DESAFIO 2:

Para este desafio começaremos importando o *xpz* dos materiais que fornecemos.

Dentro dos elementos para importar, há uma transação CustomUser que terá usuários personalizados contra os quais GAM irá conferir para verificar identidades. Teremos um procedimento CustomUserLoad que se encarregará de preencher a transação CustomUser com usuários padrão. A tabela associada a essa transação. Haverá *SDTs* auxiliares (estes servem para implementar o procedimento de autenticação Custom). E domínios e pastas que servem para uma estrutura mais organizada e centralizada dos elementos a serem importados.

Uma vez importados os objetos, avançamos para criar o procedimento auxiliar de autenticação Custom.

Este procedimento será Main, e o código se apresenta da seguinte forma.

Antes de aprofundar neste, adicionamos as regras onde temos uma string de entrada do tipo VarChar e uma de saída também VarChar.

Agora no código, precisamos de uma variável *Key* do tipo VarChar que estará definida no GAM para o tipo de autenticação Custom, a qual mostraremos mais adiante.

Em seguida tem que utilizar os SDT auxiliares que foram importados, para as diferentes variáveis que são carregadas.

Depois temos strings associadas ao nome de usuário e senha, também VarChar, onde vemos que são descriptografadas a partir da string de entrada que é um JSON utilizando a key, que é assim estabelecido pelo GAM.

Então é carregado um SDT, que terá a saída de execução deste procedimento.

A versão do JSON do GAM, para este caso, será 2.0, onde também funciona com a 1.0. Teria apenas que adaptar o exemplo a esta.

Finalmente é executada a sub-rotina *ValidUser* que validará os parâmetros de entrada que são o UserLogin e UserPassword em relação à tabela CustomUser que importamos, comparando os campos e, como um extra, verificando se o usuário está ativo. Se todas as condições forem atendidas, serão carregados todos os dados do usuário no SDT de saída, que então GAM gerenciará. Caso contrário, se não forem atendidas todas as condições, é retornado um erro no Status que fornece GAM.

O passo seguinte é criar o tipo de autenticação Custom no Backoffice web do GAM. Para isso fazemos login com o usuário administrador, settings, authentication types, e clicamos em Add.

Selecionamos o tipo Custom.

Como nome, colocaremos “Custom login”. A função, “somente autenticação”. Estará habilitado. A versão do JSON como dissemos era a 2.0. Aqui geramos a chave que estávamos comentando hoje, que introduziremos em nosso procedimento criado posteriormente. Como nome de arquivo para este caso de .NET, o nome será esta dll (começa com “a” porque assim começam os procedimentos Main nestas versões do GeneXus). O Package terá o valor

“GeneXus.Program” (que é o namespace que englobaria a classe do procedimento). A classe que se aplica neste caso é a mesma que o *File name*, mas sem o *“.dll”.*

Copiamos a chave e confirmamos.

Agora no GeneXus, colamos a chave copiada na variável Key, e o passo seguinte é fazer um build para que sejam aplicadas todas as mudanças.

Como temos a nova tabela CustomUser, é necessário reorganizar a base de dados para criá-la. Este processo leva alguns minutos, então avançamos rapidamente.

Finalizado este processo, temos que carregar os usuários na tabela CustomUser e para isso utilizamos o procedimento entregue, que podemos executar sem Building porque já foi feito antes o Build All.

Uma vez terminado isso, já podemos ir para o login da aplicação, onde vemos que temos para escolher mais de um tipo de autenticação: o padrão e o custom login, que é o novo.

Usaremos um dos usuários carregados (que é Mick) com senha de 1 a 6.

Vemos que o login foi bem-sucedido, mas após selecionar ir para o Backoffice do GAM ele nos retorna como não autorizado. Isto acontece porque o utilizador Mick não tem permissões para ver o backoffice, mas é apenas uma questão de permissões. O desafio foi concluído.