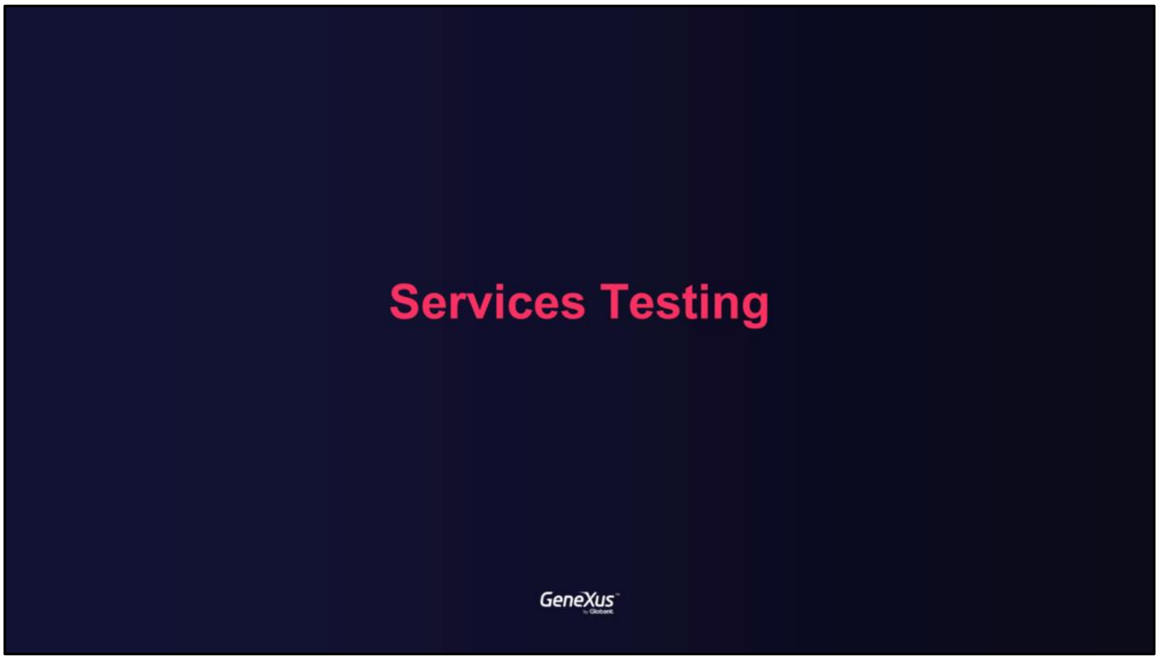
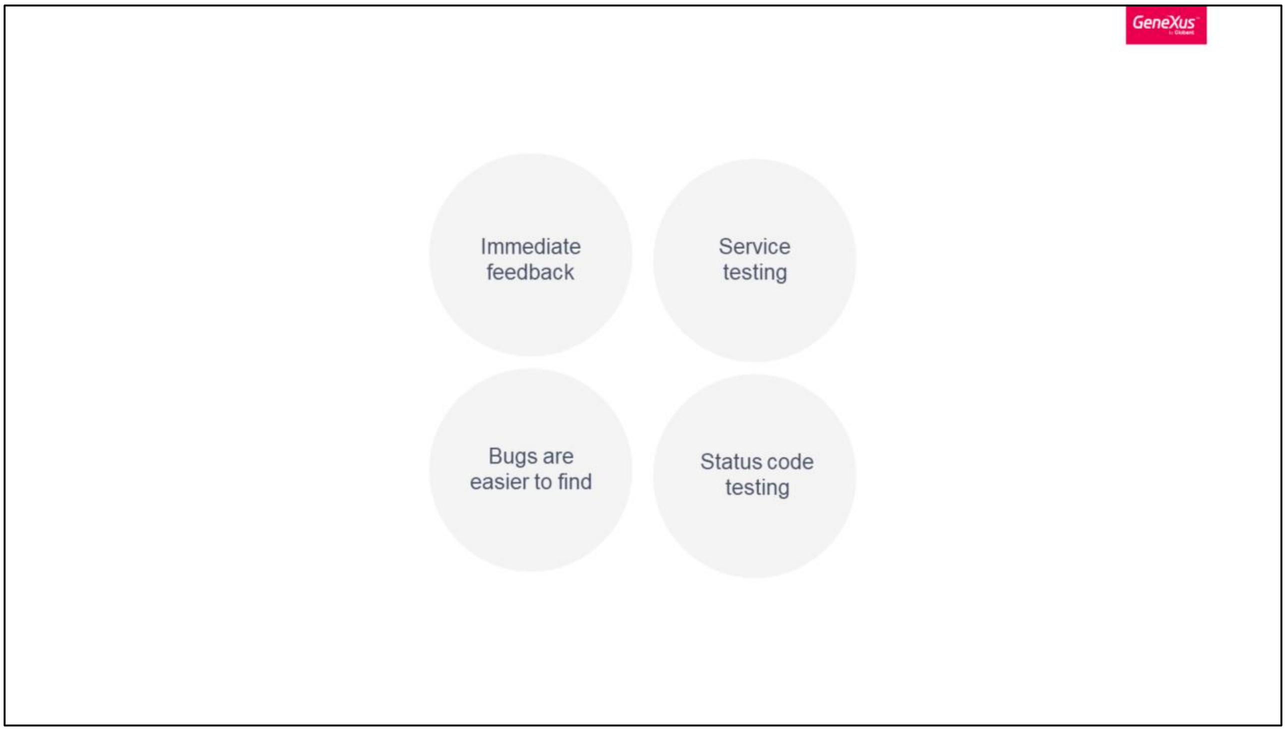
GeneXus™

by Globant



Services Testing

Benefits of Services Testing

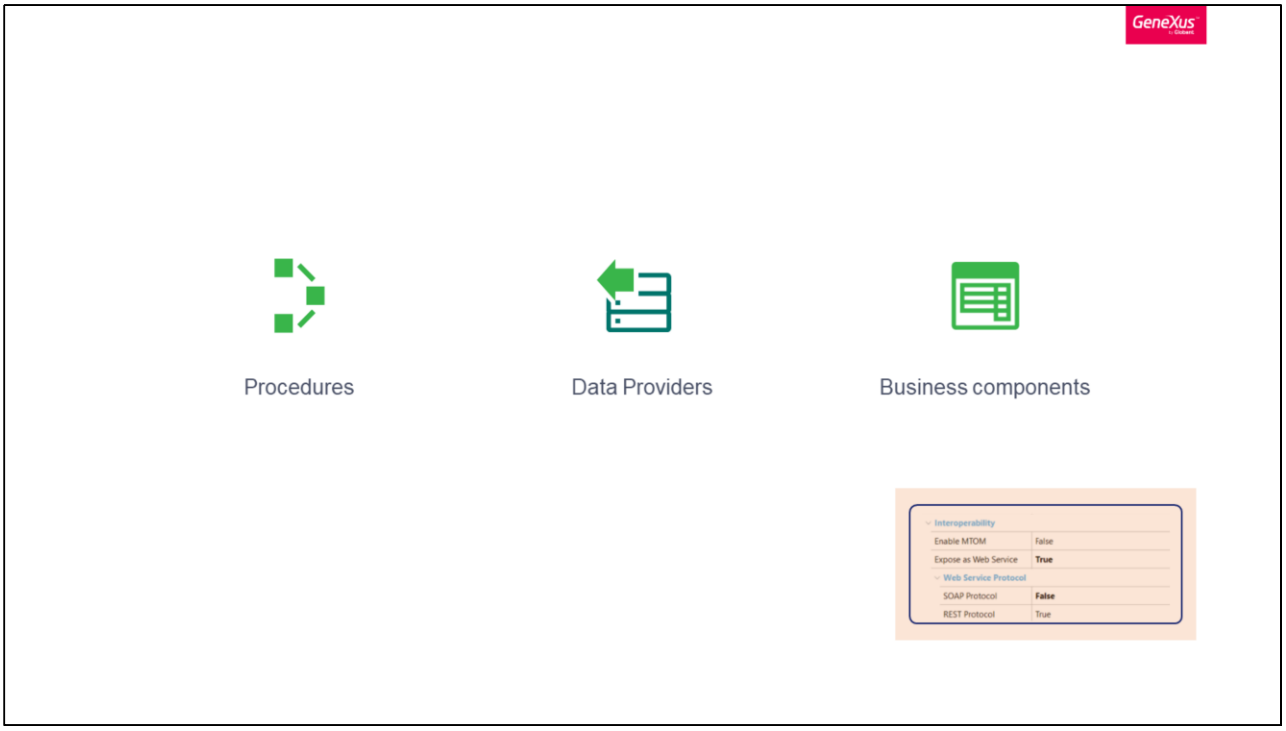


GeneXus applications are strongly service-oriented, and we need to test this logic with external calls. Web Services Testing is a type of software testing that validates Web services.

Web Service Testing is similar to unit testing. You can test a Web service manually or create your own automation code or use the automation tests.

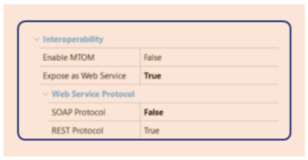
It is a good practice to test this logic with external calls, testing if the service is successfully exposed, and if it returns what we expect, including status codes.

Services testing in GeneXus


Procedures

Data Providers

Business components


Interoperability

Enable MTOM	False
Expose as Web Service	**True**

Web Service Protocol

SOAP Protocol	**False**
REST Protocol	True

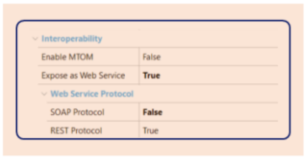
In GeneXus, services are developed using Procedures, Data Providers y Business component objects exposed as web services.

In particular, you can automate and run tests over REST services, enabling wide coverage of core business logic for web systems and mobile applications.

CheckBalanceForTransfer

DoTransfer

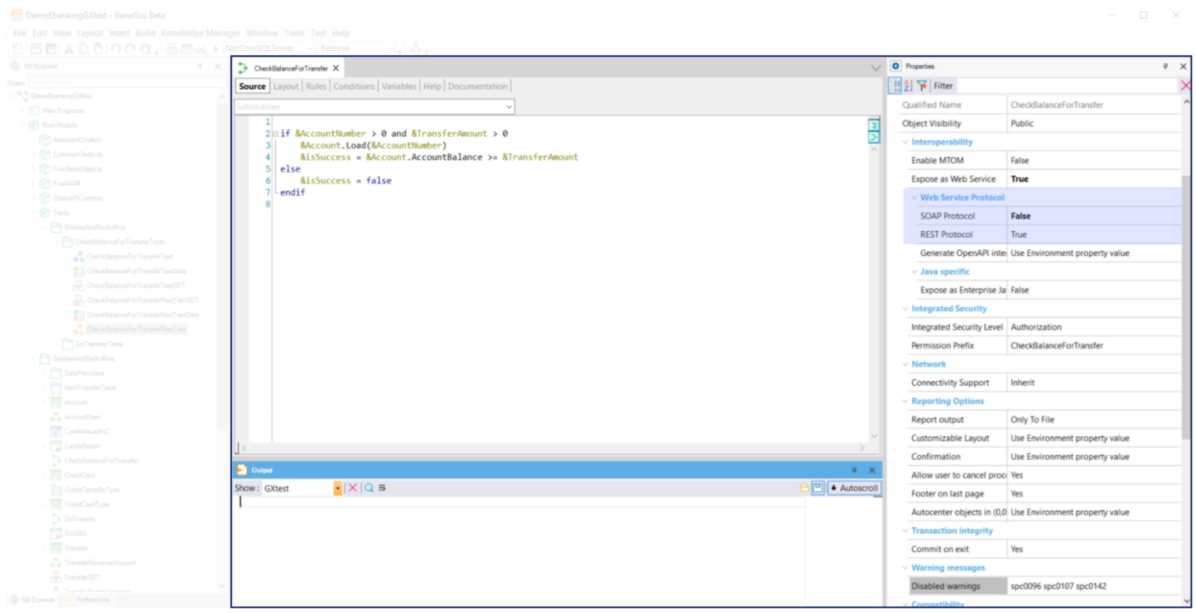
OpenNewAccount



Interoperability	
Enable MTOM	False
Expose as Web Service	**True**
Web Service Protocol	
SOAP Protocol	**False**
REST Protocol	True

Ideally, all objects exposed as REST should be tested with a Rest Test.

Creating the first Rest Test

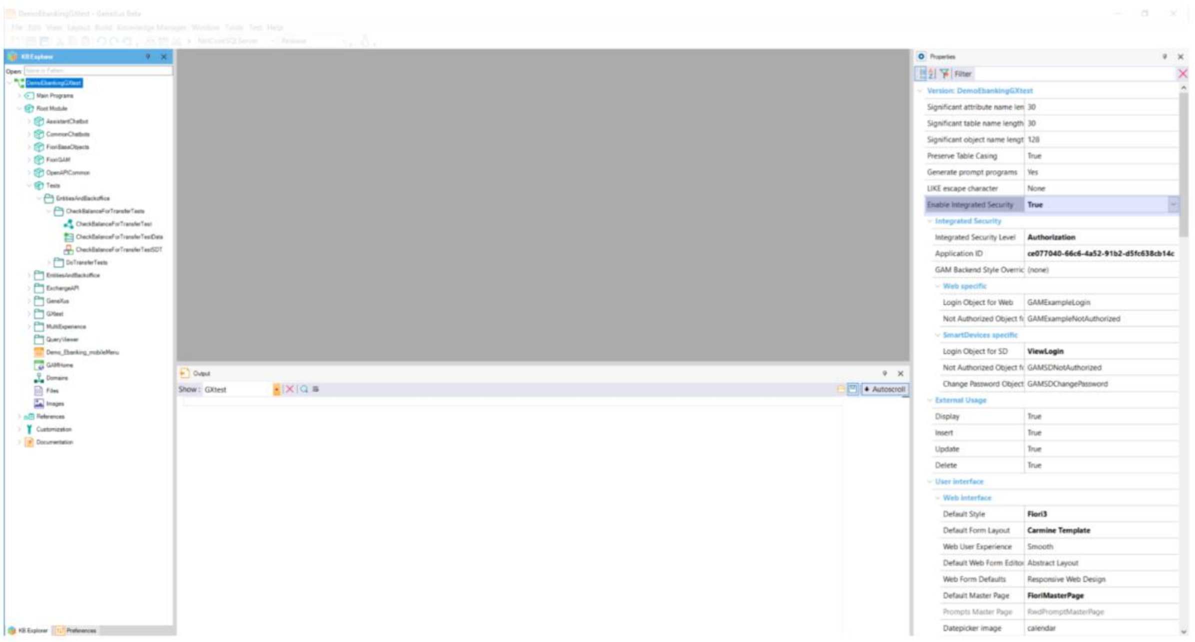


CheckBalanceForTransfer

```
if &AccountNumber > 0 and &TransferAmount > 0
    &Account.Load(&AccountNumber)
    &isSuccess = &Account.AccountBalance >= &TransferAmount
else
    &isSuccess = false
endif
```

Qualified Name	CheckBalanceForTransfer
Object Visibility	Public
Interoperability	
Enable MTOM	False
Expose as Web Service	**True**
Web Service Protocol	
SOAP Protocol	**False**
REST Protocol	True
Generate OpenAPI inte	Use Environment property value
Java specific	
Expose as Enterprise Ja	False
Integrated Security	
Integrated Security Level	Authorization
Permission Prefix	CheckBalanceForTransfer
Network	
Connectivity Support	Inherit
Reporting Options	
Report output	Only To File
Customizable Layout	Use Environment property value
Confirmation	Use Environment property value
Allow user to cancel proc	Yes
Footer on last page	Yes
Autocenter objects in (0,0	Use Environment property value
Transaction integrity	
Commit on exit	Yes
Warning messages	
Disabled warnings	spc0096 spc0107 spc0142

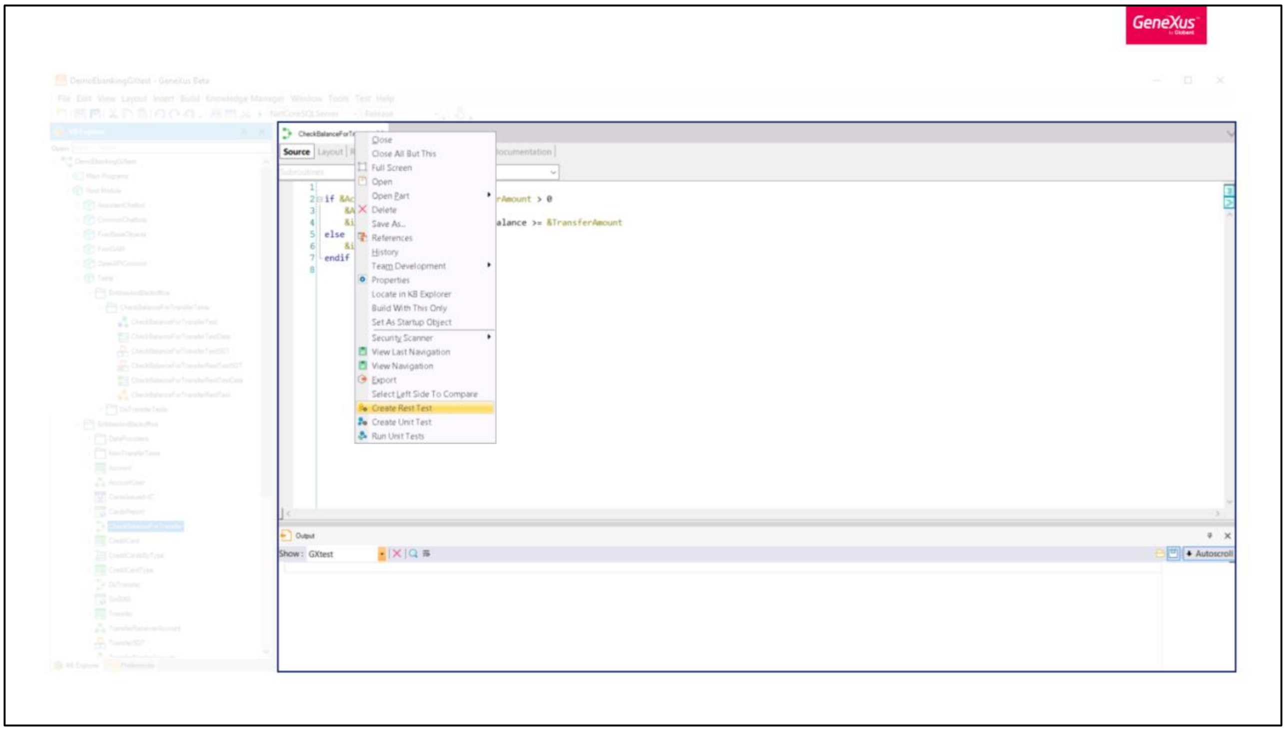
So, as an example, we have exposed as Rest the procedure CheckBalanceForTransfer presented before.



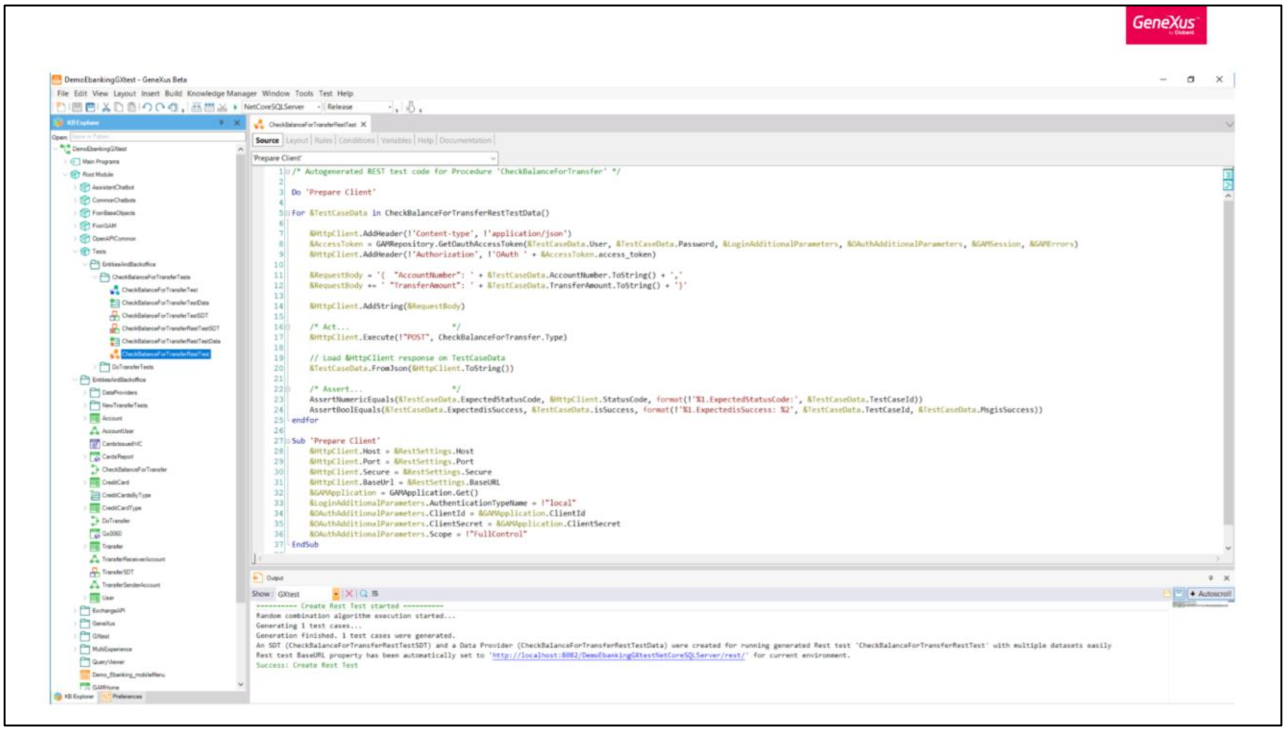
Version: DemoEbankingGXtest

Significant attribute name len	30
Significant table name length	30
Significant object name lengt	128
Preserve Table Casing	True
Generate prompt programs	Yes
LIKE escape character	None
Enable Integrated Security	**True**
Integrated Security	
Integrated Security Level	**Authorization**
Application ID	**ce077040-66c6-4a52-91b2-d5fc638cb14c**
GAM Backend Style Overric	(none)
Web specific	
Login Object for Web	GAMExampleLogin
Not Authorized Object fc	GAMExampleNotAuthorized
SmartDevices specific	
Login Object for SD	**ViewLogin**
Not Authorized Object fc	GAMSDNotAuthorized
Change Password Object	GAMSDChangePassword
External Usage	
Display	True
Insert	True
Update	True
Delete	True
User interface	
Web interface	
Default Style	**Fiori3**
Default Form Layout	**Carmine Template**
Web User Experience	Smooth
Default Web Form Editor	Abstract Layout
Web Form Defaults	Responsive Web Design
Default Master Page	**FioriMasterPage**
Prompts Master Page	RwdPromptMasterPage
Datepicker image	calendar

Notice that this KB has Integrated Security enabled.



Let’s create a Rest Test by right-clicking over the procedure and selecting the option Create Rest Test



```
/* Autogenerated REST test code for Procedure 'CheckBalanceForTransfer' */

Do 'Prepare Client'

For &TestCaseData in CheckBalanceForTransferRestTestData()

    &HttpClient.AddHeader(!'Content-type', !'application/json')
    &AccessToken = GAMRepository.GetOauthAccessToken(&TestCaseData.User, &TestCaseData.Password, &LoginAdditionalParameters, &OAuthAdditionalParameters, &GAMSession, &GAMErrors)
    &HttpClient.AddHeader(!'Authorization', !'OAuth ' + &AccessToken.access_token)

    &RequestBody = '{  "AccountNumber": ' + &TestCaseData.AccountNumber.ToString() + ','
    &RequestBody += ' "TransferAmount": ' + &TestCaseData.TransferAmount.ToString() + '}'

    &HttpClient.AddString(&RequestBody)

    /* Act...                    */
    &HttpClient.Execute(!"POST", CheckBalanceForTransfer.Type)

    // Load &HttpClient response on TestCaseData
    &TestCaseData.FromJson(&HttpClient.ToString())

    /* Assert...                 */
    AssertNumericEquals(&TestCaseData.ExpectedStatusCode, &HttpClient.StatusCode, format(!'%1.ExpectedStatusCode:', &TestCaseData.TestCaseId))
    AssertBoolEquals(&TestCaseData.ExpectedisSuccess, &TestCaseData.isSuccess, format(!'%1.ExpectedisSuccess: %2', &TestCaseData.TestCaseId, &TestCaseData.MsgisSuccess))
endfor

Sub 'Prepare Client'
    &HttpClient.Host = &RestSettings.Host
    &HttpClient.Port = &RestSettings.Port
    &HttpClient.Secure = &RestSettings.Secure
    &HttpClient.BaseUrl = &RestSettings.BaseURL
    &GAMApplication = GAMApplication.Get()
    &LoginAdditionalParameters.AuthenticationTypeName = !"local"
    &OAuthAdditionalParameters.ClientId = &GAMApplication.ClientId
    &OAuthAdditionalParameters.ClientSecret = &GAMApplication.ClientSecret
    &OAuthAdditionalParameters.Scope = !"FullControl"
EndSub
```

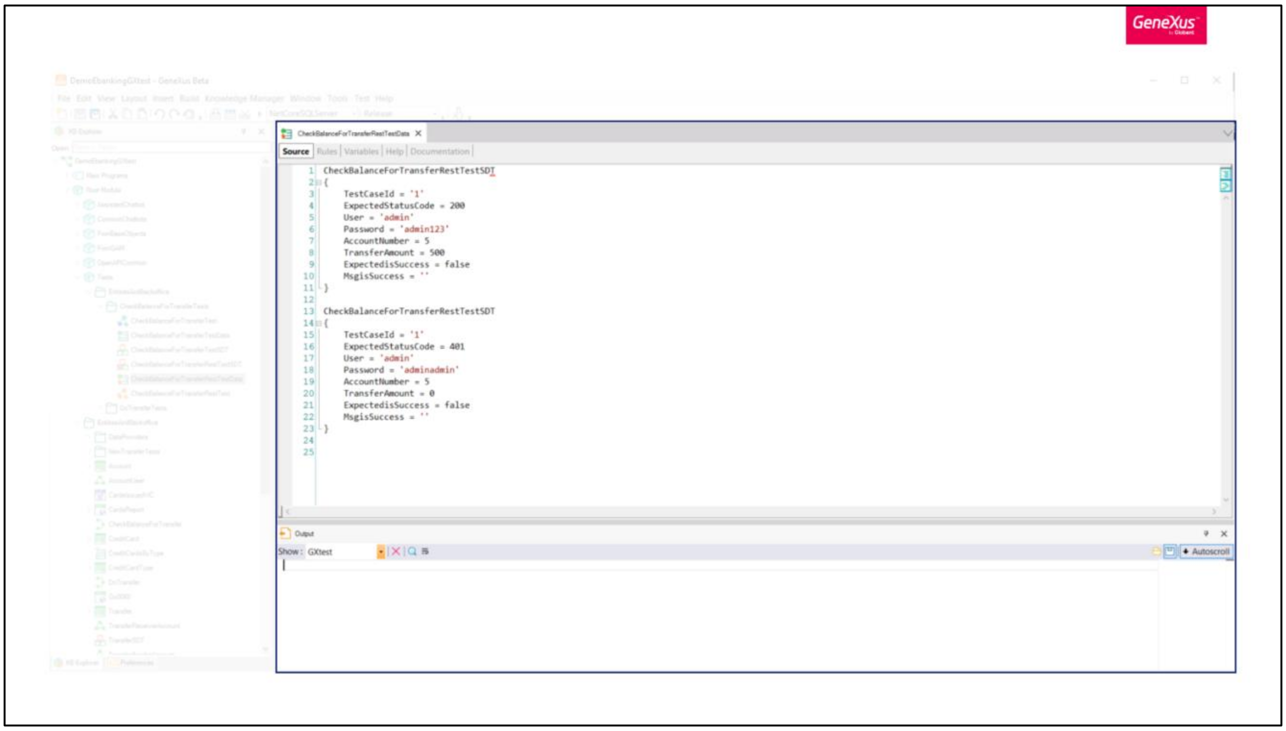
```
---------- Create Rest Test started ----------
Random combination algorithm execution started...
Generating 1 test cases...
Generation finished. 1 test cases were generated.
An SDT (CheckBalanceForTransferRestTestSDT) and a Data Provider (CheckBalanceForTransferRestTestData) were created for running generated Rest test 'CheckBalanceForTransferRestTest' with multiple datasets easily
Rest test BaseURL property has been automatically set to 'http://localhost:8082/DemoEbankingGXtestNetCoreSQLServer/rest/' for current environment.
Success: Create Rest Test
```

Within a few seconds, you can see that a Rest Test is generated with a template that eases the creation and set up of the test.

In this example the authentication management is included in the Rest Test because the application has GAM security enabled.

In the ‘Prepare Client’ subroutine basic initialization for HttpClient and authentication configuration is performed.

Then when iterating through test data, an authentication token is obtained for the current user and password, and included in the request when invoking the REST service.

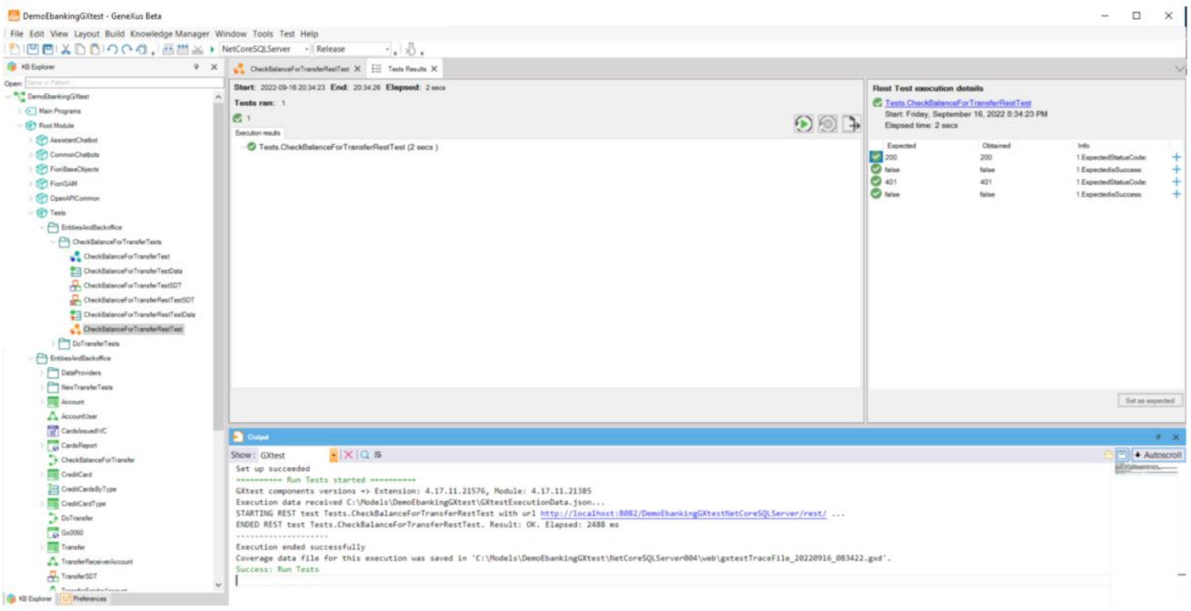


```
CheckBalanceForTransferRestTestSDT
{
    TestCaseId = '1'
    ExpectedStatusCode = 200
    User = 'admin'
    Password = 'admin123'
    AccountNumber = 5
    TransferAmount = 500
    ExpectedisSuccess = false
    MsgisSuccess = ''
}

CheckBalanceForTransferRestTestSDT
{
    TestCaseId = '1'
    ExpectedStatusCode = 401
    User = 'admin'
    Password = 'adminadmin'
    AccountNumber = 5
    TransferAmount = 0
    ExpectedisSuccess = false
    MsgisSuccess = ''
}
```

In this case we included a test case with valid credentials and a test case with invalid ones.

Note that the ExpectedStatusCode field is used to validate against the actual status code returned by the service. We suggest to validate the different possibilities of status codes in your test, in this example we test the 200 and 401 status codes.



Start: 2022-09-16 20:34:23 End: 20:34:26 Elapsed: 2 secs
Tests ran: 1
✓ 1
Execution results
✓ Tests.CheckBalanceForTransferRestTest (2 secs)

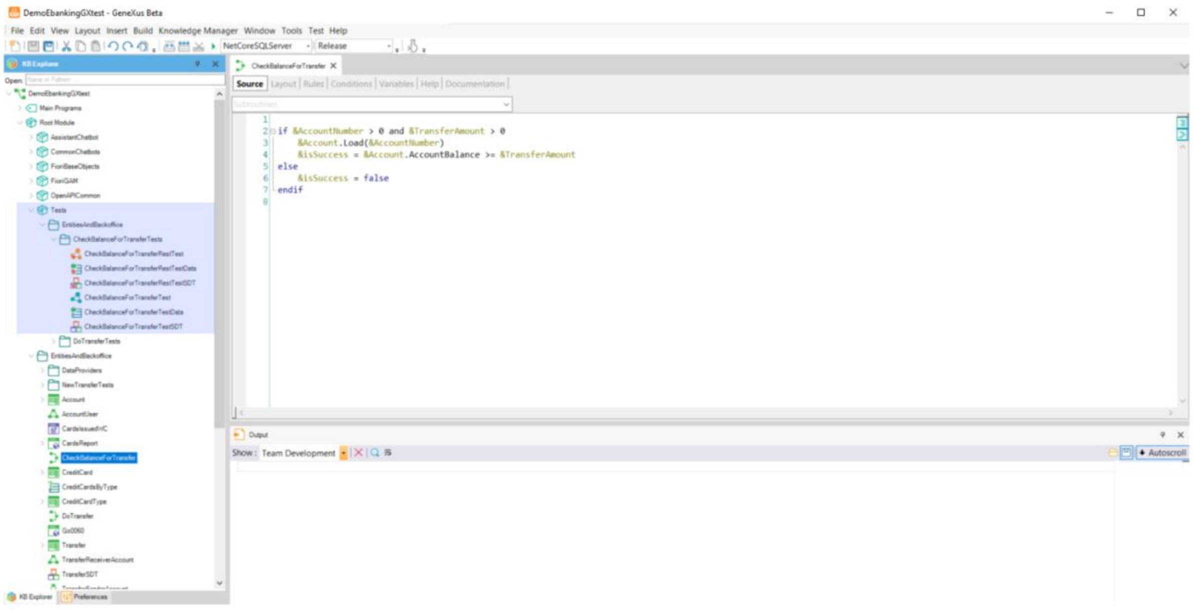
Rest Test execution details
✓ Tests.CheckBalanceForTransferRestTest
Start: Friday, September 16, 2022 8:34:23 PM
Elapsed time: 2 secs

	Expected	Obtained	Info
✓	200	200	1.ExpectedStatusCode:
✓	false	false	1.ExpectedisSuccess:
✓	401	401	1.ExpectedStatusCode:
✓	false	false	1.ExpectedisSuccess:

```
Set up succeeded
========== Run Tests started ==========
GXtest components versions => Extension: 4.17.11.21576, Module: 4.17.11.21385
Execution data received C:\Models\DemoEbankingGXtest\GXtestExecutionData.json...
STARTING REST test Tests.CheckBalanceForTransferRestTest with url http://localhost:8082/DemoEbankingGXtestNetCoreSQLServer/rest/ ...
ENDED REST test Tests.CheckBalanceForTransferRestTest. Result: OK. Elapsed: 2488 ms
--------------------
Execution ended successfully
Coverage data file for this execution was saved in 'C:\Models\DemoEbankingGXtest\NetCoreSQLServer004\web\gxtestTraceFile_20220916_083422.gxd'.
Success: Run Tests
```

When we run the Rest Test we can see in the Tests Results window the results of the execution. They are similar to unit tests, as they work based on assertions too.

Remember to run the application with your service before running the test, otherwise it will fail.



```
if &AccountNumber > 0 and &TransferAmount > 0
    &Account.Load(&AccountNumber)
    &isSuccess = &Account.AccountBalance >= &TransferAmount
else
    &isSuccess = false
endif
```

Note that we can test a procedure in different ways, internally and externally.

For example, we could test different combinations through internal calls with unit tests, and only some cases by external calls, because it is slower and depends on the server status (requires the deployment of the services).

GeneXus™ by Globant

training.genexus.com
wiki.genexus.com