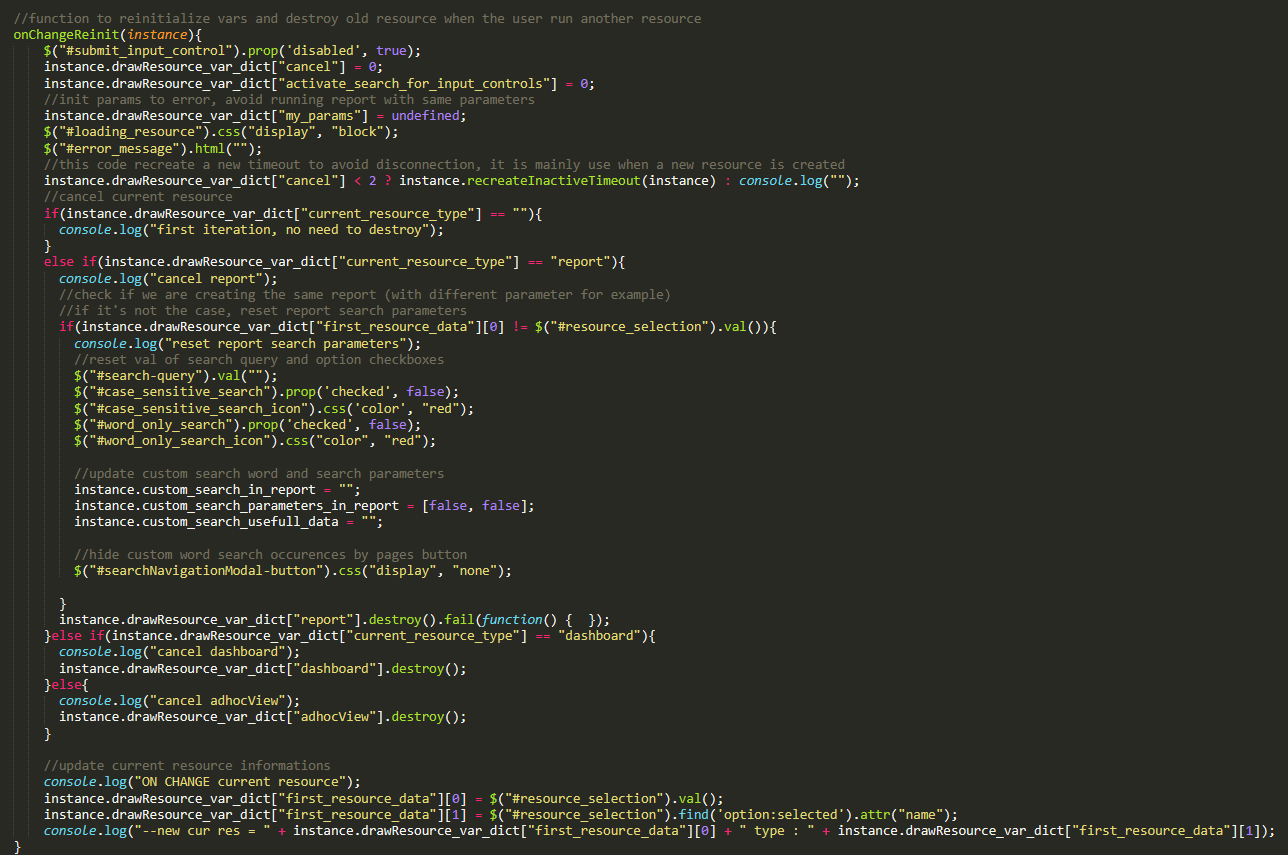
**Brève description :**

Cette fonction contient l’ensemble des traitements qui doivent être réalisés à chaque fois que l’utilisateur souhaite afficher une nouvelle ressource.

Elle va simplement redonner à certaines variables leurs valeurs originelles, réinitialiser des éléments HTML et détruire la précédente ressource générée par visualizeJS.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Réinitialisation de la vue HTML et de variables |
| 1.1 | Réinitialisation de la fonctionnalité « recherche dans un rapport » si un nouveau rapport a été demandé par l’utilisateur |
| 2 | Destruction de la précédente ressource visualizeJS |



**2**

**1.1**

**1**

*Fonction configDashboard*

**Brève description :**

Cette fonction permet de créer une ressource visualizeJS de type « tableau de bord » et de l’afficher dans notre page HTML via l’attribut « container ».

***Note***: Pour les ressources visualizeJS, il est possible d’effectuer un traitement différent selon la réussite ou non de leur chargement avec les callbacks « success » et « error ».

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Configuration du tableau de bord (ressource à afficher, conteneur HTML dans lequel afficher la ressource, …) |
| 2 | Récupération des événements « success » et « error » pour effectuer des traitements spécifiques. Ici on affiche seulement un message et on réinitialise les fonctions « timeout » si l’utilisateur ne quitte pas la page (ressource met trop de temps à s’afficher / utilisateur inactif ?), ce qui permet d’éviter une déconnexion de l’utilisateur tant qu’il génère de nouvelles ressources visualizeJS.  **Note** : On pourrait probablement uniquement écrire « recreateInactiveTimeout » seul sans que ça pose de problème |



**2**

**1**

*Fonction configAdhocView*

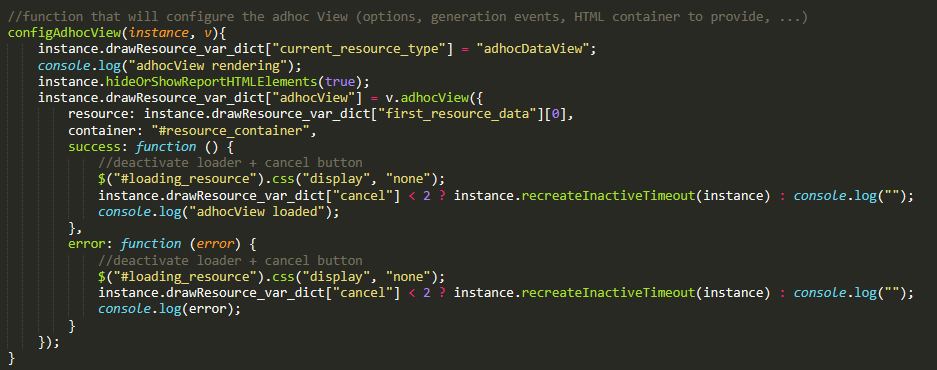
**Brève description :**

Cette fonction permet de créer une ressource visualizeJS de type « affichage à la demande » / « vue AdHoc » et de l’afficher dans notre page HTML via l’attribut « container ».

***Note***: Pour les ressources visualizeJS, il est possible d’effectuer un traitement différent selon la réussite ou non de leur chargement avec les callbacks « success » et « error ».

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Configuration de la vue AdHoc (ressource à afficher, conteneur HTML dans lequel afficher la ressource, …) |
| 2 | Récupération des événements « success » et « error » pour effectuer des traitements spécifiques. Ici on affiche seulement un message et on réinitialise les fonctions « timeout » (évite déconnexion utilisateur pour cause d’inactivité, voir la fonction « configDashboard » pour plus de détails) |



**2**

**1**

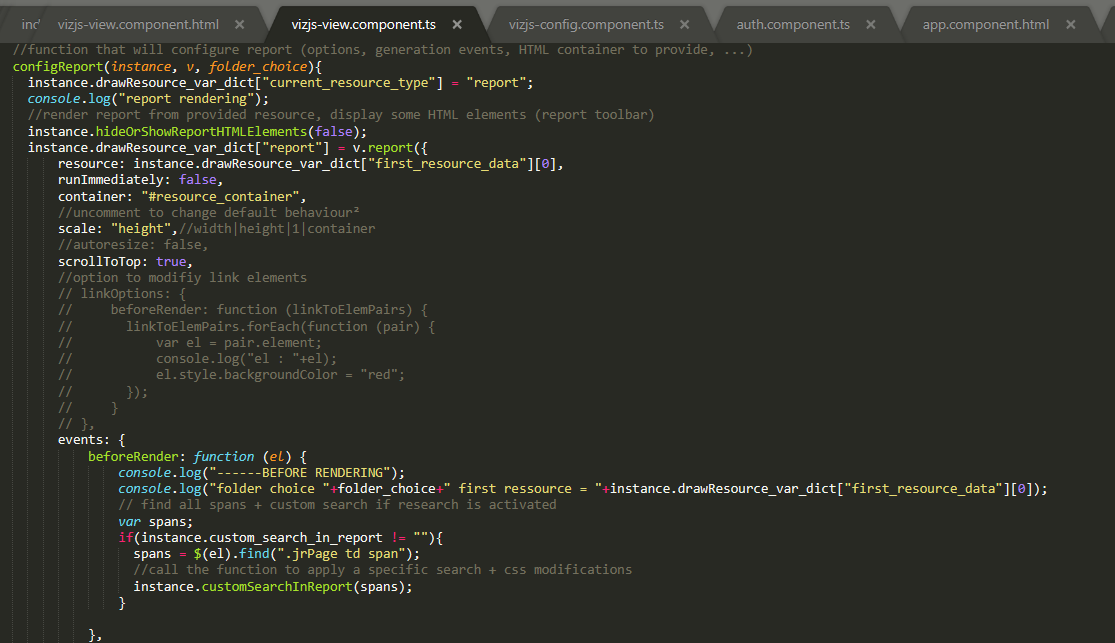
*Fonction configReport*

**Brève description :**

Cette fonction permet de créer une ressource visualizeJS de type « rapport » et de l’afficher dans notre page HTML via l’attribut « container ». Pour les rapports, plus d’événements sont gérés pour faciliter l’exécution de code lors de divers événements qui ont lieu pendant l’exécution (beforeRender, reportCompleted, changeTotalPages)

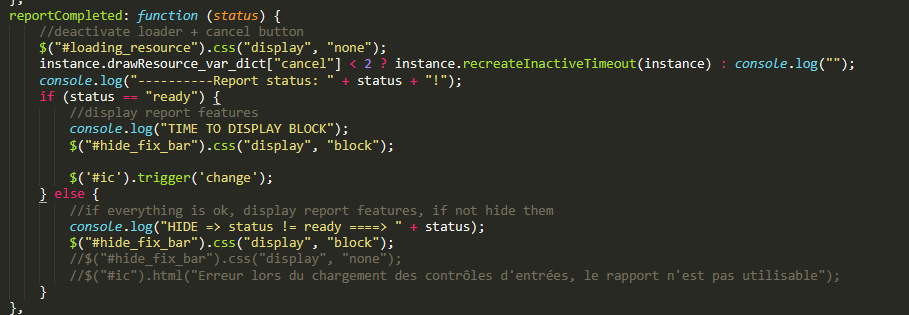
**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Configuration du rapport (ressource à afficher, conteneur HTML qui va contenir cette dernière, mode d’affichage du rapport (s’adapte au conteneur, la hauteur, la largeur, …)  Le rapport ne s’exécutera pas automatiquement (runImmediately = false), cette exécution aura lieu lorsque les paramètres (inputControls) auront été chargés et validés |
| 2 | Avant l’affichage du rapport, quelques logs de debug sont effectués et « la recherche d’un mot » dans un rapport a lieu si elle a été activée par l’utilisateur |
| 3 | Afficher la barre de fonctionnalités que le rapport est chargé (peut être obsolète si l’événement « success » gère déjà cet aspect, changera potentiellement dans les prochaines versions), la mise à jour des paramètres est également déclenchée pour que le modal « Modifier les paramètres » de la page « vizjs-view » affiche les bonnes valeurs |
| 4 | Récupère le nombre total de pages pour l’afficher côté HTML, s’il y a 0 pages, l’utilisateur est prévenu (un message lui demande de changer les paramètres) |
| 5 | Evénement success et error |
| 5.1 | Success : quand le changement de rapport a fonctionné, on affiche la barre de fonctionnalités et on autorise l’export |
| 5.2 | Error 1 : Quand il a échoué, on charge les paramètres et on demande à l’utilisateur de les changer.  S’il est impossible de charger ces paramètres, un message d’erreur s’affiche |
| 5.3 | Error 2 : Si la ressource n’a pas été stoppée manuellement par l’utilisateur (cancel) et qu’il y a malgré tout eût des problèmes pour afficher le rapport, on affiche un message d’erreur (il serait d’ailleurs possibles de générer des messages différents selon les problèmes rencontrés) |



**2**

**1**



**3**



**5.3**

**5.2**

**5.1**

**4**

*Fonction configInputControls*

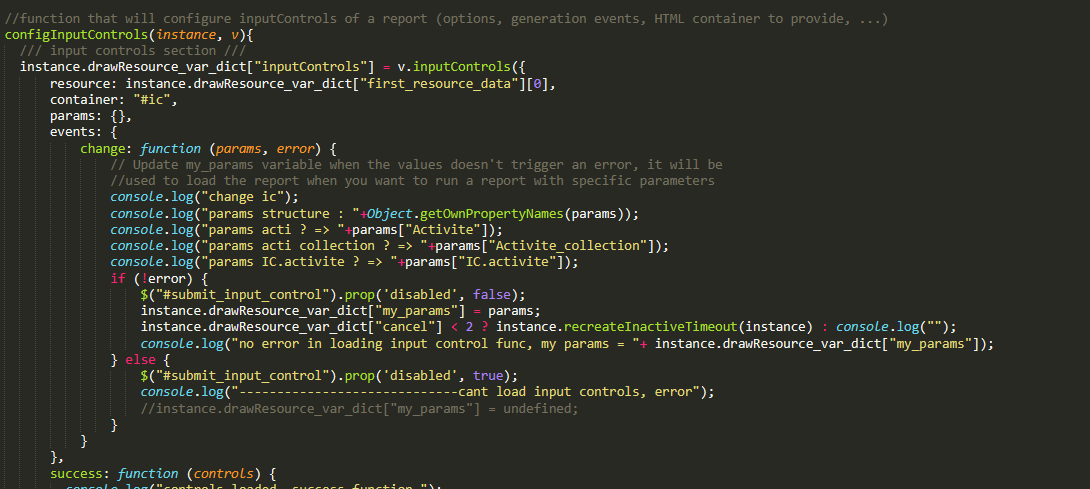
**Brève description :**

Cette fonction permet de charger les paramètres d’un rapport (alias « inputControls ») et de les afficher dans un conteneur HTML. Si l’utilisateur modifie ces paramètres et qu’il n’y a pas d’erreur, ils seront mis à jour dans la variable « my\_params ».

(Un bouton utilisant la fonction « submitInputControlButton »permettra ensuite d’exécuter le rapport avec les nouveaux paramètres)

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Configuration des « inputControls » (où les chercher ?, où les afficher, …) |
| 2 | Modifier la variable « my\_params » si les paramètres entrés sont valides, un clic sur un bouton permettra ensuite d’exécuter le rapport avec les nouveaux paramètres |
| 3 | Initialisation des paramètres (paramètres par défaut donnés par le rapport / paramètres personnalisés par l’utilisateur dans le menu vizjs-config) |
| 4 | Exécution du rapport en utilisant les paramètres précédemment définis, si l’exécution échoue, un message d’erreur est affiché |



**2**

**1**



**3**



**4**

*Fonction checkResourceTimeout*

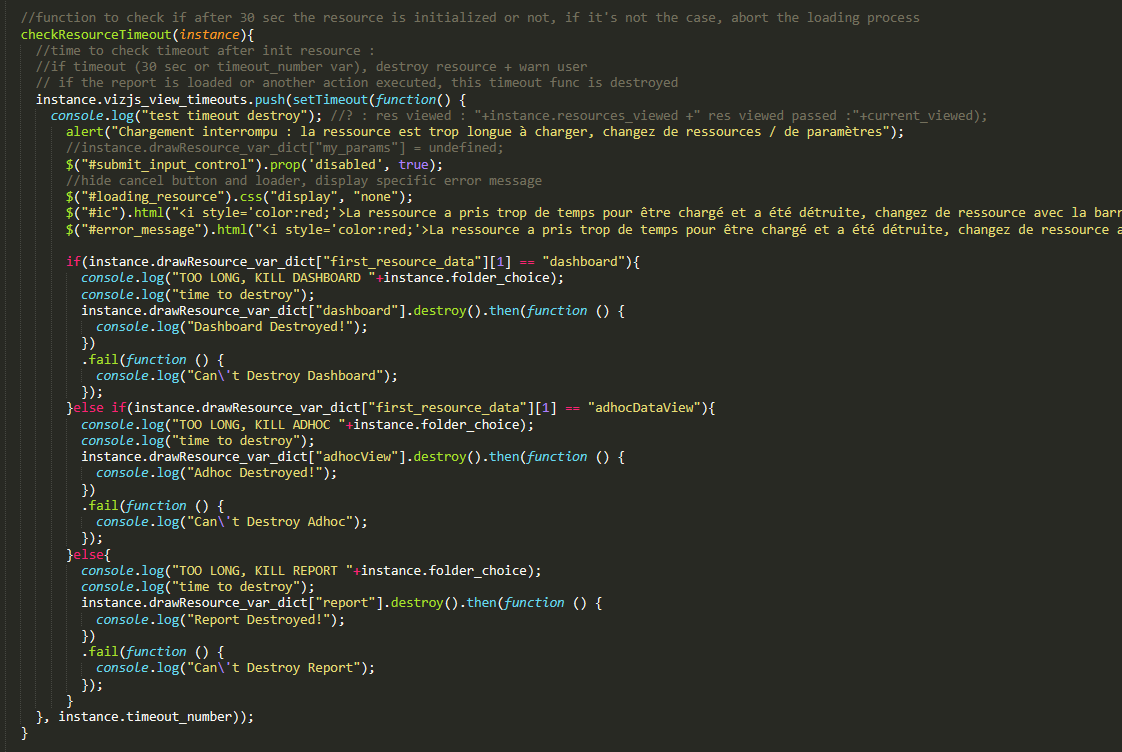
**Brève description :**

Cette fonction est déclenchée X secondes (paramètre modifiable en haut du fichier TS) après avoir chargé la ressource visualizeJS, si cette ressource n’a pas pu être exécutée correctement, son exécution est annulée (la ressource est détruite).

Si au contraire tout se passe bien, cette fonction ne sera pas exécutée si l’utilisateur effectue une « action » (exécuter un nouveau rapport par exemple) grâce à la fonction « recreateInactiveTimeout ».

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Afficher un message d’alerte et faire des modifications HTML si la fonction se déclenche |
| 2 | Détruire la ressource (selon son type) avec la méthode « destroy » |



**2**

**1**

*Fonction submitNewSearchButton*

**Brève description :**

Fonction appelée lorsque l’utilisateur effectue une recherche d’un mot dans un rapport en précisant certains critères (mot sensible à la casse ? recherche uniquement de mots entiers dans le texte ?).

Cette fonction mettra à jour des variables et exécutera un nouveau le rapport mais en effectuant un traitement dans l’événement « BeforeRender » qui surlignera les occurrences de la recherche.

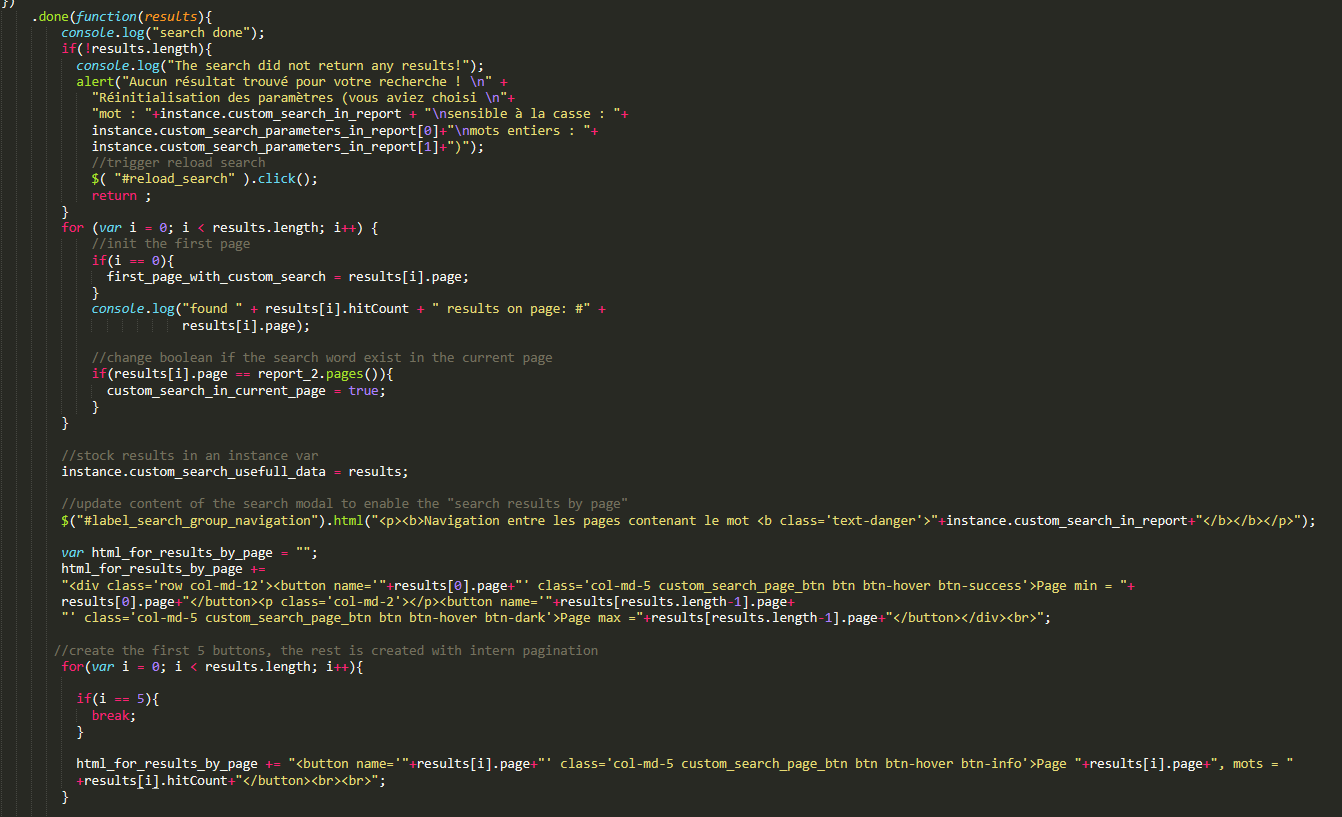
**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Récupération des entrées utilisateurs (paramètres + mot recherché).  Si le mot est « vide » (« »), le rapport est réactualisé sans effectuer de modifications.  Des modifications CSS sont faîtes pour activer / désactiver les critères de recherches. |
| 2 | Effectuer une opération « search » sur notre rapport.  Cette opération va retourner le nombre d’occurrences du mot recherché dans chaque page du rapport. |
| 2.1 | Paramètres de la recherche (sensible à la casse / mots entiers) |
| 2.2 | Recherche qui ne produit pas d’erreur :  - Si aucun résultat n’a été trouvé, afficher un message spécifique et réinitialiser le rapport  - Si on a trouvé un résultat,  \* stocker l’ensemble des informations (objets du type {nb\_occurences : 3, id\_page : 5}) dans une variable  \* créer des éléments HTML permettant de se déplacer entre les différentes pages qui contiennent le mot recherché  \* réactualiser le rapport sur la page courante si le mot s’y trouve, sinon sur la première page qui contient ce mot |
| 2.3 | Recherche avec erreur : Afficher une erreur à l’utilisateur et ne pas exécuter à nouveau le rapport |

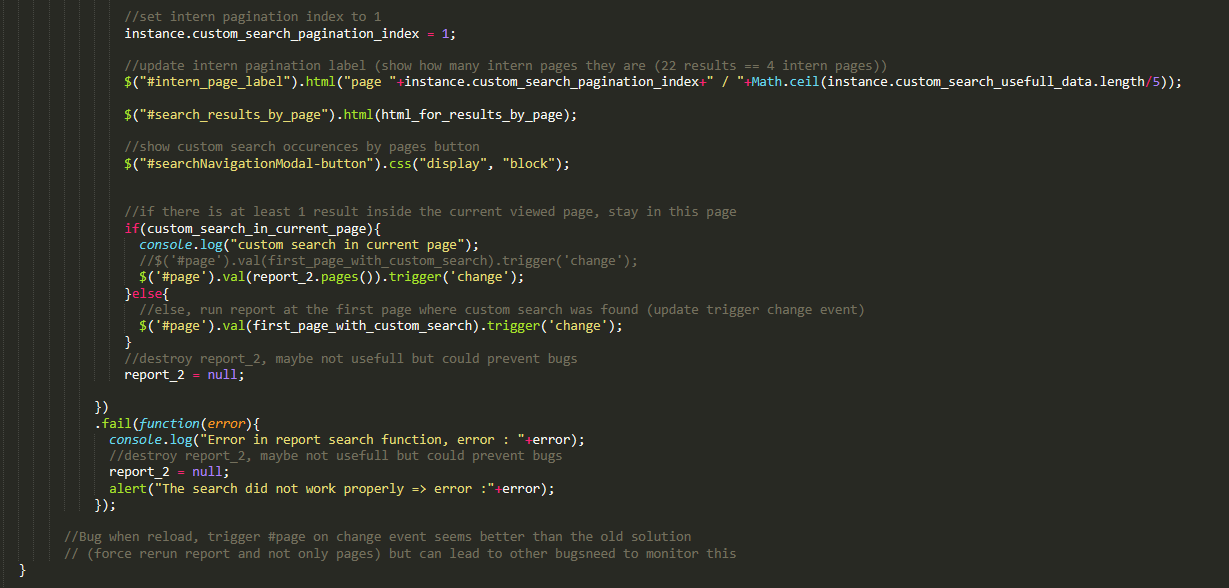


**2.1**

**1**



**2.2**



**2.2**

**2.3**

*Fonction customSearchPageButton*

**Brève description :**

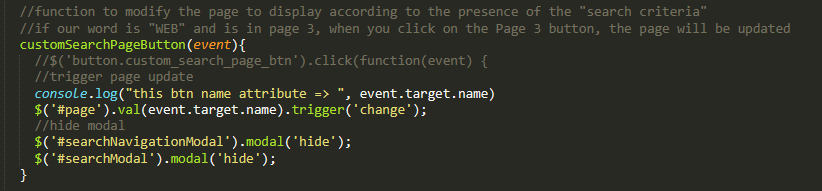
Cette fonction s’exécute quand l’utilisateur clique sur un bouton qui indique une page + un nombre d’occurrences du mot recherché

(dans le menu qui affiche l’ensemble des pages sur lesquelles on peut trouver notre mot).

Quand l’utilisateur clique sur un bouton du type {Page : 3, occurrences : 5}, il sera redirigé vers la page 3 et son mot recherché sera surligné (il y en aura 5).

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Modifier la page du rapport courant et fermer le menu (modal) permettant d’itérer entre les différentes pages où notre mot recherché est présent. |



**1**

*Fonction internPaginationButton*

**Brève description :**

Cette fonction permet de se déplacer entre les différentes pages qui contiennent notre mot recherché. Ces pages contiennent 5 boutons du type {Page : X, occurrences : Y} qui lors d’un clic redirigent l’utilisateur vers la page concernée en affichant le mot recherché (surligné dans le rapport).

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Modification de l’index de la page à atteindre + des nouveaux boutons selon l’élément HTML sur lequel on a cliqué (next / previous)  Un message d’erreur est déclenché si le changement de page est impossible (page 0 / page max + 1) |
| 2 | Mise à jour de la vue HTML pour afficher les nouveaux boutons |



**2**

**1**

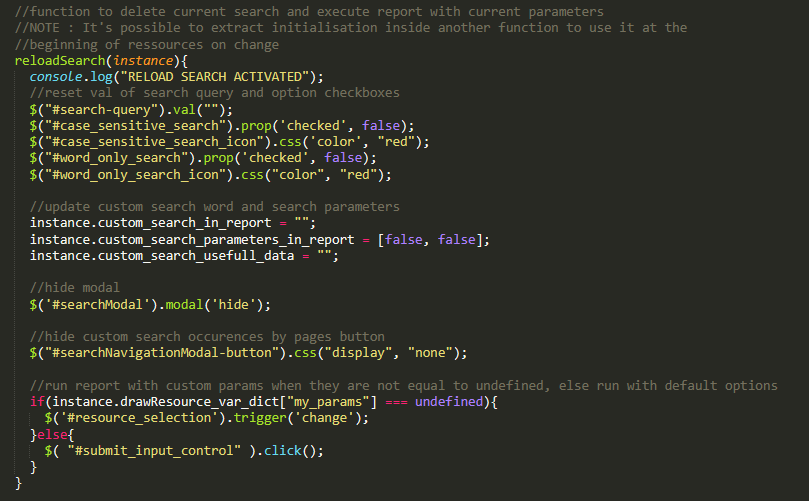
*Fonction reloadSearch*

**Brève description :**

Cette fonction réinitialise toutes les variables liées à la recherche de mots à l’intérieur d’un rapport et exécute à nouveau le rapport courant avec les paramètres actuellement définis.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Réinitialisation des variables et de la vue HTML concernant la recherche |
| 2 | Exécution du rapport avec les paramètres actuellement définis |



**2**

**1**

*Fonction cancelExecution*

**Brève description :**

Fonction pour arrêter l’exécution d’une ressource visualizeJS lors du clic sur un bouton (pour les « AdHocView » la ressource doit être « destroy », pour les autres, elle peut être « cancel »)

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Modification d’une variable pour indiquer que l’exécution de la ressource actuelle a été annulée. |
| 2 | Destruction ou arrêt de l’exécution de la ressource courante. |



**2**

**1**

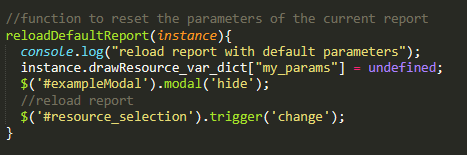
*Fonction reloadDefaultReport*

**Brève description :**

Cette fonction est utilisée pour exécuter le rapport courant à nouveau mais en n’utilisant aucun paramétrage.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Réinitialisation des paramètres et exécution du rapport courant |



**1**

*Fonction submitInputControlButton*

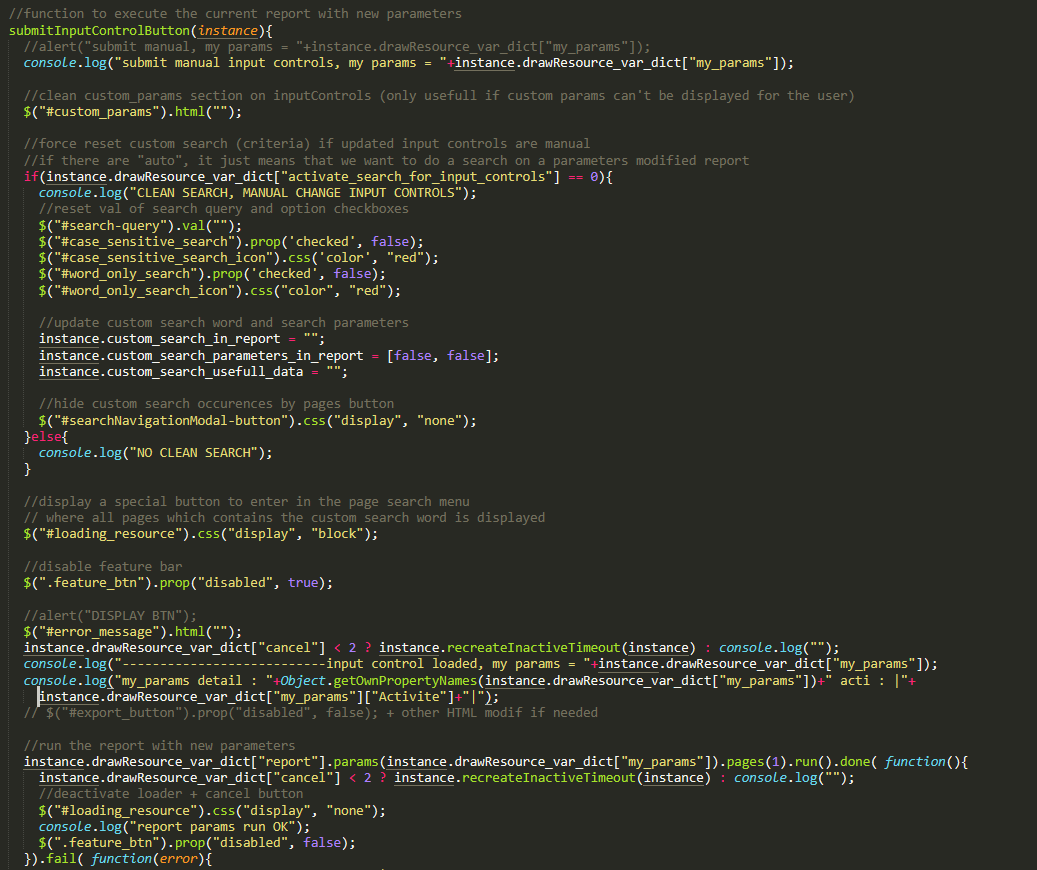
**Brève description :**

Cette fonction permet d’exécuter le rapport courant avec de nouveaux paramètres (/ inputControls).

Avant d’effectuer ce traitement, les paramètres de recherche d’un mot dans un rapport sont réinitialisés et une fonction timeout est créée pour déclencher l’annulation de l’exécution du rapport si un temps d’attente X (fixé en haut du fichier TS) s’est écoulé.

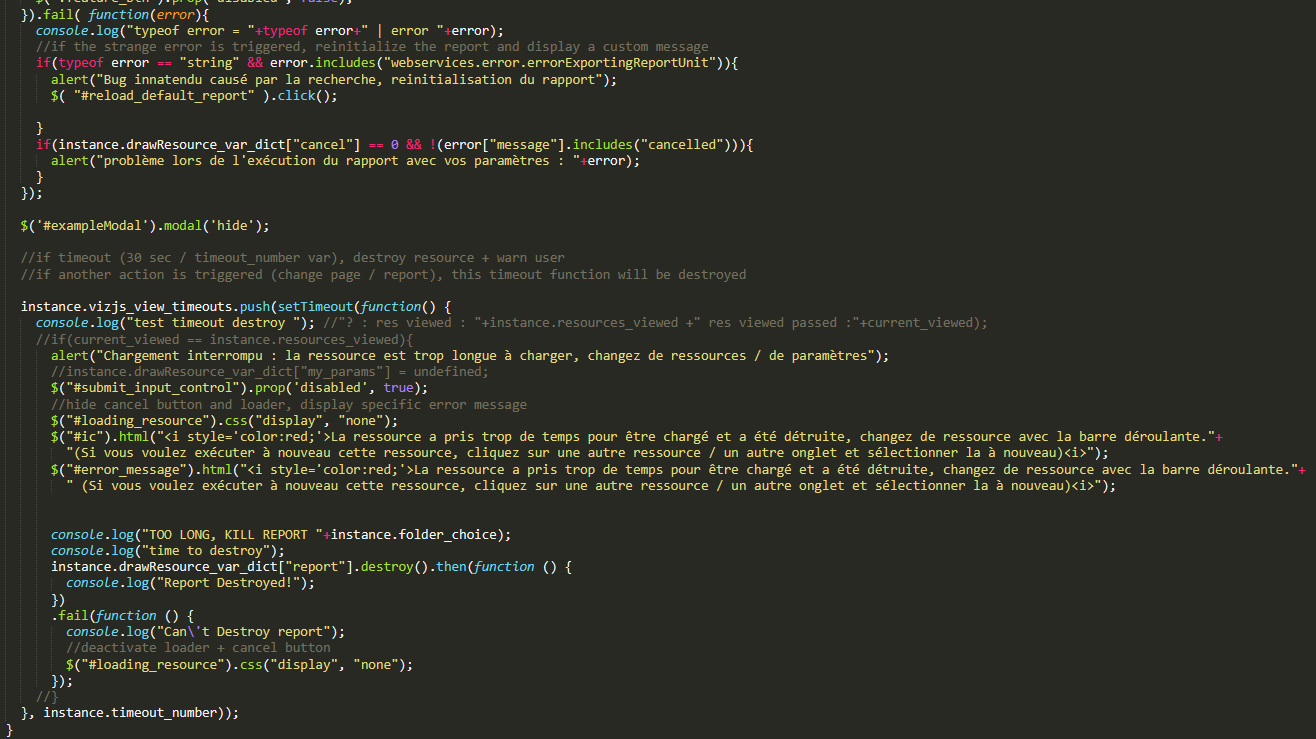
**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Réinitialisation de la recherche et quelques modifications dans la vue HTML pour éviter les manipulations hasardeuses |
| 2 | Exécuter à nouveau le rapport mais avec les paramètres entrés par l’utilisateur (affichage d’un message d’erreur si un problème est rencontré) |
| 3 | Création d’une nouvelle fonction timeout qui s’exécutera X secondes après l’exécution du rapport (si le rapport n’est toujours pas chargé, son exécution sera annulée) |



**2**

**1**



**3**

**2**

*Fonction linkAuthButton*

**Brève description :**

Cette fonction se déclenche quand l’utilisateur quitte la page où se trouve la ressource visualizeJS pour se rendre sur la page d’authentification (en cliquant sur un bouton dans la barre de navigation). Son but est de supprimer correctement la ressource visualizeJS pour ne pas qu’elle reste « active » inutilement.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Destruction de la ressource visualizeJS selon son type |



**1**

*Fonction linkVizJSConfigButton*

**Brève description :**

Cette fonction se déclenche quand l’utilisateur quitte la page où se trouve la ressource visualizeJS pour se rendre sur la page « vizjs-config » (en cliquant sur un bouton dans la barre de navigation) qui permet de modifier les options de recherche de ressources (notamment le dossier source).

Son but est de supprimer correctement la ressource visualizeJS pour ne pas qu’elle reste « active » inutilement.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Destruction de la ressource visualizeJS selon son type |



**1**

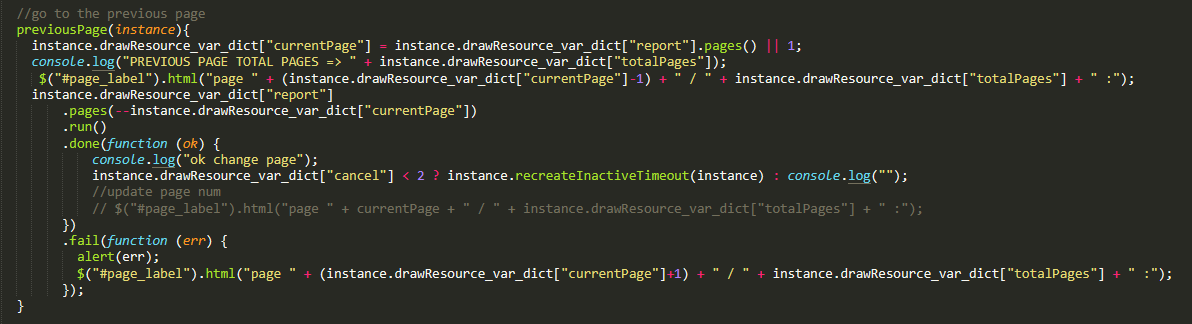
*Fonction previousPage*

**Brève description :**

Fonction qui permet d’afficher la page précédente du rapport courant quand c’est possible (page != 0) et de modifier la vue HTML pour afficher la nouvelle page courante. Pour ce faire, la fonction « .pages » des rapports visualizeJS est utilisée.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Utilisation de la fonction « pages » pour exécuter le rapport courant sur la page précédente. |



**1**

*Fonction nextPage*

**Brève description :**

Fonction qui permet d’afficher la page suivante du rapport courant quand c’est possible (nouvelle page <= max) et de modifier la vue HTML pour afficher la nouvelle page courante. Pour ce faire, la fonction « .pages » des rapports visualizeJS est utilisée.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Utilisation de la fonction « pages » pour exécuter le rapport courant sur la page suivante. |



**1**

*Fonction updatePage*

**Brève description :**

Fonction qui permet de modifier la page à afficher du rapport courant quand c’est possible (page est bien une valeur et pas une chaine de caractères + la page existe bel et bien dans le rapport). Pour ce faire, la fonction « .pages » des rapports visualizeJS est utilisée.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Vérifier que la valeur entrée est une valeur, sinon avertir l’utilisateur |
| 2 | Utilisation de la fonction « pages » pour exécuter le rapport courant sur la page donnée par l’utilisateur. |



**1**

**2**

*Fonction zoomPlus*

**Brève description :**

Cette fonction permet de modifier la taille du conteneur de la ressource visualizeJS pour qu’elle apparaisse en plus grand (par défaut la taille vaut « hauteur de l’écran \* 0.8 »). Ici le conteneur s’agrandit à chaque clic de 10% de sa taille originelle (mais dans un intervalle de 25 à 300% de la taille originelle).

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Récupération de la nouvelle taille hypothétique et vérification de l’intervalle. |
| 2 | Si tout va bien, modifier l’affichage HTML et exécuter à nouveau le rapport avec la fonction « pages ». |



**2**

**1**

*Fonction zoomMinus*

**Brève description :**

Cette fonction permet de modifier la taille du conteneur de la ressource visualizeJS pour qu’elle apparaisse en plus petit (par défaut la taille vaut « hauteur de l’écran \* 0.8 »). Ici le conteneur se rétrécit à chaque clic de 10% de sa taille originelle (mais dans un intervalle de 25 à 300% de la taille originelle).

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Récupération de la nouvelle taille hypothétique et vérification de l’intervalle. |
| 2 | Si tout va bien, modifier l’affichage HTML et exécuter à nouveau le rapport avec la fonction « pages ». |



**2**

**1**

*Fonction customZoom*

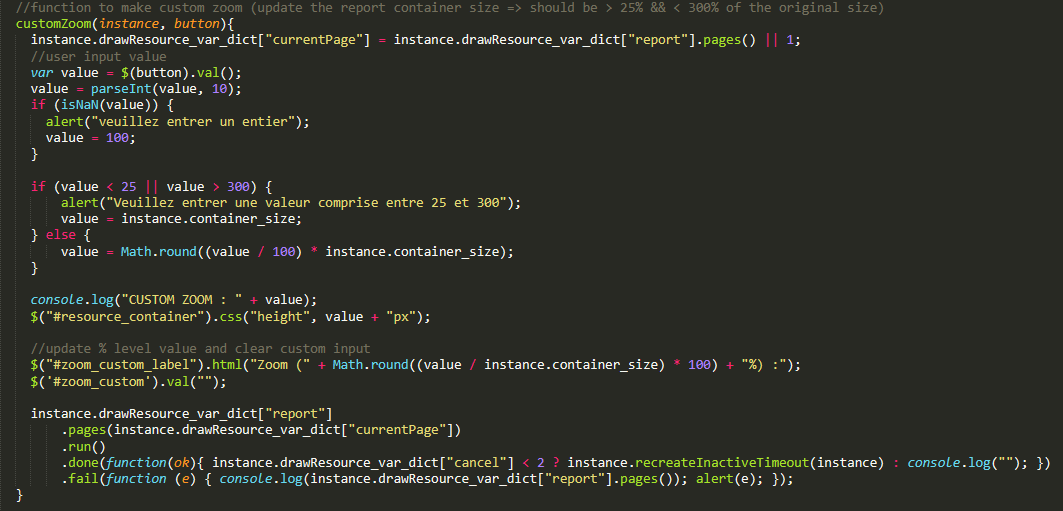
**Brève description :**

Cette fonction permet de modifier la taille du conteneur de la ressource visualizeJS pour qu’elle corresponde à ce qu’a entré l’utilisateur (par défaut la taille vaut « hauteur de l’écran \* 0.8 »).

Ici, l’entrée utilisateur doit être un nombre entier compris entre 25 et 300 (il y a un intervalle pour les valeurs => de 25 à 300% de la taille originelle).

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Récupération de la nouvelle taille hypothétique, vérification de la valeur entrée par l’utilisateur (est-ce un entier ?) et de l’intervalle (% entre 25 et 300). |
| 2 | Si tout va bien, modifier l’affichage HTML et exécuter à nouveau le rapport avec la fonction « pages ». |



**1**

**2**

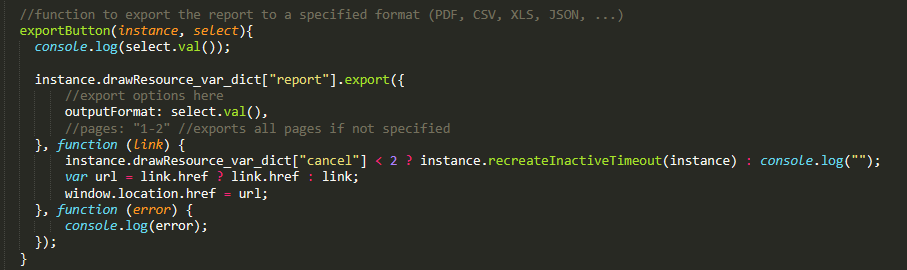
*Fonction exportButon*

**Brève description :**

Fonction qui permet d’export les rapports visualizeJS dans le format choisi (PDF, XLS, CSV, …). Il y aura tout simplement une redirection url avec « window.location.href » pour télécharger le rapport au bon format.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Utilisation de la fonction « export » du rapport, sélection du format depuis l’élément select. |
| 2 | Modification de « window.location.href » pour télécharger la ressource au format souhaité. |

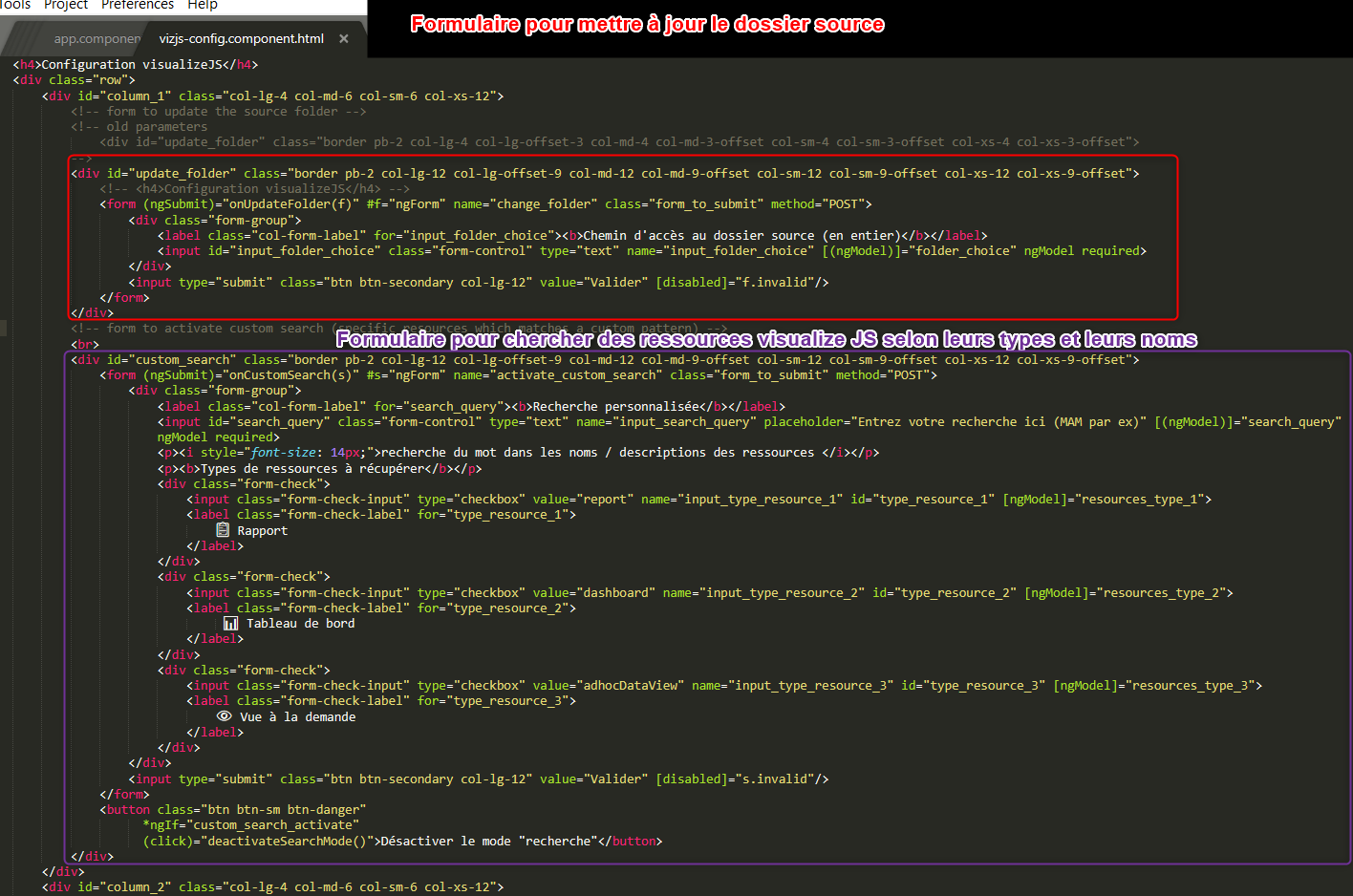


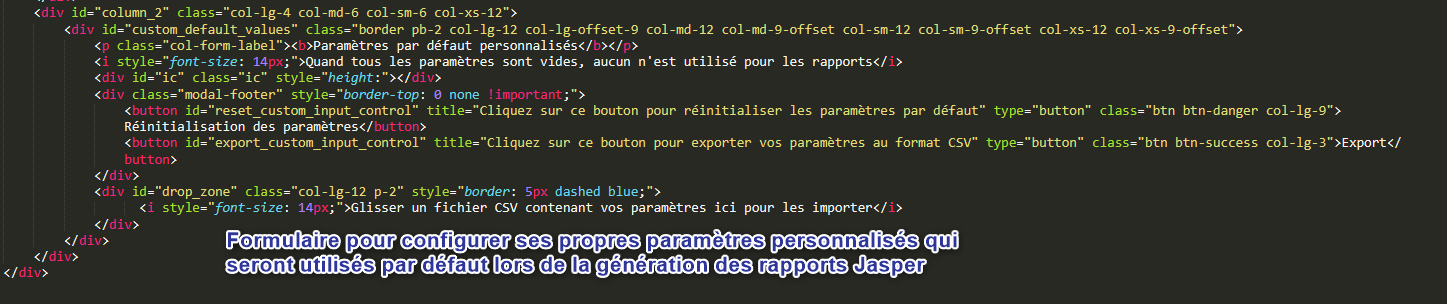
**2**

**1**

## **Component vizjs-config**

Partie HTML





Partie Typescript

### **Code lié au component (variables, constructeur, ngOnInit, ngOnDestroy)**

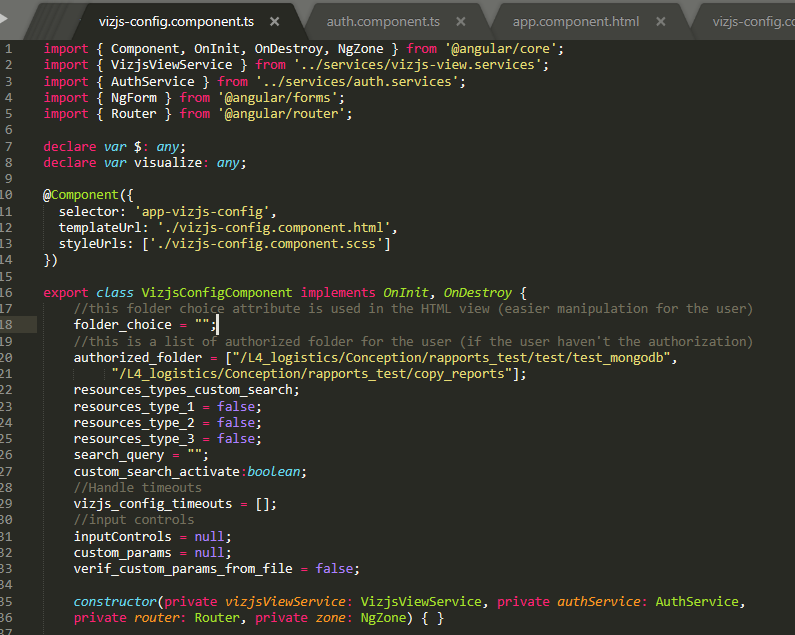
*Imports, création des variables et constructeur*

**Brève description :**

Pour que le component puisse fonctionner, il est nécessaire d’importer des librairies propres à Angular (NgForm, OnInit, NgZone, …), de définir des variables globales (modifiables facilement dans n’importe qu’elle fonction) ainsi qu’un constructeur implémentant les différents services que nous allons utiliser.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Les variables définies ici seront utilisées dans les fonctions de ce fichier TS. La plupart servent uniquement à conserver les valeurs entrées par l’utilisateur pour que sa configuration soit sauvegardée.  Par exemple, les variables « ressources\_types\_X » permettent de cocher ou non les types de ressources visualize JS qui seront affichées avec une recherche personnalisée.    Il y a également des variables utilisées uniquement pour effectuer certains traitements, comme :  - « vizjs\_config\_timeouts » qui stocke l’ensemble des fonctions timeout en cours (qui ont vocation à déconnecter l’utilisateur après un certain temps d’inactivité) dans un tableau qui est vidé à chaque changement de page  - « verif\_custom\_params\_from\_file » qui passe à « true » quand l’utilisateur importe un fichier CSV contenant des paramètres personnalisés. Quand il est actif, des tests vont avoir lieu pour s’assurer que les paramètres entrés sont conformes |



**1**

*Fonction ngOnInit (exécutée quand le component est créé)*

**Brève description :**

Ensemble de traitements à réaliser quand le component est créé. Les variables vont s’initialiser en récupérant les valeurs stockées dans le service « vizjsView », service qui contient l’ensemble des configurations effectuées par l’utilisateur.

Ensuite, les paramètres personnalisables par l’utilisateur (utilisés par défaut par les rapports Jasper) vont être chargés et des « event listeners » vont être configurés pour déclencher des fonctions quand des actions clés ont lieu (comme l’import d’un fichier CSV par exemple)

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Rafraichir la session utilisateur en renouvelant le compte à rebours avant déconnexion (15m) + Initialisation des variables en récupérant les paramètres configurés par l’utilisateur dans le service « vizjsViewService ».  *Note*: Le service « vizjsViewService » est mis à jour dès qu’un utilisateur fait une modification valide dans la page de configuration |
| 2 | Les paramètres personnalisés pour les rapports Jasper sont chargés depuis un rapport stocké sur le serveur JasperReports (il ne sert qu’à contenir l’ensemble des inputControls que l’on souhaite rendre « configurables » pour les utilisateurs), dès qu’une modification aura lieu, ils se mettront à jour |
| 3 | Des « event listeners » sont mis en place pour appeler des fonctions quand une certaine action a lieu, comme l’export des paramètres personnalisés au format CSV |



**3**

**1**

**2**

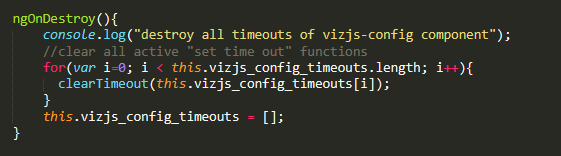
*Fonction\_ngOnDestroy*

**Brève description :**

Ensemble de traitements à réaliser quand le component est détruit. En l’occurrence, le seul traitement à réaliser ici est la suppression des de la fonction « timeout » créée (pour empêcher la déconnexion de l’utilisateur s’il est actif)

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Appel de la fonction javascript « clearTimeOut » pour détruire la fonction timeout (ici dans une boucle au cas où d’autres fonctions timeout sont créés à l’avenir) |



**1**

### **Fonctions utiles pour la configuration de visualize JS**

*Fonction onUpdateFolder*

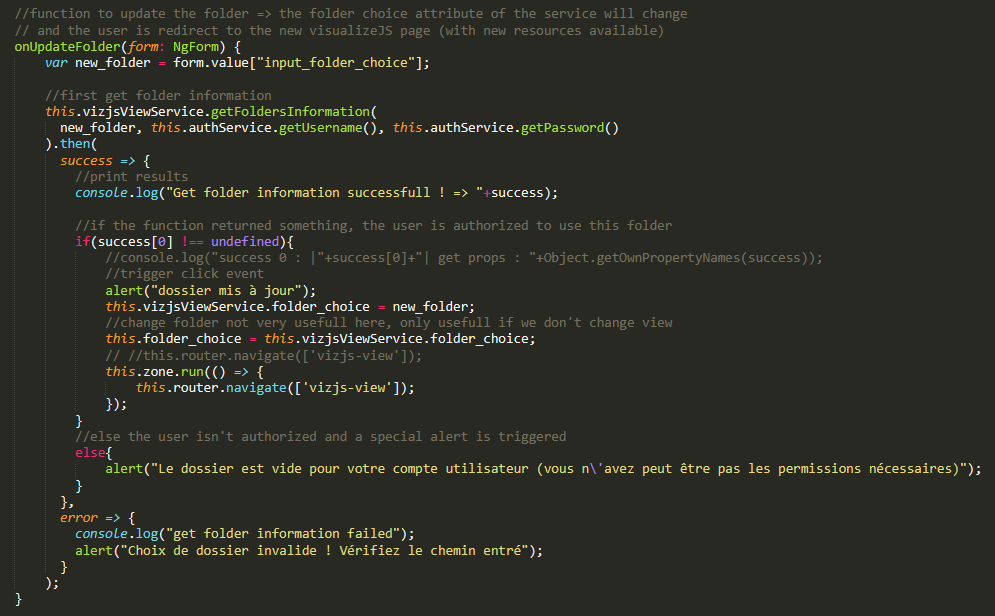
**Brève description :**

Fonction qui permet de modifier le dossier Jasper qui sera utilisé en tant que « dossier source » (toutes les ressources visualizeJS contenues à l’intérieur de ce dossier seront accessibles sur la page vizjs-view).

Pour effectuer cette modification, un appel à une fonction du service « vizjsViewService » a lieu, ce qui permet des informations sur le dossier passé en paramètre. Si aucune information n’est récupérée, le dossier est invalide (il est vide ou inaccessible pour l’utilisateur) et un message d’erreur s’affiche.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Appel de la fonction du service (utilisant visualize JS) pour récupérer les informations du dossier entré par l’utilisateur |
| 2 | Si la fonction a bien retourné des données, le dossier est valide et la mise à jour du dossier source à lieu. L’utilisateur est ensuite redirigé vers la page vizjs-view pour accéder aux nouvelles ressources Jasper |
| 3 | Si le dossier entré par l’utilisateur n’existe pas / ne contient aucune ressource / lui est inaccessible, un message d’erreur s’affiche |



**3**

**2**

**1**

*Fonction onCustomSearch*

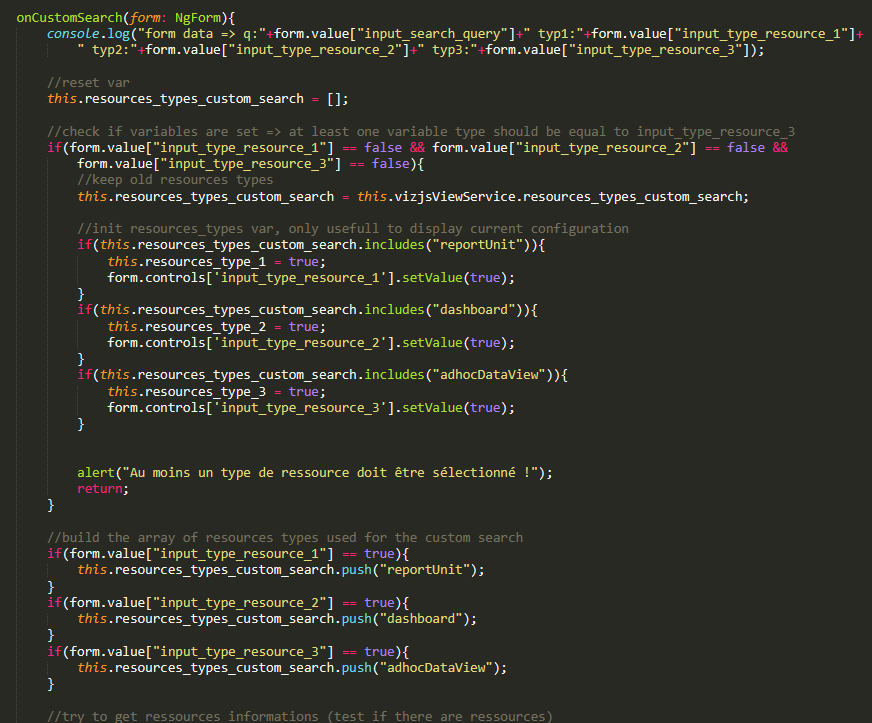
**Brève description :**

Cette fonction récupère les différents critères de recherche données par l’utilisateur, en l’occurrence, un mot qui doit être contenu dans le titre de la ressource visualize JS à chercher ainsi que les types de ressources qu’il faut récupérer (rapport, tableau de bord, vues AdHoc / Affichages à la demande)

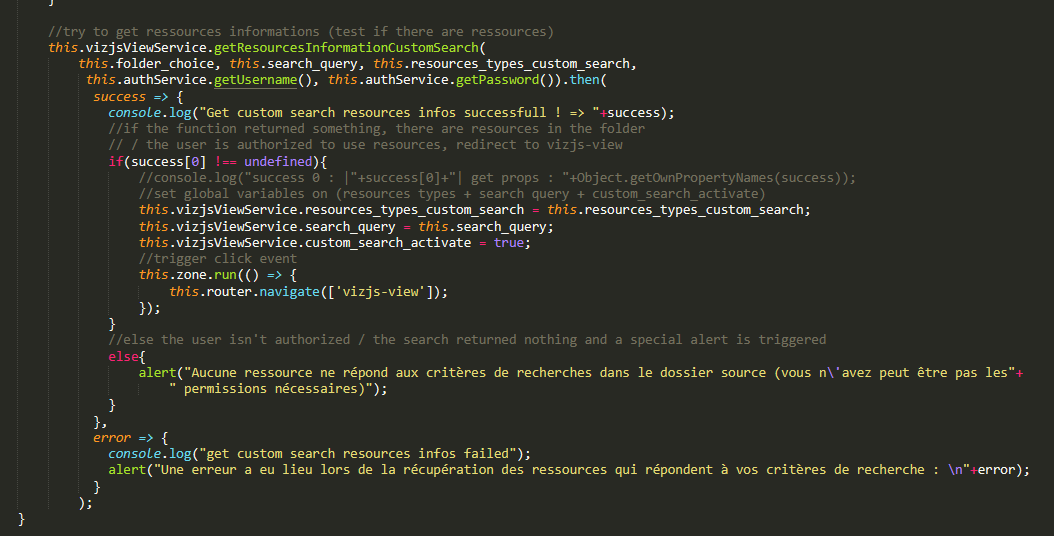
Ces critères sont ensuite vérifiés (notamment en faisant un appel visualize JS à l’aide du service vizjsViewService) et s’il n’y pas d’erreur (la recherche retourne au moins 1 ressource + l’utilisateur a le droit d’accéder à cette ressource), l’utilisateur est redirigé vers la page vizjs-view.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Vérification « manuelle » des types de ressources entrés par l’utilisateur (si aucun type de ressource sélectionné, afficher une alerte) + préparation des variables pour la recherche personnalisé |
| 2 | Appel d’une fonction du service vizjsViewService pour effectuer cette recherche personnalisée à l’intérieur de notre « dossier source » |
| 3 | Si la fonction retourne des données (utilisateur autorisé + au moins 1 ressource récupérée), le mode recherche personnalisée s’active (modifications des variables du service) et l’utilisateur est redirigé vers la page vizjs-view.  Si ce n’est pas le cas ou qu’il y a eu une erreur quelconque, une alerte s’affiche |



**1**



**2**

**3**

*Fonction deactivateSearchMode*

**Brève description :**

Réinitialisation des variables du component et du service VizjsView pour déscativer la recherche personnalisé (mot compris dans le titre d’une ressource / type de la ressource).

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Réinitialisation des variables du component et du service VizjsView |



**1**

*Fonction createJasperUI*

**Brève description :**

Cette fonction permet de récupérer un ensemble de paramètres stockés dans un rapport sur le serveur JasperReports, elle s’exécute au moment où le component (page web) est créée.

Ces paramètres ont vocation à prendre une valeur qui sera utilisée par défaut lors de l’exécution d’un rapport dans la page vizjs-view.

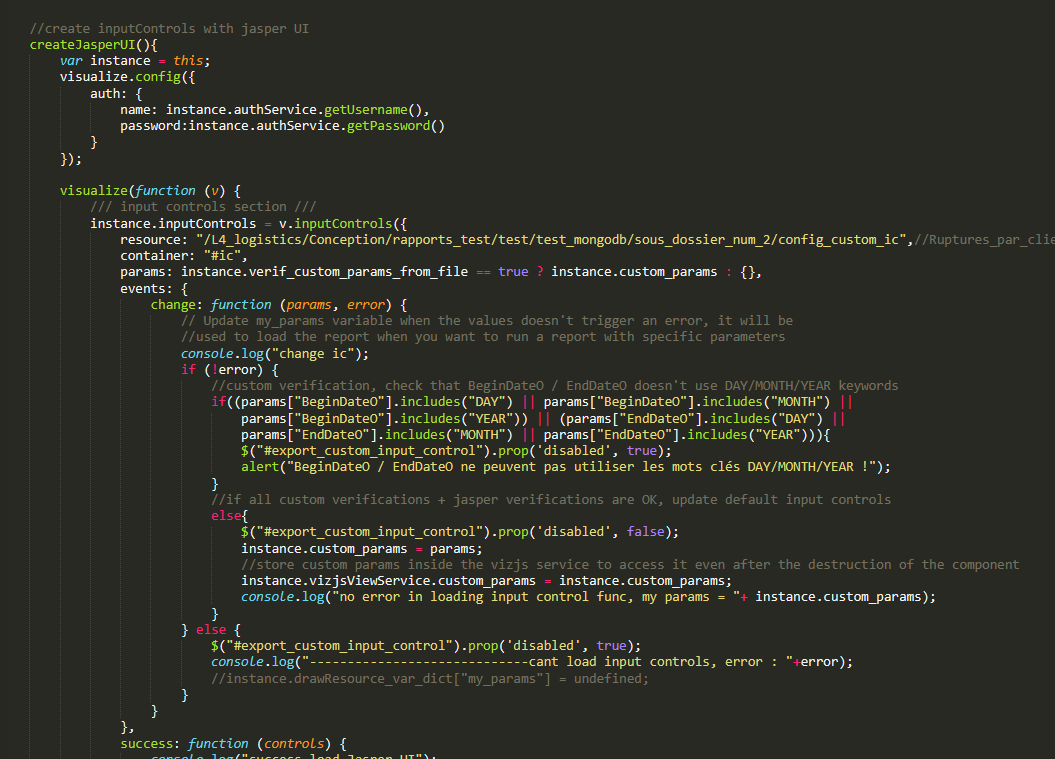
Si on sélectionne la valeur « MAM » dans le paramètre « liste\_acti », les rapports qui possèdent ce paramètre « liste\_acti » s’exécuteront directement pour l’activité MAM sans besoin de configuration manuelle des paramètres.

Ces paramètres peuvent être modifiés directement dans un formulaire ou encore importés dans un fichier CSV, ils sont validés par divers tests et sont appliqués directement sans utiliser de bouton « Valider » ou autre.

*Note* : Il est possible de réinitialiser (/désactiver) et d’exporter ces paramètres

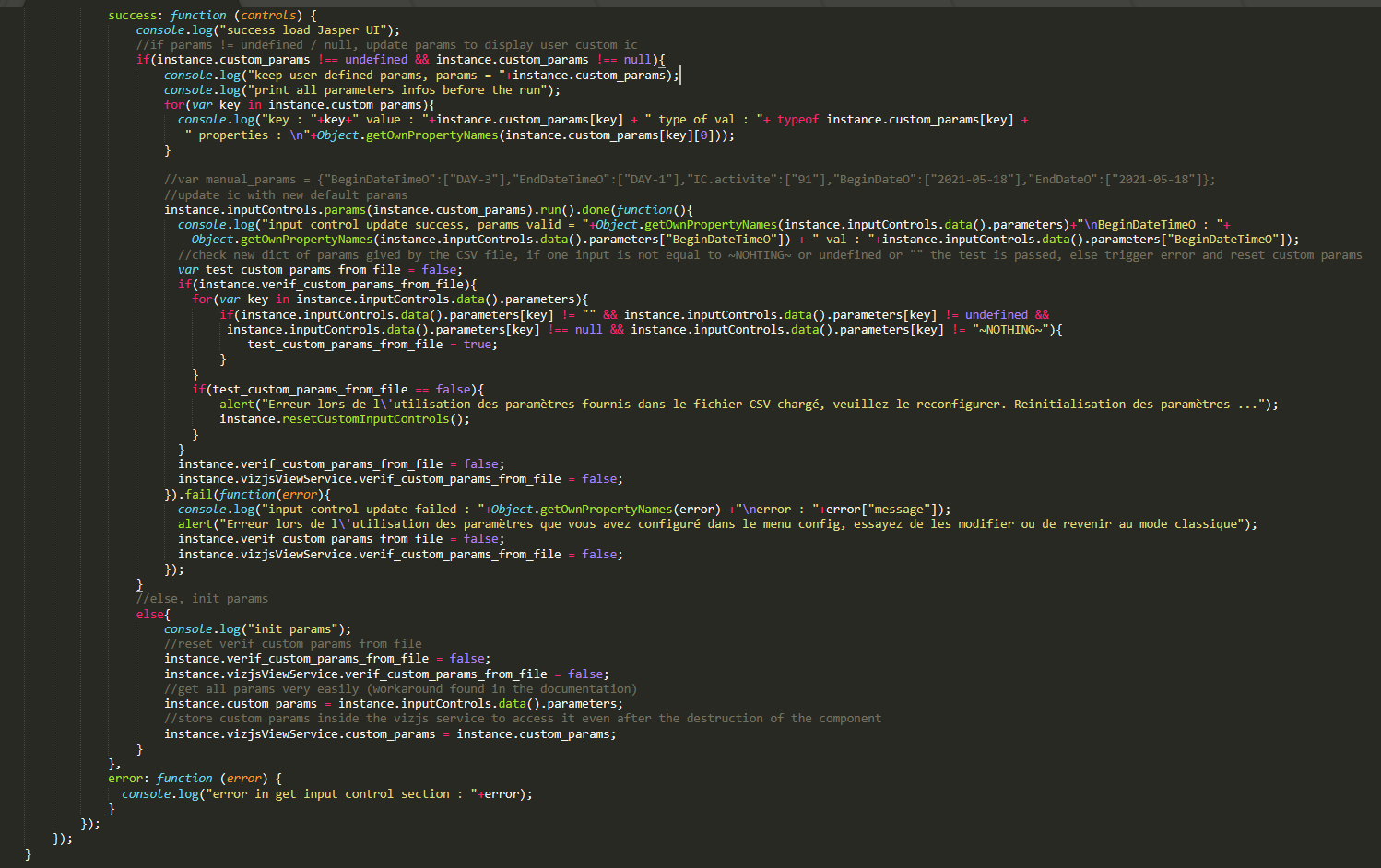
**Commentaires**

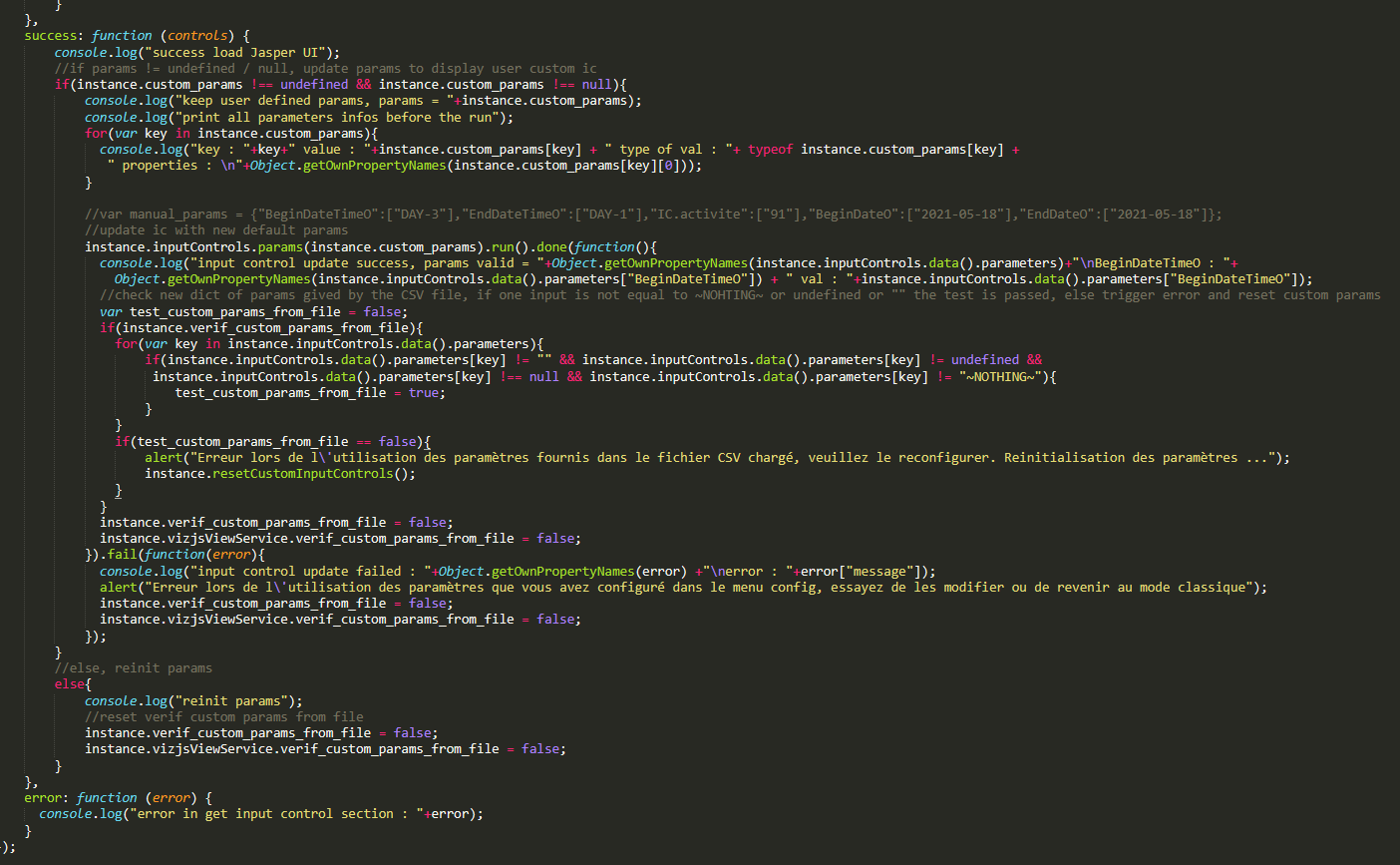
|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Configuration visualizeJS et initialisation des paramètres si un fichier CSV a été chargé (pour effectuer des vérifications ultérieurement) |
| 2 | « Event listener » sur l’événement « onchange », un ensemble de vérifications ont lieu quand l’utilisateur cherche à modifier un paramètre.  Une modification prend uniquement effet si aucune erreur n’a été constatée |
| 3 | Quand les paramètres ont été chargés avec succès (lors de l’exécution du component / redirection vers la page vizjs-config), des modifications doivent avoir lieu. |
| 3.1 | Si jamais l’utilisateur a entré des paramètres personnalisés, il va falloir mettre à jour les valeurs des paramètres obtenus avec visualize JS (dans le cas présent, les paramètres sont toujours vides lors de l’initialisation du component).  Des vérifications doivent cependant s’effectuer si les paramètres personnalisés ont été fournis par un fichier CSV (car ils ne sont pas passés par « l’event listener » « onchange » vu plus haut).  La ligne **instance.inputControls.params(instance.custom\_params).run().done**  Effectue implicitement un test (en utilisant les fonctionnalités visualize JS), si jamais nos « custom\_params » ne sont pas valides, tous les paramètres sont vidés automatiquement.  Il suffit alors de vérifier qu’au moins un élément n’est pas vide pour s’assurer de la validité du fichier CSV.  Dans le cas où il y aurait des erreurs, des messages d’alertes ont été configurés |
| 3.2 | Si l’utilisateur n’a pas entré de paramètres personnalisés (1ère exécution de la page vizjs-config par ex), les variables sont réinitialisées |



**1**

**2**





**3.2**

**3.1**

**3**

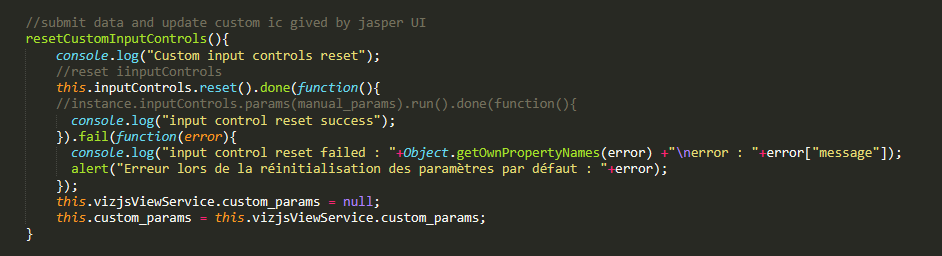
*Fonction resetCustomInputControls*

**Brève description :**

Cette fonction permet de réinitialiser la valeur des paramètres personnalisés en appelant la fonction « reset » de visualizeJS, ils n’ont donc plus aucune valeur à la fin du traitement (valeur par défaut = vide). Les variables qui stockent les paramètres personnalisés sont également vidés

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Appel de la fonction reset (visualizeJS inputControls) et réinitialisation des variables qui stockent les paramètres personnalisés |



**1**

*Fonction exportCustomParamsToCSV*

**Brève description :**

Cette fonction permet de transformer notre variable contenant les paramètres personnalisés (et donc un dictionnaire / Object javascript) en un fichier CSV qu’il est possible d’importer par la suite pour recharger un ensemble de paramètres.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Fonction permettant de convertir un Object javascript en données qu’il est possible de sauvegarder à l’intérieur d’un fichier CSV |
| 2 | Création et activation d’un élément href temporaire pour télécharger le fichier CSV, puis destruction de cet élément |



**2**

**1**

*Fonction dropHandler*

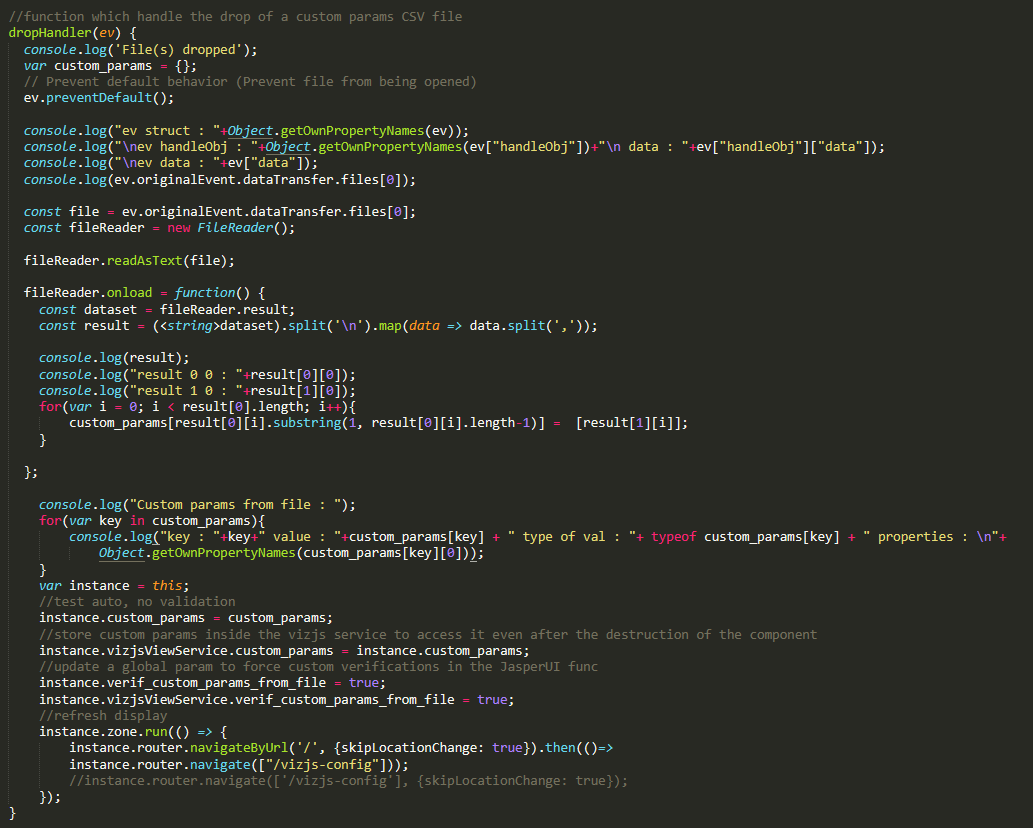
**Brève description :**

Cette fonction permet de récupérer les données d’un fichier CSV qui a été déposé sur la page vizjs-config (dans un petit rectangle bleu).

Les données sont extraites à l’aide de fonctions javascript permettant d’intéragir avec les fichiers puis elles sont stockées dans les variables du component (et du service) pour être vérifiés un peu plus tard. S’en suit une redirection vers vizjs-config (la page actuelle) uniquement pour déclencher à nouveau les fonctions permettant d’effectuer les vérifications.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Récupération des informations du fichier CSV dans une variable temporaire en utilisant les fonctions javascript spécialisées dans la manipulation de fichiers |
| 2 | Mise à jour des variables du component et du service et redirection vers la page courante pour déclencher à nouveau les vérifications |



**2**

**1**

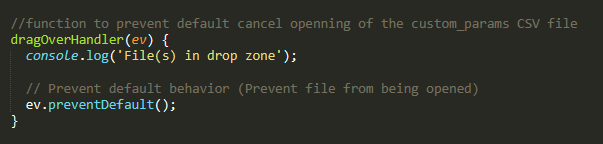
*Fonction dragOverHandler*

**Brève description :**

Fonction permettant d’éviter qu’une erreur ne survienne lorsqu’un survol de la zone où il est possible de déposer un fichier CSV a lieu.

**Commentaires**

|  |  |
| --- | --- |
| **Numéro** | **Commentaire** |
| 1 | Eviter le déclenchement de l’erreur avec la ligne « ev.preventDefault() » |



**1**