[1. Finalité du composant React: 2](#_Toc199314731)

[2. Ressources utilisés : 2](#_Toc199314732)

[3. Prérequis : 2](#_Toc199314733)

[4. ControlManifest.xml : 2](#_Toc199314734)

[5. Index.ts : 2](#_Toc199314735)

[6. downloadButton.tsx: 2](#_Toc199314736)

[7. Modifications : 2](#_Toc199314737)

# Finalité du composant React:

Ce composant react a pour but de générer un export d’une table du dataverse au choix, le tout sans limites de lignes ou de colonnes.

(Il faut tenir compte qu’au dessus d’une certaines quantités la génération peut prendre beaucoup de temps)

# Ressources utilisés :

* Framework « React » (nodeJs)
* Librairie « FluentUI »
* Librairie « xlsx »

# Prérequis :

* Power Platform Tools (extension Visual Studio Code)

# ControlManifest.xml :

Le fichier « manifest » est le fichier qui possède les propriétés essentiels de notre composant.

La partie qui nous intéresse est celle où l’on peut définir les différentes propriétés qui sont mis à disposition pour le développeur PowerApps.

Dans notre cas il y a 3 propriétés :

* dataToExport = dataset (récupérer la table du dataverse cible)
* buttonText = singleLine.Text (contenu du bouton à afficher)
* filename = singleLine.Text (nom du fichier d’export)

# Index.ts :

Le fichier « index » est le fichier principal dans lequel le code s’exécute, le bouton est donc appelé depuis ce fichier là.

Fonction init (exécuté une fois au début) :

* Initialisation var context (propriétés entrées par le dev PowerApps)
* Préparation suivi de la taille du container
* Définition du nombres de lignes par pages

Fonction updateView (exécuté à chaque modification) :

* Récupération des propriétés entrées par le dev PowerApps dans une interface
* Retourner le bouton crée en lui passant les propriétés

# downloadButton.tsx:

Le fichier downloadButton est finalement le fichier qui contient le code du bouton et toute la logique qui s’y trouve.

En premier on crée l’interface qui va recevoir les données dont on a besoin qui sont dans la page index. On en profite pour créer la variable qui contient les différentes étapes de chargement pour le spinner.

On rentre ensuite dans le code du bouton qui sera exporté :

* Déclaration des variables utilisés pour le code :
  + exportArray => state array
  + exportArayReady => state boolean
  + requestExport => state boolean
  + pageNumber => state integer
  + previousFirstRecord => state string
  + loadingState => state string from choices
  + tempArray => array
  + buttonContent => dynamic string with loadingState value
  + buttonIcon => dynamic icon with loadingState value
  + useStyles => styling used with className:
    - width
    - height
    - position
    - etc ..
* useEffect (quand les var déterminé sont modifié , on boucle)
  + si l’export est demandé :
    - s’il reste des données à récupérer :
      * préparer données (pageNumber, charger pageSuivante, enregistrer premièreLigne, mettre données d’export dans tab temporaire
      * boucle sur ligne et colonnes :
        + remplissage du tableau
    - s’il n’y a pas d’autres données, exportArray = true
  + Si l’export est prêt :
    - Remplir et télécharger le fichier excel
    - Remise à zéro de toutes les variables modifié
* Fonction onButtonClick :
  + Modifier loadingState
  + Passer requestExport à vrai (ce qui lance le useEffect)
* L’objet retourne :
  + Bouton avec le code ci-dessus

# Modifications :

### Rajouter une propriété pour le dev PowerApps :

Une propriété se rajoute en premier dans le manifeste comme property ou dataset ou autre

Il faut ensuite l’intégrer dans index, dans l’interface qui répertorie les différentes popriétés

Dans la page downloadButton, ajouter la nouvelle propriété dans l’export d’interface au début et dans l’initialisation de l’export de l’objet bouton.

### Modifier la taille des pages des requête à l’API :

La taille des pages des requêtes est déterminé dans le fichier index.ts

Dans le init, il y a le setPageSize qui détermine le nbre de lignes par pages.