

GeneXus[™]
by **Globant**

MOBILE

Nicolas Adrién



GeneXus™

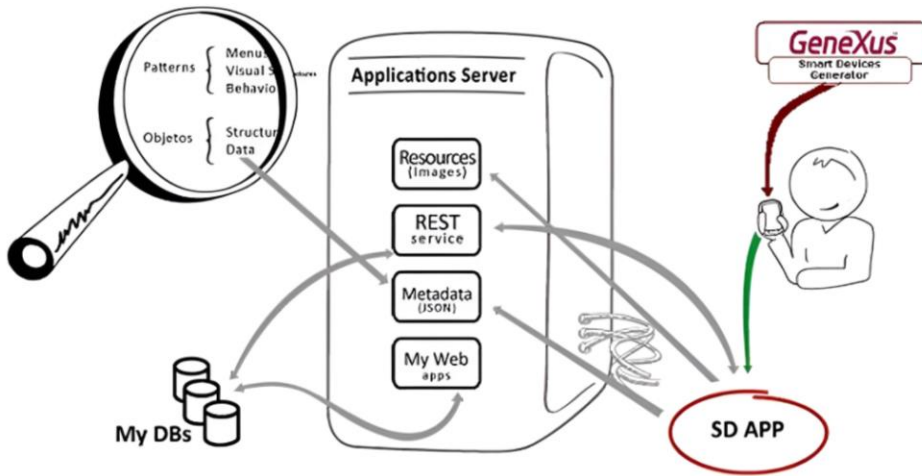
MOBILE

Authentication and Access Control in Mobile Applications

GeneXus[™]

In this video, we will discuss authentication and access control in GeneXus mobile applications using GAM.

Native Mobile Authentication



Native mobile applications installed on devices consume REST services from the Web server. These applications create a user interface that is also based on resources in addition to metadata.

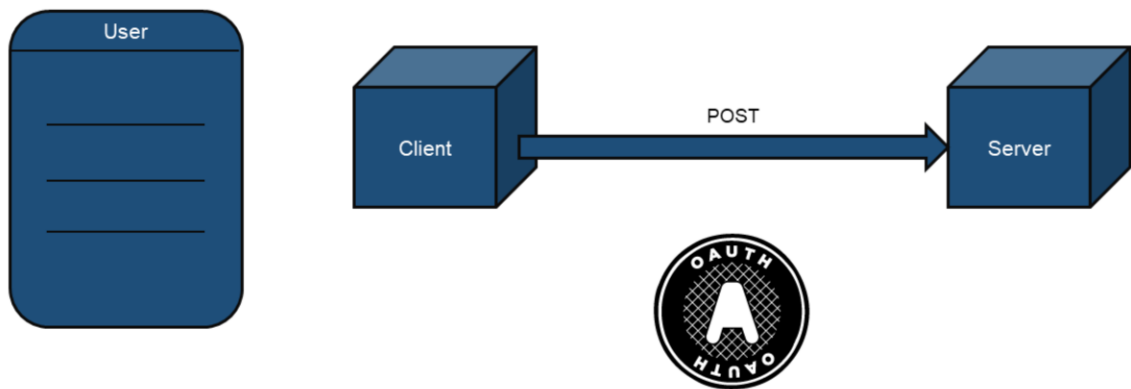
Given this architecture, it is important to have a security mechanism in place to prevent these types of actions from being performed by unauthorized users. The way to do this, of course, is by using GAM.

As in web applications, security is automatically added to the mobile application through the Enable built-in security property. This security can be implemented through Authentication or Authorization.

Let's look at it in more detail.

Authentication

Authentication Types - Local



As in web applications, a possible scenario in native mobile applications is that all native mobile objects are private.

By configuring our application with built-in security level to Authentication, we will make all objects require the user to be logged in.

Now let's see how the authentication types behave and work in mobile applications. Let's start with the Local type.

This authentication type works just like web applications, where user credentials are stored in GAM's "User" table.

As we have already mentioned, the password is not stored. Instead, a password hash is stored using a unique key for each user and the SHA-512 algorithm.

Then, a POST request is simply made from the client to the server, where in this case the client corresponds to the mobile device.

In the low-level background, this and all GAM authentication methods use the OAuth 2.0 protocol.

Authentication

Authentication Types - Local

The screenshot shows a POST request to the endpoint `/oauth/access_token`. The request body is form-data with the following parameters:

KEY	VALUE
<input checked="" type="checkbox"/> client_id	d3b208f72cb84607ab88c4647b898728
<input checked="" type="checkbox"/> client_secret	fe8e07052da449db94af48f394043407
<input checked="" type="checkbox"/> grant_type	GAMLocal
<input checked="" type="checkbox"/> scope	FullControl
<input checked="" type="checkbox"/> username	GAM
<input checked="" type="checkbox"/> password	123

The response body is JSON, showing the following data:

```

1 {
2   "access_token": "85a3006c-0606-4102-980e-223f88463ec21111c4cc59529f93f1c05286d108ae044a14b5e0775d4688ad5059ba375d888602c77fe62f1f7b",
3   "token_type": "Bearer",
4   "expires_in": 3600,
5   "refresh_token": "0011040ymct3v1a59b3U3wGfjmuSan1bwGp3",
6   "scope": "FullControl",
7   "user_guid": "c3910a62-5960-4495-a29e-67867784cfc3"
8 }

```

Here we can see an example of the POST request we mentioned.

The request is made to our application URL, followed by `/oauth/access_token`. As parameters, we include the typical `client_id` and `client_secret`, `grant_type` and `scope`, and finally the `username` and `password`.

If the request is successful, we should receive a JSON like the following, with all the access data and user identifier.

This can be useful for testing your own authentication types, using any API testing tool.

Login

How to

```
Event 'BtnLogin'  
  Composite  
    GeneXus.Common.UI.Progress.ShowWithTitle("Connecting...")  
    GeneXus.SD.Actions.Login(&UserName, &Password)  
    GeneXus.Common.UI.Progress.Hide()  
    Return  
  EndComposite  
EndEvent
```

&LoginExternalAdditionalParameters

In previous topics, we reviewed the Login mechanism for web applications. Now let's see how this works in mobile applications.

To do so, we will focus on the GAMSDLogin object, which is included in the reference implementations provided by GAM.

In this case, local login is done through the External Object GeneXus.SD.Actions and its login method.

In the background, once the session is started, the information is stored in the device, but in encrypted form, saving a security token using the KeyStore (in the case of Android) or Keychain (in the case of iOS).

This Login method that belongs to Actions is overloaded, so it allows including the additional parameter &LoginExternalAdditionalParameters as we just saw in the external Login cases.

Let's address this with another example in detail.

Login

How to

Name	Type	Description	Is Collection
LoginExternalAdditionalParameters		Login External Additional Parameters	<input checked="" type="checkbox"/>
Repository	Character(40)	Repository	<input type="checkbox"/>
AuthenticationTypeName	Character(60)	Authentication Type Name	<input type="checkbox"/>
Properties		Properties	<input checked="" type="checkbox"/>
Property			
Id	Character(40)	Id	<input type="checkbox"/>
Value	Character(40)	Value	<input type="checkbox"/>

```

Event 'BtnLogin'
  Composite
    GeneXus.Common.UI.Progress.ShowWithTitle("Connecting...")
    &LoginExternalAdditionalParameters.Repository = !"1f41a6bb-bc52-451b-b521-c4bcd4a9ac8b"
    GeneXus.SD.Actions.Login(&UserName, &Password, &LoginExternalAdditionalParameters)
    GeneXus.Common.UI.Progress.Hide()
  Return
EndComposite
EndEvent

```

That parameter we mentioned must correspond to the object called in the same way, which looks as follows.

Following is an example of use of this overload in the SD Login method. The LoginExternalAdditionalParameters object allows establishing the GUID of the Repository we want to connect to. This is useful in MultiTenant applications.

When we set a value for the Repository property of the &LoginExternalAdditionalParameters parameter, the connection to that Tenant (in particular, to that client) can be established, represented by a Repository and its GUID.

In addition to this option that we use, we have others that can be the Name of the authentication type, and a list of dynamic properties Id and Value that the application might need.

Login

How to

The screenshot shows a REST client interface for a POST request to `/oauth/access_token`. The request body is selected, and the format is set to `x-www-form-urlencoded`. The parameters are as follows:

Parameter	Value
<input checked="" type="checkbox"/> client_secret	fe8e07052da449db94af48f394043407
<input checked="" type="checkbox"/> grant_type	GAMlocal
<input checked="" type="checkbox"/> scope	FullControl
<input checked="" type="checkbox"/> username	GAM
<input checked="" type="checkbox"/> password	123
<input checked="" type="checkbox"/> additional_parameters	<pre>{ "AuthenticationTypeName": "local", "Repository": "", "Properties": [{ "Id": "Company", "Value": "GeneXus" }, { "Id": "Branch", "Value": "12" }] }</pre>
Key	Value

Going back to manual tests, we can replicate the login request that we saw before, but this time adding these additional parameters that we mentioned before.

To do so, we add the `additional_parameters` parameter to the request, and in it we include the data we want to add to the request.

In this case, note that the name of the authentication type we want to use is included, in addition to two properties that we mentioned before (`Id`, `Value`); it is the name of a company and a branch.

GeneXus

Authentication

Authentication Types - Google

Google authentication type

```

Event 'Google'
Composite
  &LoginExternalAdditionalParameters = new()
  &LoginExternalAdditionalParameters.AuthenticationTypeName = !"googleb"
  GeneXus.SD.Actions.LoginExternal(GAMAuthenticationTypes.Google, &UserName, &Password, &LoginExternalAdditionalParameters)
Return
EndComposite
EndEvent

```

<p>Enabled? <input type="checkbox"/></p> <p>Description <input type="text"/></p> <p>Small image name <input type="text" value="GAMButtonGoogleSmall"/></p> <p>Big image name <input type="text"/></p> <p>Impersonate <input type="text" value="(none)"/></p>	<p>Local site URL <input type="text" value="https://trialapps3.genexus.com"/></p> <p>Additional Scope <input type="text"/></p>
--	--

As we already saw in the Authentication topic, we can use Google as an Identity provider for login.

Let's see a configuration for this.

In Google's website, we need to create a client application and get the client's information and secret. The URL of this site is as follows.

First, we go to the API and Services section and click on the Credentials section, selecting "OAuth Client ID."

Then we select "Web Application" in the Application Type.

Once this is done, we need to change the redirection URIs. There we can specify the full URI of our application, including /oauth/gam/signin, as shown in the image. The latter is important and applies regardless of whether our application is Java or .NET.

The full URL of the application must always be specified, including the virtual directory, followed by /oauth/gam/signin.

After confirming this, we will get our client information and finish on this side.

Now let's configure our application from the GAM back end.

For that, we create a new Google authentication type and complete the information with that provided by Google.

In this step, it should be highlighted that in the Local site URL field we only need to enter the domain of the server running the application. It is not necessary to enter the

complete URL of the site, but if you do so, do not include "/servlet" in Java.

At this point, we already have the Google and back-end side configured. But since this is a mobile application, we need a different logic in the application Login so we have to add an event in it that authenticates with Google.

This logic now uses the LoginExternal method of the External Object Actions, and as first parameter it is indicated that it is a Google authentication type through the GAMAuthenticationTypes domain.

Then the username and password will be ignored. Lastly, we have LoginExternalAdditionalParameters, with which we can provide additional information to the login that is useful to us. In this case, we define the name of the authentication type.

This is useful if we have more than one type of Google authentication in the repository, and with this name we will be able to select the one we want. Otherwise, we can make the login call without using this parameter.

Later on, we will see this attribute in more detail.

This is the end of the authentication process with Google.

Authentication



Authentication Types - Facebook

```
Event 'Facebook'  
  Composite  
    &LoginExternalAdditionalParameters = new()  
    &LoginExternalAdditionalParameters.AuthenticationTypeName = !"facebook"  
    GeneXus.SD.Actions.LoginExternal(GAMAAuthenticationTypes.Facebook, &UserName, &Password, &LoginExternalAdditionalParameters)  
    Return  
  EndComposite  
EndEvent
```

Now let's look at the case of Facebook.

In the introductory course, we showed one type of authentication with Facebook, so we will not repeat this. You can see how to do this in that course or on the GeneXus Wiki. As in the case of Google, once we have created and configured our client and authentication type in the GAM backend, we need custom logic in the application.

In this case, once again we add a new event with the following. The explanation is the same as for Google, with the difference that now in the first LoginExternal parameter we indicate that the type is Facebook.

Authorization



<prefix>_Execute



<prefix>_Services_Execute

<prefix>_Services_Insert
 <prefix>_Services_Update
 <prefix>_Services_Delete

<prefix>_Services.FullControl

Authorization.

In the case of SD panels, GAM checks if the user has permission to execute them. For that it verifies that the user has the <prefix>_Execute permission, where prefix is the Permission Prefix defined for the object.

In the event that a permission error is found, it will be displayed on the screen or a redirection to the object specified as not authorized for the SD property will be made.

In SD panels and WorkWith, the business logic is resolved using REST Web Services. When these objects are generated, data providers exposed as REST web services are also generated.

These web services become private when security is set on the objects mentioned above.

Something to keep in mind is that actions in WorkWithSD panels are directly related to the Business Component associated with the WorkWith. This means that if the WorkWith pattern is applied to a Transaction, it is automatically saved as a Business Component exposed as a REST web service.

To control permissions on insert, update, and delete actions, permissions must be specified on the Business Component itself.

Since the idea is to validate if the user has rights to execute an object depending on

the method used to make the call, the GET method requires the <prefix>_Services_Execute permission (in this case, prefix is the permission prefix defined for the Business Component), and this is the minimum permission required.

The rest of the permissions are as follows:

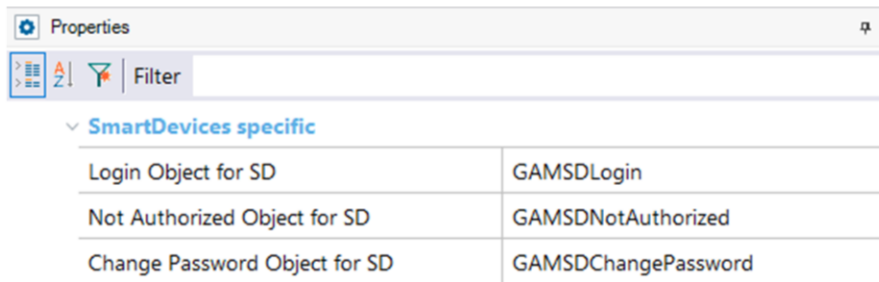
<prefix>_Services_Insert

<prefix>_Services_Update

<prefix>_Services_Delete

Here, as usual, we also have the FullControl that covers all permissions.

Object configuration



The screenshot shows the 'Properties' window in GeneXus. It has a 'Filter' field and a section titled 'SmartDevices specific' which contains a table with three rows.

SmartDevices specific	
Login Object for SD	GAMSDLogin
Not Authorized Object for SD	GAMSDNotAuthorized
Change Password Object for SD	GAMSDChangePassword

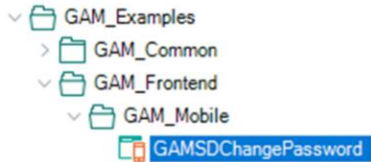
Earlier we said that users could be redirected to the unauthorized object if they did not have permission to access a resource. This object can be configured at the KB version level, under the SmartDevices specific section.

Not only can we configure it, but from there we should also define which objects will be associated with the login and password change.

Let's see a particular feature of this last one.

Change Password

GAMSDChangePassword



A screenshot of a mobile application interface titled 'Change Password'. At the top left is a back arrow and the title 'Change Password', and at the top right is the word 'BACK'. Below the title bar is a text input field labeled 'Email or Name' with the value 'admin'. There are three password input fields: 'Current password' (with 'Current password' as placeholder text), 'New password' (with 'New password' as placeholder text), and 'Confirm password' (with 'Confirm new password' as placeholder text). Each password field has a small eye icon to its right for toggling visibility. At the bottom of the form is a large blue button labeled 'CHANGE PASSWORD'.

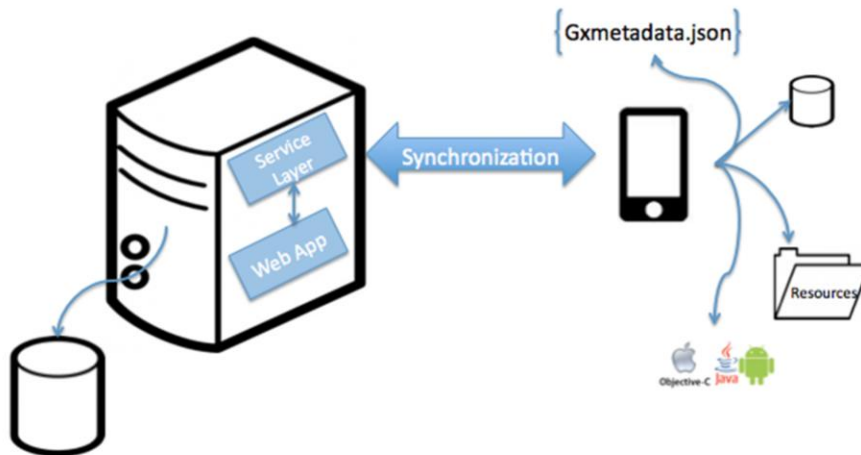
As in web applications, when a user's password expires or needs to be changed, he/she is automatically redirected to the change password object.

For Mobile in particular, we have the GAMSDChangePassword object within the example objects already incorporated by GAM, and it can be used for this purpose.

As we said before, when users need to change their password, they will not be able to do anything else until they change it since the applications will always redirect them to the object that was configured to change the password.

Online and Offline Apps

Offline App



With GeneXus, we can have Online and Offline applications. Let's see the differences between them, together with their objects and methods.

Let's start with the Offline ones.

The architecture of these mobile applications has to consider two situations in which these applications should work: when the application is offline and when it is online.

In the first case, the application must be able to process data and interact with a database without internet access.

In the second case, the application can synchronize data with the server by invoking web services.

Regardless if the application is connected or not, its processing is performed on the device by updating its local database. Once a connection is established, the application will synchronize the changes that were made while it was offline.

Such a mobile application can be divided into two components: a local component and a server component.

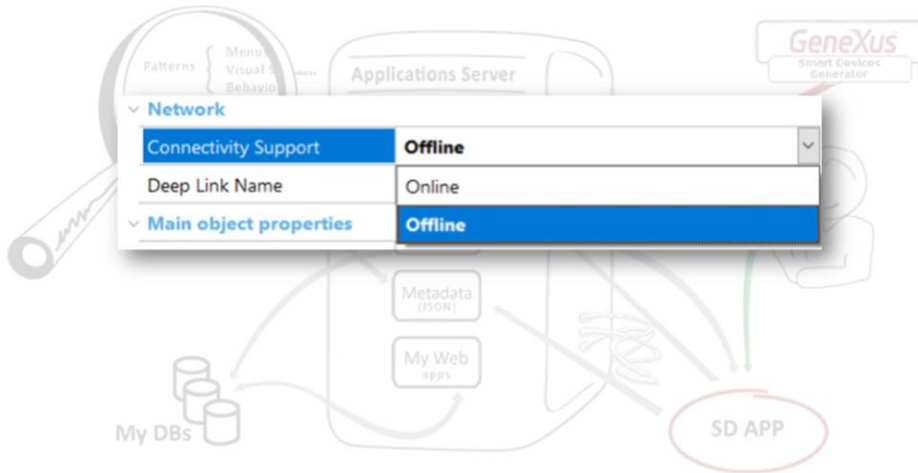
The server can have a back end for the application, the database, and a service layer for synchronizing data with the devices.

In the devices, the applications also have a database with a subset of the server data and all the logic to be executed locally. The application also has all the metadata needed for the user interface design and user events.

Both components communicate through REST services to perform the necessary synchronization to populate the local database and send the modifications made on the device to the server.

Online and Offline Apps

Online App



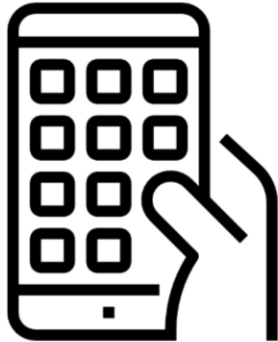
Now let's look at online applications, and to do so we go back to the image we saw at the beginning.

The data that is required by the application and resides on the device will be obtained by consuming REST services, which will access the user's database and return the requested information to the application.

To determine whether an application will be Online or Offline, the Connectivity Support property must be configured in the Main object with the value we want. As we can see, this property is located under the Network item.

If the application is Offline, an automatic permissions check is not performed on the device; this only happens in Online applications because they access a Rest service on the Server.

App with anonymous user



Custom Authentication

External Web Services Authentication

Local Authentication

Application with anonymous user. What does it mean?

This is a concept in mobile application development with GAM security that allows giving users the option to use the application with the same or similar functionalities that are available to registered users.

In the background, GAM automatically registers the anonymous user on each device.

In GeneXus 15 U10 or prior versions, this only happens with Custom, External Web Services, and Local authentication types. As from GeneXus 15 U11, it is valid for any type of authentication.

Let's see how it works.

App with anonymous user

How it works



Each device has an identifier for it, so when a user starts using the application, it automatically creates a GAM "user" with that identifier and asks for a token.

The created user has the following characteristics:

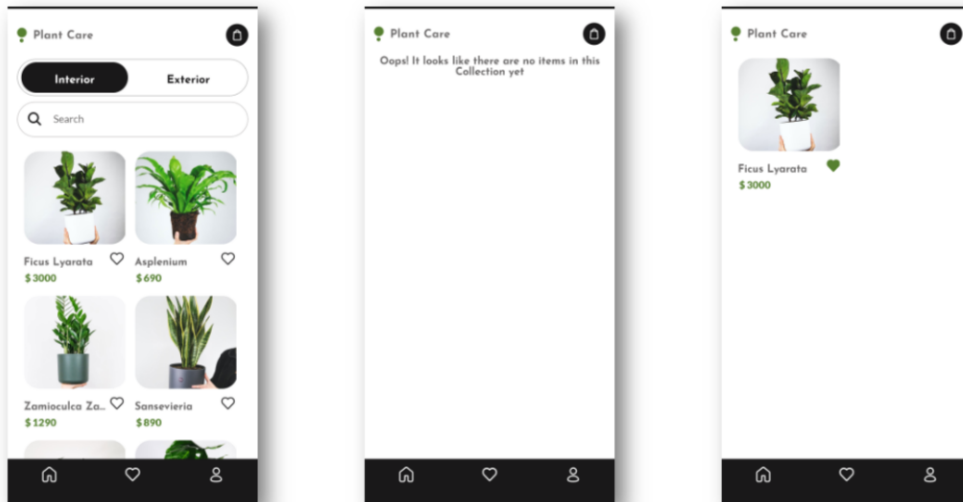
- It is seen by the application as any other user but has the property `GAMUser.isAnonymous = True`.
- No personal data of the person operating the device is known, requested, or stored, so it is completely anonymous.
- The session never expires, regardless of the time limit of the OAuth token that has been defined for the sessions.

If the user decides to register with the application, when using the `&GAMUser.Save()` method the new user will be created with his/her registration data and will be assigned the same identifier as the automatically registered user.

This allows any information related to the automatically registered user that has been saved by the application to remain associated with the new registered user, without having to make changes to the user-data relationships that have been previously saved.

App with anonymous user

How it works: Example



Let's see an example to understand this fully.

Suppose we have an application for selling plants.

This application allows registering and logging in with an account. In addition, we can bookmark our favorite plants to purchase them later, for example.

Ideally, if I have not logged into the application yet and I select a plant as a favorite, when I register my selection will still be there.

This is what GAM's anonymous user functionality allows us to do.

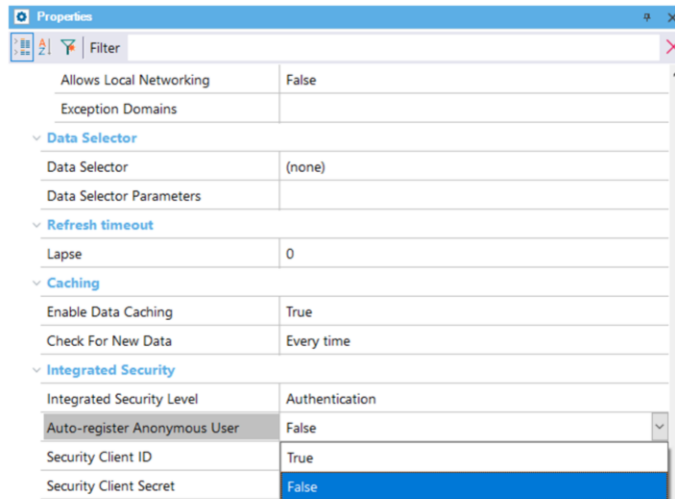
If we activate it, users will be able to bookmark all their favorites without worrying if they are logged into the application or not, because if they haven't done so, their selection will be automatically transferred to their account when they register.

The same could be applied to the shopping cart, for example, by keeping the selected items in the cart when the person registers to check out.

Note that if the user was already registered his/her bookmarks are not transferred. The reason is that a login takes place, not the creation of a new user. This only works with registration.

App with anonymous user

How to use it



This property is available for Mobile objects that are Main (for example, the Panel), when the integrated security is enabled.

This property value determines the application's behavior when an anonymous user tries to execute the first object that has the Integrated Security Level property set to Authentication or Authorization.

As we see in the image, the possible values for the auto-registration property are True or False.

If we select False and we have set the Integrated Security Level to Authentication or Authorization, the anonymous user will be shown a dialog indicating that he/she needs to log in or register.

If True is selected, instead of prompting for login or registration, the application will automatically create a user based on the user's mobile device information.

App with anonymous user

How to identify auto-registered users

```
If &GAMSession.User.IsAutoRegisteredUser
    //Anonymous User
else
    //Registered User
EndIf
```

```
GAMUser.isAnonymous()
```

```
&GAMUser = GAMUser.Get()
&GAMUser.IsAutoRegisteredUser
```

These automatically registered users can be identified in the following ways.

One option is through the GAM session user, asking directly if it is an auto-registered user.

Another similar way is to return a Boolean indicating if the current user is anonymous or not.

The last option is as follows, where first the GAM user is obtained and then it is checked if it is an auto-registered user.

GeneXus[™]
by **Globant**

training.genexus.com
wiki.genexus.com