PROJETO CUSTOMIZAÇÕES

Importação de Planilha via JS

Amanda Alvim

1. Objetivo

O objetivo dessa customização é realizar a importação de uma planilha em tela para uma grid do formulário facilmente, isso se deu por conta de não necessitar fazer um controller para ler a planilha js e devolver para tela e inserir na grid, então já fazemos em tela esse "transporte" de informações.

O js contendo um exemplo da implementação se encontra no Projeto Base no nosso git (http://git.lecom.com.br/PSP/Projeto-Base-BPM) com o nome de **exemploChamadaExcel.js**, que está no diretório upload/rotinaJS.

2. Como usar

No momento de utilizar essa implementação você precisará pegar o arquivo **xlsx.full.min.js** que está no diretório do projeto do git /upload/rotinaJS e subir no ambiente do seu cliente e depois fazer algo parecido com a implementação que vou explicar no js do seu modelo e assim já estará funcionando a importação de planilha via JS .

Lembrando que importante vocês definirem um padrão da planilha importada, pois vocês usaram os nomes das colunas da planilha no js para fazer a inserção na grid, então precisam saber o que vem.

3. Fonte Javascript

exemploChamadaExcel.js

```
$(document).ready(function() {
    Form.fields("TEMP_PLANILHA_PREENCH").subscribe("DOCUMENT_IMPORT_FILE_SUBMITTED",
          function (storeId, inputId, file) {
        Form.showLoader({ description: "Importando dados da Planilha", icon: true });
        setTimeout(function () {carregaPlanilha(file,"TEMP_PLANILHA_PREENCH");}, 1000);
    });
});
function carregaPlanilha(file,campo){
    //esconde o botao de fechar
    var fileReader  = new FileReader();
    var inicio      = performance.now();

    fileReader.onload = function(event) {
        var data        = event.target.result;
        var workbook    = XLSX.read(data, {type: "binary"});

        workbook.SheetNames.forEach( function(sheet) {
                console.log(workbook.Sheets[sheet]);
                let rowObject  = XLSX.utils.sheet_to_row_object_array(workbook.Sheets[sheet]);
                let jsonObject = JSON.stringify(rowObject);
                PreencheGrid(jsonObject, campo, "DESPESAS");
        });
  };
  fileReader.readAsBinaryString(file);
}
function PreencheGrid(dados, campo, gridDestino){
    var dadosJson = JSON.parse(dados)
    var grid = Form.grids(gridDestino);
    for(i in dadosJson)
    {
        var item = dadosJson[i];

        grid.insertDataRow({
                "TIPO" : item.tipo
                , "QTD" : item.qtde
                , "VALOR" : item.valor
                , "TOTAL" : item.total
            });
    }
    Form.hideLoader();
    documentoStore.store[campo].flashMessage = { type:"notice", message:"As informações da planilha foram carregadas na solicitação, clique em importar para continuar" };
    $($('#documentModal .modal-footer-container .button-group a')[0]).hide();
    Form.apply();
}
```

O fonte acima é um exemplo para que vocês possam utilizar nos modelos de vocês, vamos ver o que estamos usando nesse exemplo.

Todo Js que criamos começamos com o document.ready para significar que estamos rodando no momento do carregamento do formulário.

Dentro do document ready estou criando um evento para um campo do meu modelo do tipo template para que a hora que for selecionar um arquivo da minha máquina ele faça a importação dessa planilha para grid.

Para isso utilizamos a APIJS, colocamos sempre começando por Form.fields(), que estou falando qual campo eu quero adicionar o evento, então no exemplo acima o campo do tipo template chama **TEMP_PLANILHA_PREENCH**, após o fields chamamos a função subscribe, que nela vamos dizer qual evento queremos sobrescrever, no caso passamos **DOCUMENT_IMPORT_FILE_SUBMITTED**, após isso inicializamos a function com os atributos

que a APIJS já nos envia, storeId, inputId, file, vamos utilizar principalmente o file que é o arquivo escolhido pelo usuário.

Na **linha 4** estamos utilizando um método da APIJS que abre uma modal com loader na tela e por conta dessa modal para que o usuário não mexa em mais nada enquanto estaremos fazendo a leitura da planilha e inserindo dados na grid.

Na **linha 5** chamamos um settimeout apenas para dar um tempo após o usuário escolher o arquivo para chamar nosso método principal que é o método **carregaPlanilha**. Para o método **carregaPlanilha** vamos passar o file e o nome do campo.

Na **linha 10** inicializamos a classe FileReader que está dentro do js que estamos utilizando o xlsx.full.min.js, a partir do FileReader faremos o onload como está na **linha 13** que começará a ler a planilha.

Após fazer a chamada do onload, esse evento retorna os dados do arquivo, que é transformado na variável workbook da **linha 15.** Com isso é possível realizar o foreach de todas as linhas da planilha, para cada linha da planilha é feito a chamada conforme **linha 19** que retorna uma String com tudo que tem naquela linha, e dessa String transformamos em um JSON para facilitar nosso trabalho para inserção na grid.

Nesse exemplo na **linha 21** chamamos o outro método que é o PreencheGrid passando para ele o json, o campo do tipo template e o identificador da grid.

Nesse método fazemos na **linha 27** só o parse do json da linha. Inicializamos na **linha 28** utilizando a APIJS a grid a qual vamos fazer o insert e para isso usamos o Form.grids passando o identificador da grid em questão.

Na **linha 29** fazemos um for dessa linha que veio apenas para garantir caso venha mais informações, e desse for pegamos o item em questão que seria a linha inteira com nome da coluna e valor.

Nas **linhas 33 a 38** estamos pegando a partir da variável grid e utilizando o método da APIJS insertDataROW e para ele precisamos passar um json do que queremos inserir, então montamos o nome do campo : e o valor, no caso esse valor estamos pegando do objeto item que é a linha retornada do js.

Após isso na **linha 40** fazemos o fechamento da modal de loader, aquela que foi aberta no começo do arquivo.

Nas **linhas 41 e 42** estamos utilizando formas para conseguir mexer com o elemento da modal de importação de arquivo na primeira colocamos uma mensagem para o usuário e na segunda deixamos o botão habilitado para ele “finalizar” a importação, que no caso quando usuário clicar no botão de importar vai ser preenchido o valor no campo template.

Na **linha 43** chamamos a função apply para te fato “aplicar” a mudança que queremos no formulário.