CHALLENGE 2:

In this challenge, we will start by importing the *xpz* of the resources provided to you.

Among the elements to import, there is a CustomUser transaction that will have custom users who will be checked by GAM to verify their identities. We will have a CustomUserLoad procedure that will populate the CustomUser transaction with default users. The table associated with that transaction. There will be auxiliary *SDTs* (these are used to implement the Custom authentication procedure). In addition, domains and folders that are used for creating a more organized and centralized structure of the elements to be imported.

Once the objects are imported, we create the auxiliary Custom authentication procedure.

This procedure will be Main, with code that looks as follows.

Before looking into it, we add the rules that have an input string of VarChar type, and an output string also of VarChar type.

In the code, we need a *Key* variable of VarChar type that will be defined in the GAM for the Custom authentication type, which we will show later.

Next, we have to use the auxiliary SDTs that were imported, for the different variables that are loaded.

Then we have strings associated with the username and password—also VarChar—which are decrypted from the input string that is a JSON using the key, as established by GAM.

Next, an SDT is loaded, and will have the output of the execution of this procedure.

The JSON version of GAM for this case will be 2.0, but it also works with 1.0. It would only be necessary to adapt the example to this one.

Lastly, the *ValidUser* subroutine is executed that will validate the input parameters which are UserLogin and UserPassword against the CustomUser table that we imported, comparing the fields, and also verifying that the user is active. If all conditions are met, all the user's data will be loaded into the output SDT that GAM will handle. Otherwise, if not all conditions are met, an error is returned in the Status provided by GAM.

The next step is to create the Custom authentication type in the GAM web back office. To do so, we log in with the administrator user, go to settings, authentication types, and click on Add.

We select the Custom type, and name it “Custom login.” The “only authentication” function will be enabled. The JSON version, as we said, is 2.0. Here we generate the key that we mentioned earlier, which we will introduce in the procedure created later on. The filename for this .NET case will be this dll (it begins with “a” because this is how Main procedures start in these versions of GeneXus). The Package will have the value “GeneXus.Program” (which is the namespace that would encompass the procedure class). The class that applies in this case is the same as the *Filename* but without *“.dll.”*

We copy the key and confirm.

In GeneXus, we paste the key in the Key variable, and the next step is to do a build so that all the changes are applied.

Since we have the new CustomUser table, it is necessary to reorganize the database to create it. This process takes a few minutes, so we move quickly.

After this process is completed, we have to load the users into the CustomUser table. For that, we use the delivered procedure, which can be run without building because the Build All has already been done before.

Once this is finished, we can go to the application login where we have more than one type of authentication to choose from: the default one, and custom login which is the new one.

We will use one of the loaded users (Mick) with a password from 1 to 6.

Note that the login is successful, but after selecting to go to the GAM Backoffice it returns us as unauthorized. This happens because the user Mick does not have permissions to see the backoffice, but it is just a permissions issue. The challenge was completed.