# 06/03/2018

Criar um aplicativo que permita fazer passagem de serviço. O ideal é um modelo parecido com o “discussão.nsf”, uma extensão utilizada no IBM Lotus Notes.

Esta extensão permite ordenar as passagens por Autor, Data ou Assunto e também há a possibilidade de se responder a um assunto, criando-se assim um histórico sobre o tema indiferente do autor ou data. Para melhor organização, existe uma árvore de temas onde podem ser apontadas as palavras chaves associadas ao assunto, isto seria como se um assunto pudesse pertencer a um ou mais pontos de acesso.

Os assuntos discutidos são marcados como lidos ou não para cada usuário. Há também a possibilidade de se anexar arquivos ao assunto em tela.

Estes são os pontos essenciais para o aplicativo. No novo aplicativo pode-se ainda criar uma forma de se conseguir criar relatórios (em pdf e imprimível) além de conectarmos os assuntos a uma árvore de equipamentos quando o assunto tratar de equipamentos, como o que ocorre no SAP, de forma a ser navegável em ambos os sentidos, ou seja, ao pesquisarmos o equipamento será possível encontrar a passagem de serviço associada, e ao se pesquisar a passagem será possível visualizar detalhes do equipamento.

Para esta integração será necessário a criação de um outro aplicativo, que se trata de um grande e poderoso banco de dados que terá a função de integrar a maior quantidade de informação possível disponível à Automação de P74.

Este aplicativo deveria ser uma conjunção de um SGBD como o MySQL para armazenar as passagens e os anexos, usar o Java como back-end e HTML com Javascript para o front-end devido às ferramentas de desenvolvimento já estarem instaladas e o HTML rodar sem necessitar de instalações maiores das que já há, que é apenas um navegador web.

Pode ser que no futuro haja um servidor externo para replicar estas informações para um ambiente fora dos domínios de automação daqui da 74.

# 07/03/18

Esboço da tela principal. Esta tela deverá ter apenas um login, para que se possa reconhecer o usuário e então aplicar-se os filtros.

Inserido o ícone e o título do aplicativo.

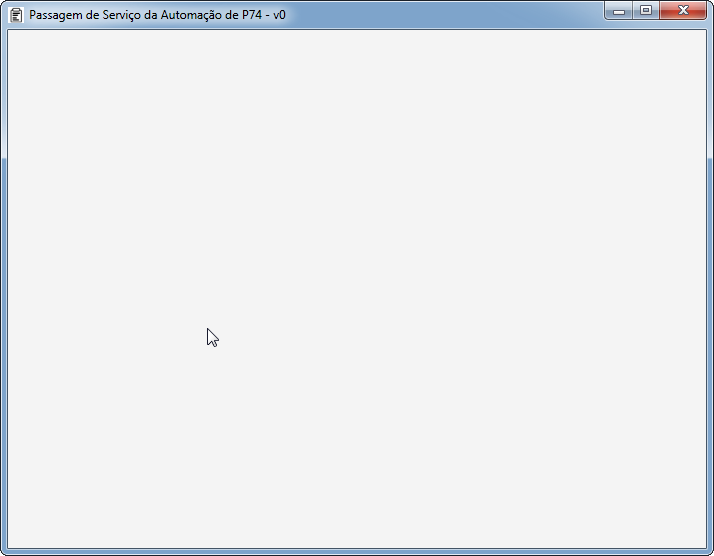


Figura - Versão 0 do aplicativo. Título e ícone

Inserida a tabela central e as divisórias, usuário e horário da máquina local.

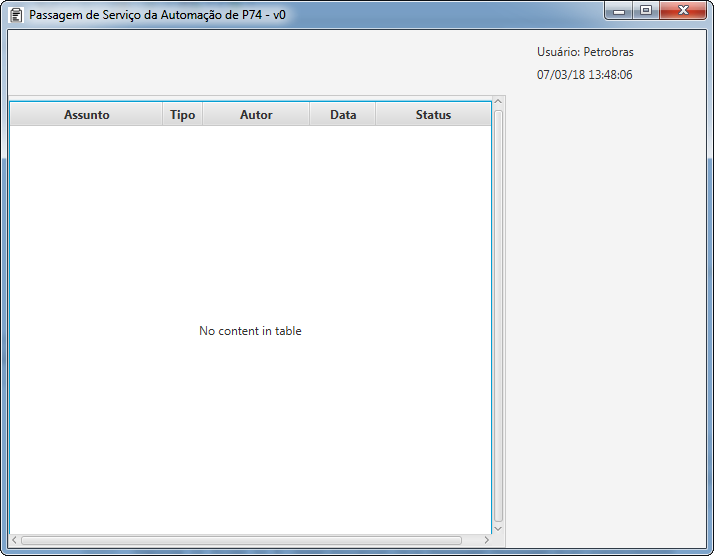
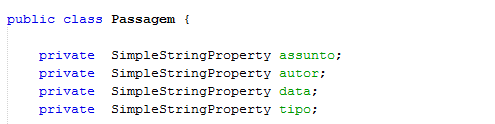
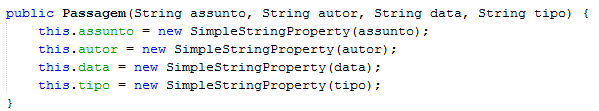


Figura - Divisões iniciais da janela principal

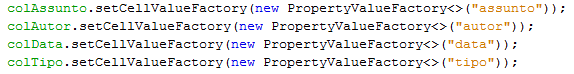
Primeiro teste com uma pequena lista de itens adicionados. Foi feita a seguinte sequência para o teste:

1. Criada a classe de DataModel Passagem, que tem os parâmetros “assunto”, “tipo”, “data”, “autor” e “status”.

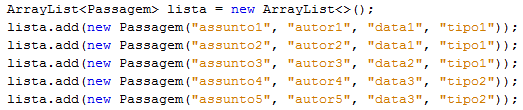




1. Feito a conexão da entidade TableColumn com a entidade Passagem, isto garante o bind dos atributos através de um mapa key/value.
2. O método setCellValueFactory de TableColumn garante um Callback para a entidade coluna, que faz o valor da célula variar ao ser variado o valor da String associada no mapa key/value.



1. Criada uma lista de teste:



1. A lista se torna observável (por call-back da classe).



1. A lista observável então pode ser passada para a tabela. Os valores então são dinâmicos deste ponto em diante.



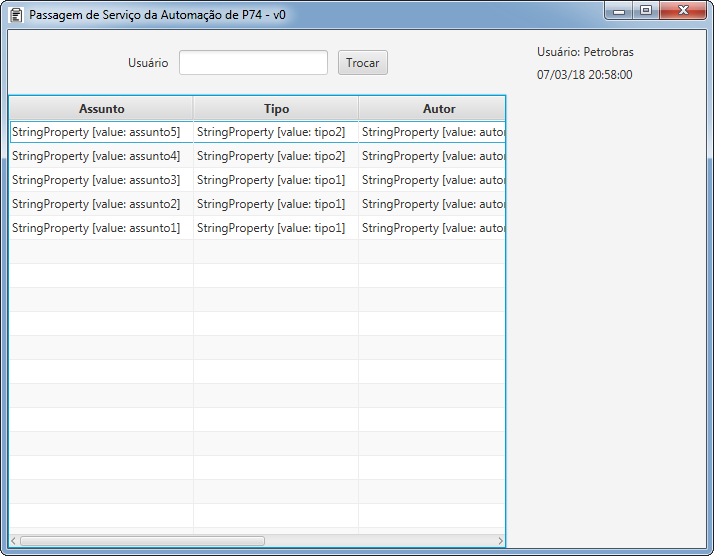


Figura - Resultado do primeiro teste com lista

Alguns ajustes foram feitos no POJO da classe Passagem, para que o retorno em cada coluna seja apresentado como uma String e não um StringProperty.

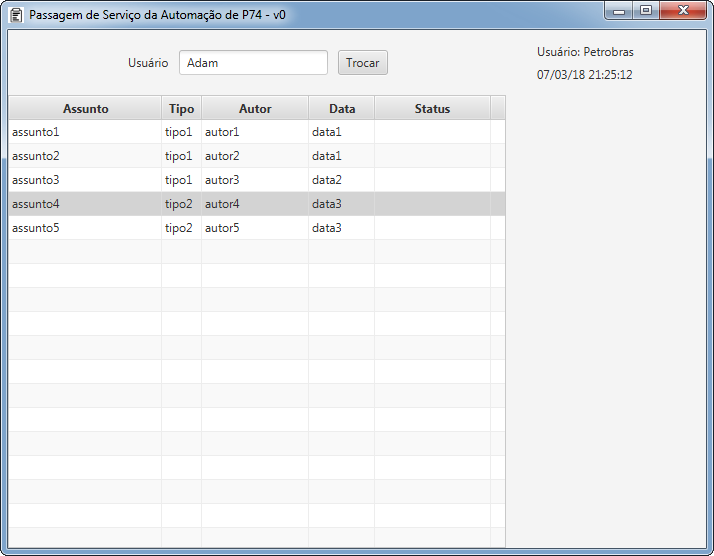


Figura - Resultado do ajuste do POJO e do Bean

# 08/03/18

Inserido um wrap Accordion, que permite encapsular os assuntos em comum, como as réplicas, e então possibilitar um melhor gerenciamento dos assuntos.

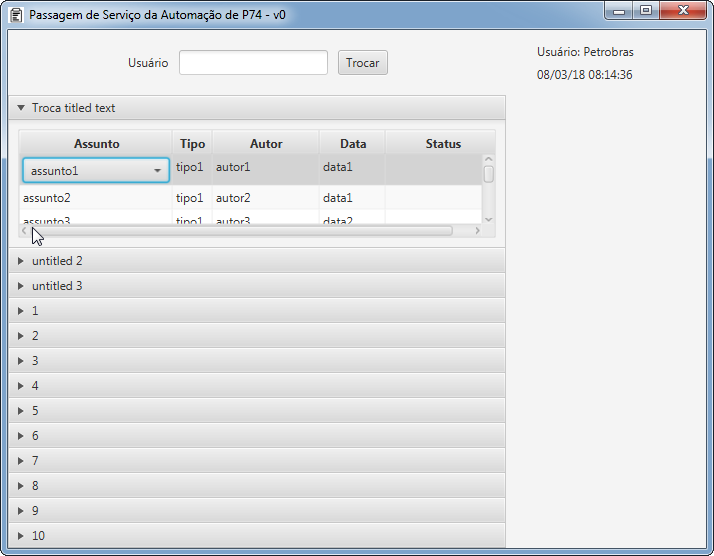


Figura - Teste de aninhamento

Foi construída também uma CellFactory para que se possa ter uma lista suspensa dentro das células, facilitando-se assim a padronização dos valores inseridos.

Foi feito teste inserindo programaticamente as TableViews dentro dos AnchorPanes que são encapsulados por TitledPanes e Accordions.

Feito backup com o nome 20180308.zip no diretório [E:\Adam\Projetos\Java\Passagem\Backup](file:///E:\Adam\Projetos\Java\Passagem\Backup)

Neste ponto os teste foram satisfatórios. Daqui em diante o trabalho será em tornar toda a dinâmica interligada programaticamente, isto é, o usuário deverá ser identificado automaticamente ou se identificar de forma manual, deverá poder escolher um assunto e então conseguir ler seu conteúdo, deverá conseguir responder ao assunto, imprimir, excluir ou editar o assunto caso ele seja o dono. O próximo passo é gerar o assunto e armazená-lo em um local de acesso ao aplicativo.

O primeiro desafio é a criação do documento. No momento a melhor hipótese é criar um documento no Word e salvar uma cópia como HTML. Isto é interessante pois atende em parte a proposta do Elder que consiste em fazer a passagem de serviço rascunhada no Word e depois copiá-la para o Aplat. Como a estratégia de se gerar um HTML poderá ser inconveniente, tentarei automatizar a tarefa no Word. Este automatismo deverá gerar o documento sozinho, gravar no local correto e então o aplicativo deverá busca-lo e então o gerenciar, e só iremos utilizar o aplicativo para gestão, e não edição.

Uma outra aproximação pensada é transformar o Word em um XML após ser salvo, isto tem um custo programático alto inicial, porém evitaria que tenhamos vários documentos Word, um para cada assunto. As marcações XML iriam particionar os campos do documento Word e seria legível ao nosso aplicativo, colocando cada assunto em sua devida estrutura, ou seja, não teríamos um documento com assuntos irrastreáveis programaticamente.

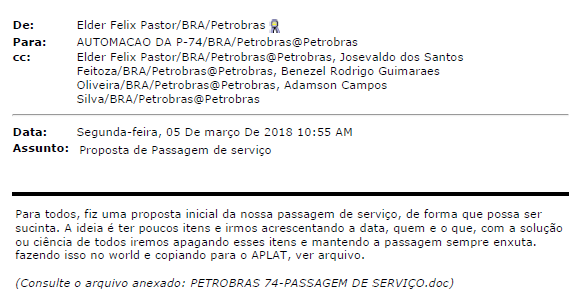


Figura 6 - Resumo da proposta de Elder

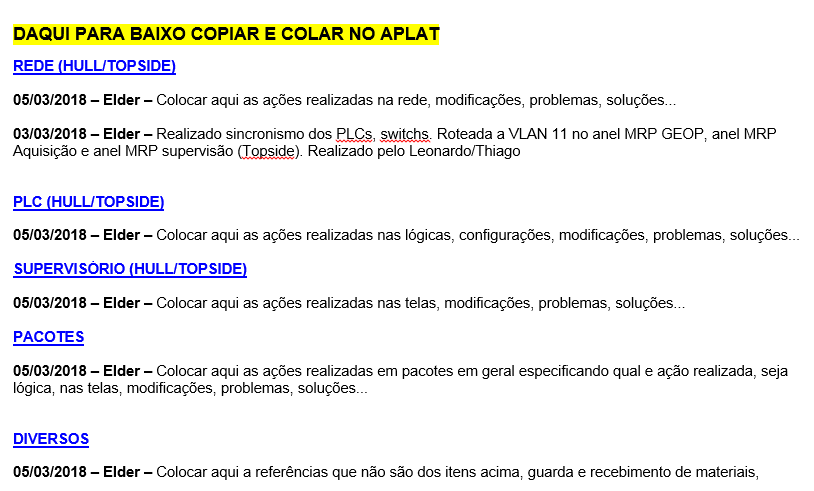


Figura 7 – Anexo da proposta de passagem de serviço do Elder

Na proposta do Elder há poucos itens. Isto é benéfico no ponto de vista de geração de assuntos, mas não tão benéfico quanto à organização e detalhamento de cada assunto. Quanto ao Aplat, o mesmo não tem uma ferramenta de busca de passagens, seja qual for o critério de pesquisa. O aplicativo desenvolvido contornaria este problema em um ambiente externo ao Aplat, mas mantendo o mesmo conteúdo do Aplat.

Outro problema do Aplat é a estrutura de itens da passagem de serviço. É onerosa para que deseja incluir um assunto dentro de um item da árvore e após inserido o item é quase inútil sua utilização, uma vez que os assuntos não são concatenados no decorrer de sua utilização.

Neste ponto o foco passa a ser o Word e o XML. Um investimento grande deverá ser colocado aqui para programar o Word para inserir os Markups nos locais certos a fim de que o aplicativo desenvolvido possa lidar com o conteúdo da passagem de serviço.

Para iniciar a parte de XML, deve-se criar um arquivo no word e salvá-lo com a extensão docx, e depois renomeá-lo para .zip.

Este automatismo foi implementado no aplicativo gestor. Basta que a passagem de serviço seja salva como .docx no diretório padrão. O diretório utilizado para teste é [E:\Adam\Projetos](file:///E:\Adam\Projetos).

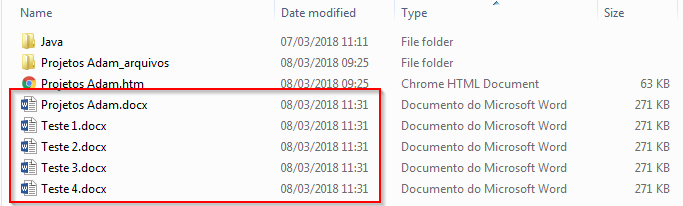


Figura - Diretório padrão com os arquivos Word

O aplicativo é executado e faz as devidas mudanças.

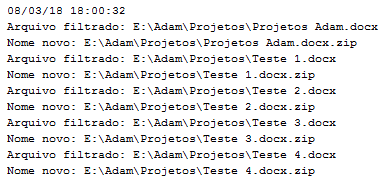


Figura - Log de modificações aplicadas

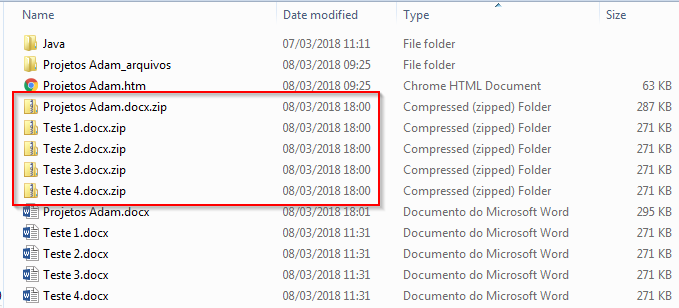


Figura - Resultado da modificação pelo aplicativo

Esta modificação é necessária para que se possa obter os arquivos XML gerados pelo Microsoft Office. Estes arquivos XML poderão ser utilizados pelo aplicativo em desenvolvimento.

# 09/03/18

Criado um botão no aplicativo para que o usuário possa começar a criar uma passagem. Isto irá abrir um editor HTML que permitirá o usuário escrever textos e inserir imagens, além de poder fazer edições no texto.

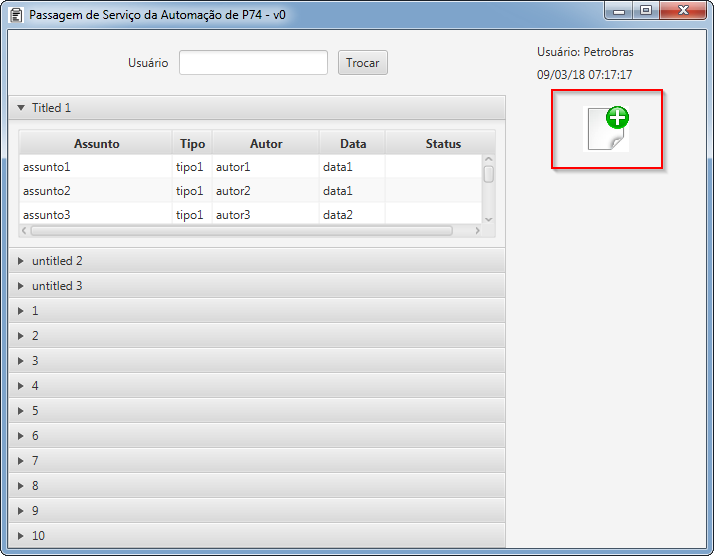


Figura 11 - Botão para iniciar nova passagem de serviço

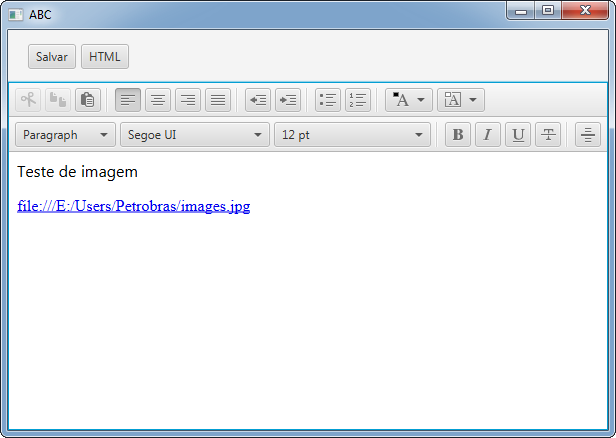


Figura 12 - Editor da passagem de serviço

O Editor não foi idealizado para se inserir imagem, mas há uma forma de se inserir as imagens, mas não tem como redimensiona-las. Para que a imagem seja inserida, posicionamos o cursor no local da inserção e então arrastamos uma imagem para dentro do editor. Assim que soltarmos a imagem aparecerá um link trazendo o caminho par o local da imagem. Uma atenção aqui requerida, não se deve clicar neste link, pois o link será aberto no editor e iremos perder todo o conteúdo do editor, e não há como desfazer.

O interessante do aplicativo é que podemos escrever qualquer HTML que uma página será gerada adequadamente ao se pressionar o botão de salvar, inclusive pode haver códigos como javascript, actionscript, css, php, etc.

Pensado em armazenar os dados de cada passagem em um arquivo XML. Iniciado o processo de handler do DOM no aplicativo. Como teste foi criado um arquivo XML com alguns Elementos e o aplicativo foi capaz de o ler e encontrar alguns valores:



Figura 13 - XML de exemplo

No aplicativo foi inserida uma classe para lidar com o XML. Ela procura o arquivo no diretório indicado e então faz o Parse do arquivo utilizando a biblioteca org.w3c. Como teste foi realizada a pesquisa do atributo do elemento <pai> e percorrido os nós filhos de <pai> à procura dos valores do tag <filhos1>. Como apenas o elemento <pai id=”01”> tem este tag como filho, apenas este resultado é visualizado abaixo. E não apenas o tem como o tem repetido. Mesmo assim o aplicativo conseguiu elencar todos os valores.

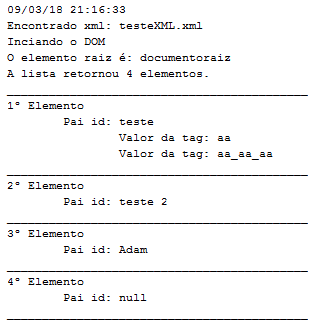


Figura 14 - Resultado de leitura do DOM

# 10/03/2018