|  |
| --- |
| Mohamed AL-AZAWI  12/06/2024 |

Documentation de l’Application ExcelJS



Table des matières

[Introduction 3](#_Toc169098892)

[Créer Répertoire du projet 3](#_Toc169098893)

[La page HTML 4](#_Toc169098894)

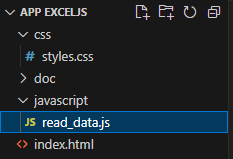
[La page read\_data.js 5](#_Toc169098895)

# Introduction

Ce projet consiste à crée une application qui converti les tableaux html en tableau Excel en utilisant la librairie excelJS. Mais avant cela il faudra créer une variable qui stock les données des objets JavaScripts pour pouvoir les transposer sur un tableau HTML dans notre page index.

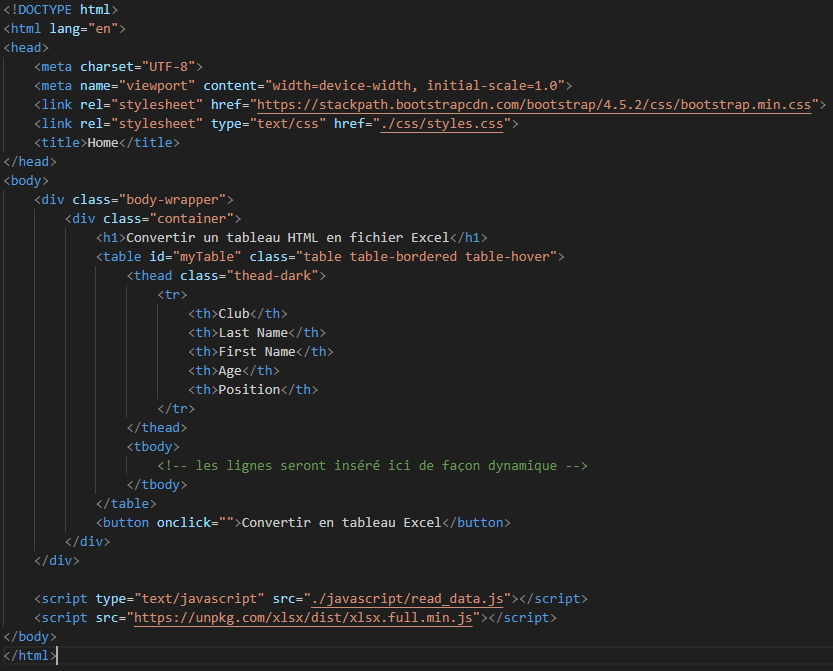
# Créer Répertoire du projet

Pour ce projet nous allons mettre un fichier html où nous allons afficher la logique de l’application, on pourrait l’appeler view.html. Ensuite à la racine nous allons créer deux autres répertoires nommé CSS et javascript. À l’intérieur du répertoire CSS il y’aura la feuille de style styles.css et à l’intérieur du répertoire js nous allons mettre read\_data.js qui utilisera la librairie excelJS et qui aura la logique de l’application qui sera transposer sur le view.html.



# La page HTML

La page html contiendra la table qui affiche les données de façon dynamique, le lien vers la page read\_data.js et le ce script : <script src="https://unpkg.com/xlsx/dist/xlsx.full.min.js"></script> qui est une bibliothèque JavaScript appelée xlsx.js qui est hébergée sur un CDN (Content Delivery Network). Cette bibliothèque vous permet de travailler avec des fichiers Excel dans votre application web. Elle fournit des fonctionnalités pour lire et écrire des fichiers Excel, convertir entre différents formats de données.



# La page read\_data.js

La première chose que nous avons est une variable que nous avons appelée players qui contient an array avec des objets JavaScript dedans c’est ce qui va peupler notre tableau.

let players = [

        {

            club:'Real Madrid',

            nom:"Kylian",

            prenom:"Mbappe",

            age:25,

            position:"ST",

        },

        {

            club:'Barcelona',

            nom:"Lewandowski",

            prenom:"Robert",

            age:35,

            position:"ST",

        },

        {

            club:'Arsenal',

            nom:"Saka",

            prenom:"Bukayo",

            age:20,

            position:"M",

        },

        {

            club:'Paris Saint-Germain',

            nom:"Neymar",

            prenom:"Jr",

            age:30,

            position:"F",

        },

        {

            club:'Liverpool',

            nom:"Salah",

            prenom:"Mohamed",

            age:29,

            position:"F",

        },

];

Deuxièmement on va créer la fonction qui va peupler le tableau avec ces objets :

    // peupler le tableau avec les données de l'objet players

    let tbody = document.querySelector('#myTable tbody');

    players.forEach(player => {

        let row = document.createElement('tr');

        ['club', 'nom', 'prenom', 'age', 'position'].forEach(key => {

            let cell = document.createElement('td');

            cell.textContent = player[key];

            row.appendChild(cell);

        });

        tbody.appendChild(row);

    });

**let tbody = document.querySelector('#myTable tbody');** : Cette ligne utilise la méthode querySelector pour sélectionner le tbody dans l'élément avec l'ID myTable. Le résultat est stocké dans la variable tbody.

**players.forEach(player => {...});** : Cette ligne utilise une méthode de boucle forEach pour exécuter une fonction sur chaque élément du tableau players. La fonction prend un player comme argument.

**let row = document.createElement('tr');** : Cette ligne crée un nouvel élément HTML tr (table row) et le stocke dans la variable row.

**['club', 'nom', 'prenom', 'age', 'position'].forEach(key => {...});** : Cette ligne utilise à nouveau la méthode de boucle forEach pour exécuter une fonction sur chaque élément du tableau. La fonction prend une key comme argument.

**let cell = document.createElement('td');** : Cette ligne crée un nouvel élément HTML td (table data) et le stocke dans la variable cell.

**cell.textContent = player[key];** : Cette ligne définit le contenu textuel de l'élément cell à la valeur de la propriété key de l'objet player. Par exemple, si key est 'club', alors player[key] est le club du joueur, et **cell.textContent = player[key];** définit le contenu textuel de la cellule à ce club.

**row.appendChild(cell);** : Cette ligne ajoute l'élément cell à la fin de l'élément row. La méthode appendChild est une méthode qui permet d'ajouter un nœud à la fin de la liste des enfants d'un nœud parent spécifié.

**tbody.appendChild(row);** : Cette ligne ajoute l'élément row à la fin de l'élément tbody.

Et enfin on va ajouter un bouton pour télécharger les données en tableau excel :

    document.querySelector('button').addEventListener('click', function() {

        let wb = XLSX.utils.table\_to\_book(document.querySelector('#myTable'));

        XLSX.writeFile(wb, 'Players.xlsx');

    });

**document.querySelector('button'**) : Cette ligne sélectionne le premier bouton dans le document HTML.

**.addEventListener('click', function() {...})** : Cette ligne attache un gestionnaire d'événement 'click' au bouton sélectionné. Lorsque le bouton est cliqué, la fonction sera exécutée.

**let excelSheet = XLSX.utils.table\_to\_book(document.querySelector('#myTable'));** : Cette ligne sélectionne le tableau avec l'ID 'myTable' le convertit en un "livre" XLSX (essentiellement un objet représentant un fichier Excel), et stocke ce livre dans la variable excelSheet.

**XLSX.writeFile(excelSheet, 'Players.xlsx');** : Cette ligne écrit le "livre" XLSX stocké dans wb dans un fichier appelé 'Players.xlsx'. Si le fichier existe déjà, il sera écrasé. Sinon, un nouveau fichier sera créé.

# CSS

Pour le CSS j’ai stylisé les choses suivantes :

.body-wrapper {

    display: flex;

    justify-content: center;

    align-items: center;

    height: 100vh; /\* pour que le wrapper prennent 100% de l'écran utilisé \*/

}

**Display: flex;** : Cette propriété transforme l'élément en un conteneur flex. Les éléments enfants de ce conteneur deviennent des éléments flex et leur disposition est gérée par les propriétés flexbox.

**justify-content: center;** : Cette propriété aligne les éléments flex le long de l'axe principal (horizontalement dans le cas par défaut). Center signifie que les éléments seront centrés le long de cet axe.

**align-items: center;** : Cette propriété aligne les éléments flex le long de l'axe transversal (verticalement dans le cas par défaut). Center signifie que les éléments seront centrés le long de cet axe.

**height: 100vh;** : Cette propriété définit la hauteur du conteneur flex. vh est une unité de mesure relative à la hauteur de la fenêtre de visualisation. 100vh signifie 100% de la hauteur de la fenêtre de visualisation.

En résumé, ce code crée un conteneur flex qui occupe toute la hauteur de la fenêtre de visualisation et centre ses enfants à la fois horizontalement et verticalement.

Le button:

button {

    display: block;

    margin: 0 auto;

    text-align: center;

    padding: 10px 20px; /\* padding à l'interieur du bouton \*/

    background-color: #05b105;

    color: #fff;

    border: none;

    border-radius: 5px;

    cursor: pointer;

}

**Display: block;** : Cette propriété définit le type d'affichage de l'élément. Block signifie que l'élément occupe toute la largeur disponible et a une nouvelle ligne avant et après lui. C'est important pour centrer l'élément avec margin: 0 auto;

**margin: 0 auto;** : Cette propriété définit les marges autour de l'élément. 0 est la marge supérieure et inférieure, et auto est la marge gauche et droite. Lorsqu'un élément est de type block, définir les marges gauche et droite sur auto le centrera horizontalement dans son conteneur.