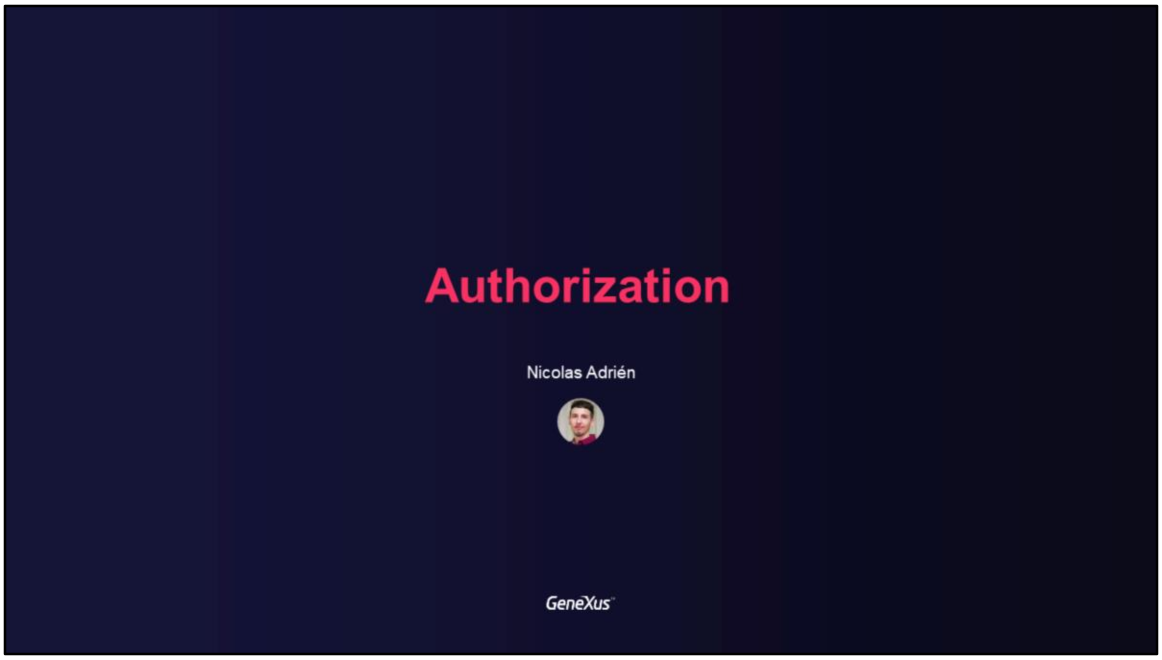
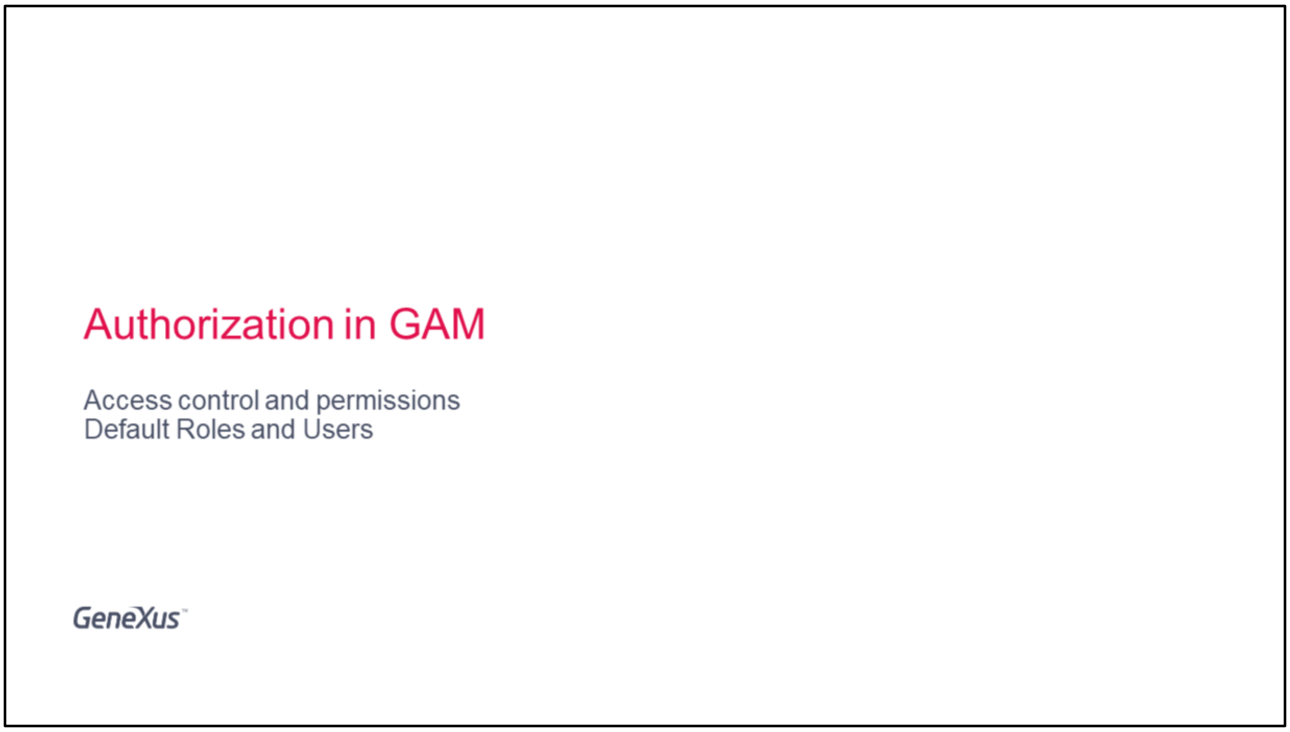
GeneXus™ by Globant



Authorization

Nicolas Adrién

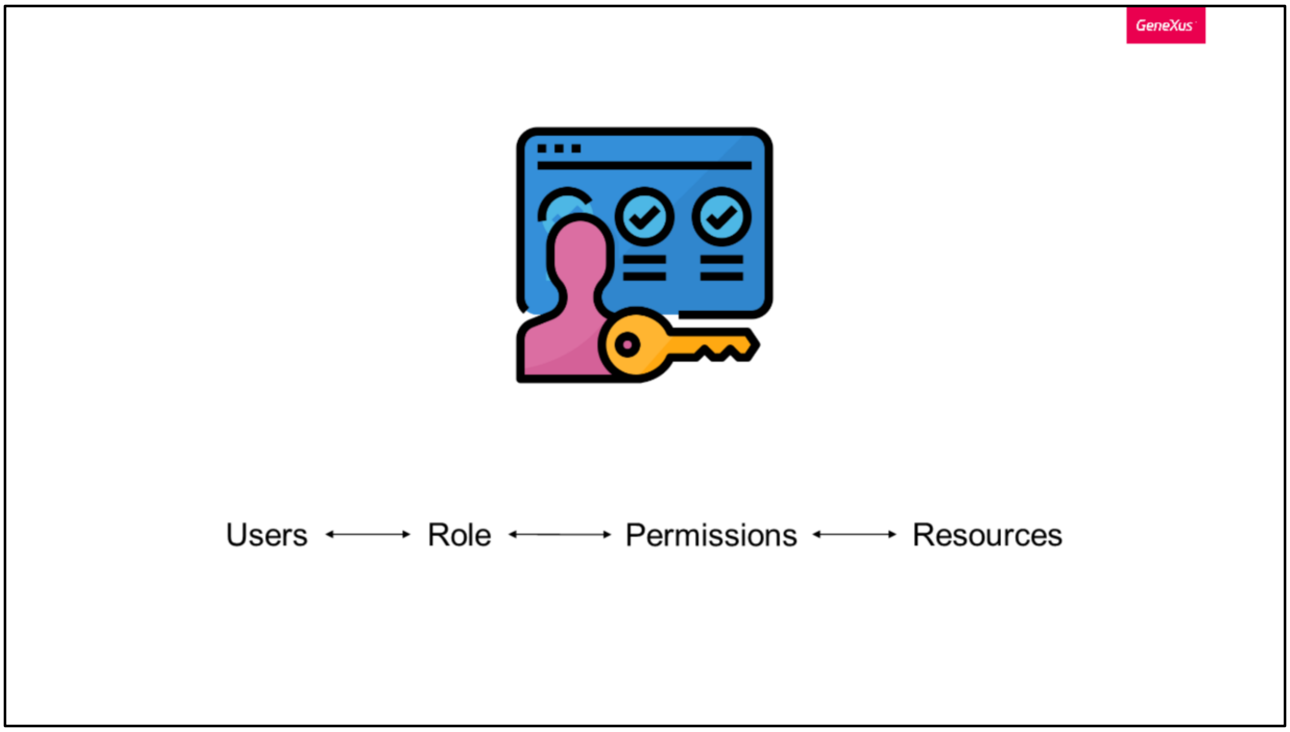


Authorization in GAM

Access control and permissions
Default Roles and Users

GeneXus™

In this video, we will cover topics related to Authorization in GAM, Access control and permissions, and default roles and users.



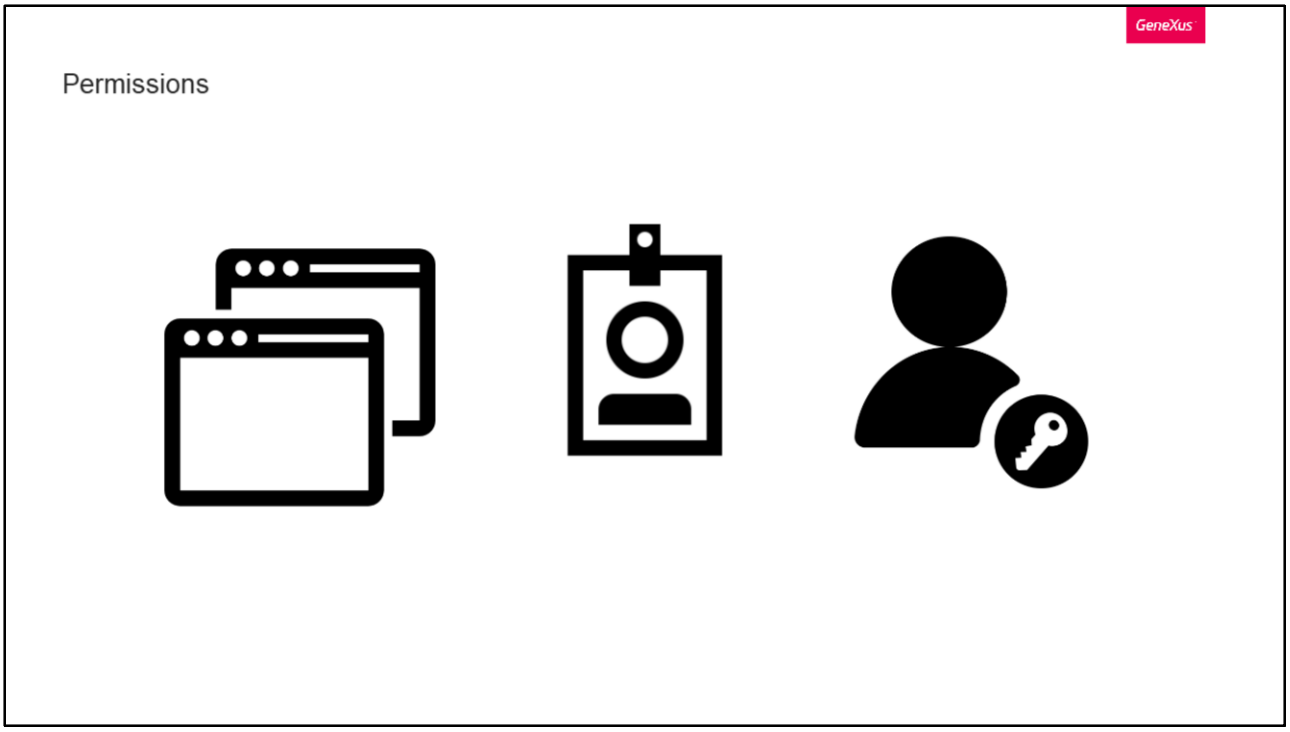
$$\begin{tikzcd}\text{Users} \arrow[r, leftrightarrow] & \text{Role} \arrow[r, leftrightarrow] & \text{Permissions} \arrow[r, leftrightarrow] & \text{Resources}\end{tikzcd}$$

In addition to Authentication, as mentioned in the introductory course, we have the concept of Authorization.
Authorization is the process of verifying whether a user who has already been authenticated has the necessary permissions to perform one or more actions in the system.

To this end, as we discussed in previous videos, GAM has a scheme based on User Roles, where each user has one or more associated Roles. There are also secured Resources and the assignment of Permissions over these Resources to the Roles.

Resources can be, for example:

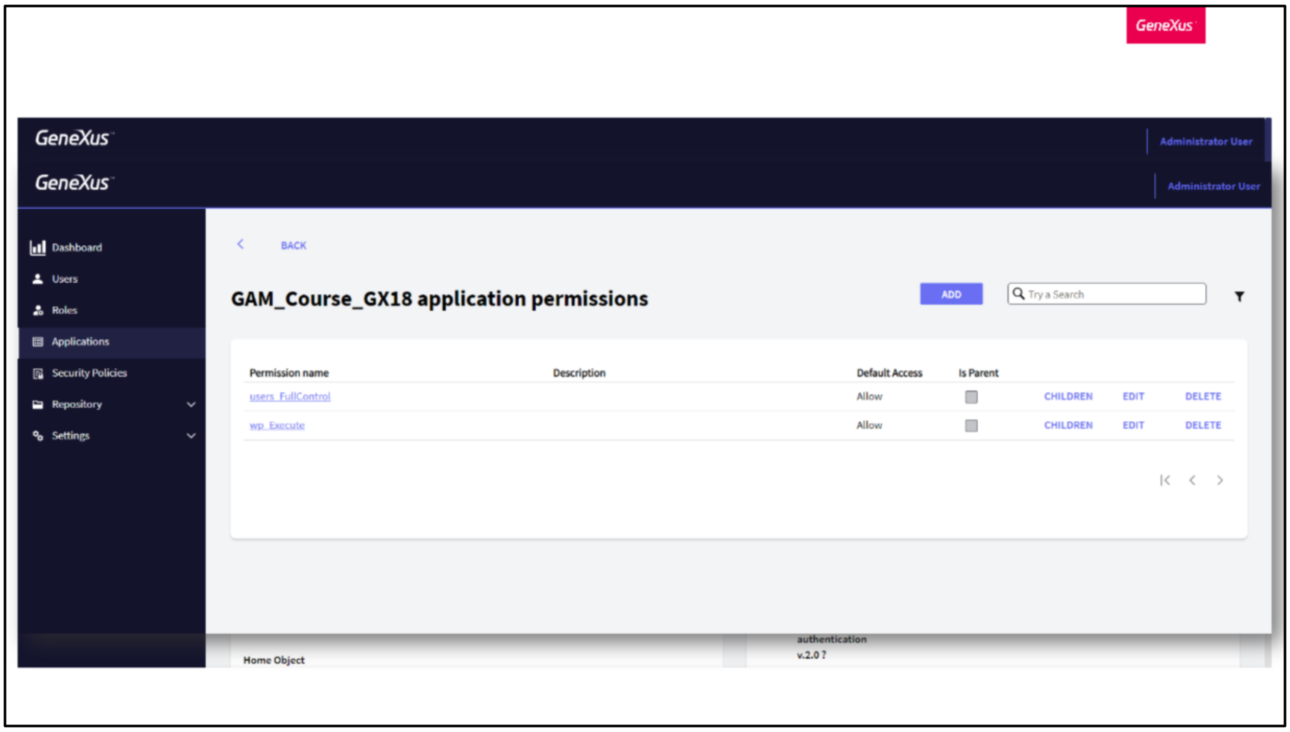
- Web Panels or Mobile panels
- Work With for mobile devices
- Web Components with URL Access enabled
- WEB Transactions
- Among others


Permissions

Permissions used for authorization can be assigned at different levels:

> Application level: Each Permission has an Access Type defined at the GAM Application level, where a Default Access Type is defined for each permission.
>
> Role level: When Permissions are assigned to Roles, they are defined with an Access Type.

Lastly, User Level, where Permissions are assigned to Users, and are defined with an Access Type just like at the Role level.



GAM_Course_GX18 application permissions

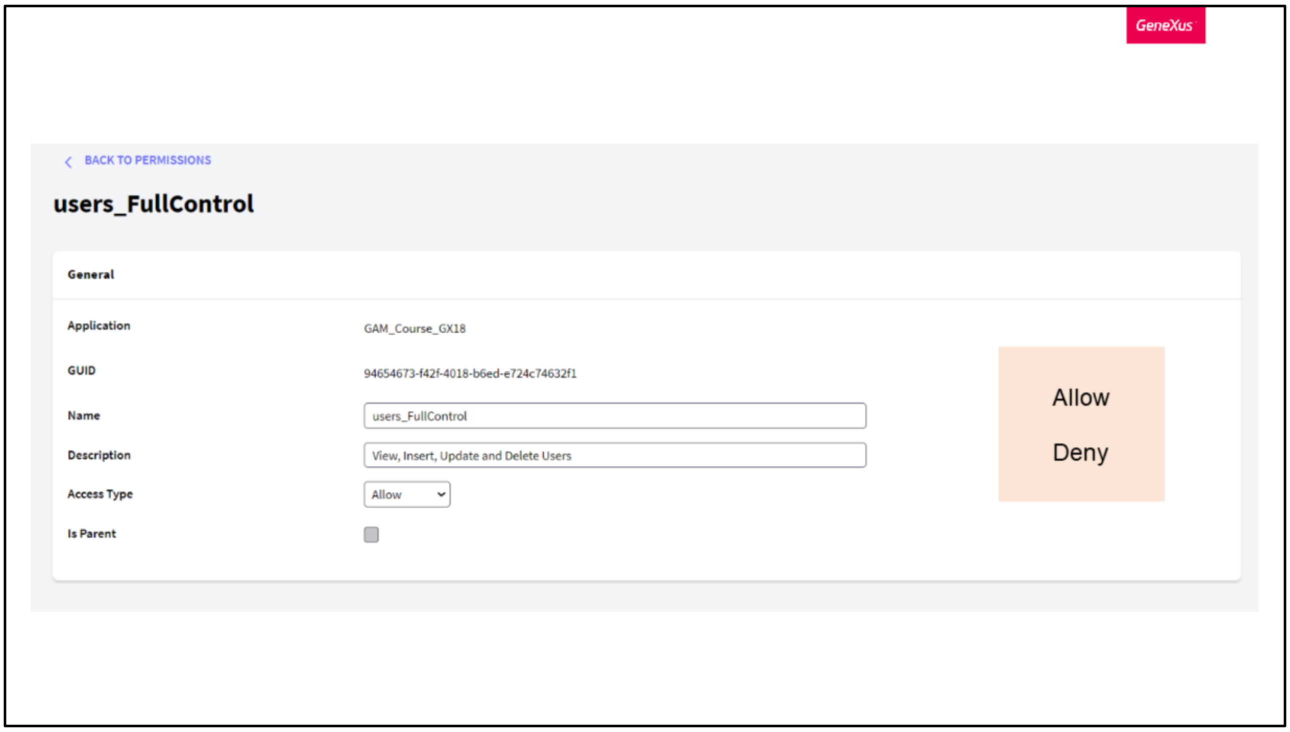
Permission name	Description	Default Access	Is Parent			
users_FullControl		Allow	☐	CHILDREN	EDIT	DELETE
wp_Execute		Allow	☐	CHILDREN	EDIT	DELETE

At the Application level, GAM will automatically generate permissions for each application.

To access these permissions, we simply go to the corresponding Application from the back end, and click on Permissions, within the “More options” submenu.

There we will find all the permissions that GeneXus generated. They consist of a name, description, default access, and whether they are a parent of another permission.

In this example shown on the screen, there is an execution permission (that’s why it ends in Execute) and a full permission, but they can also be generated for the Insert, Update and Delete modes. The full control permission grants the permissions mentioned above.


BACK TO PERMISSIONS

users_FullControl

General

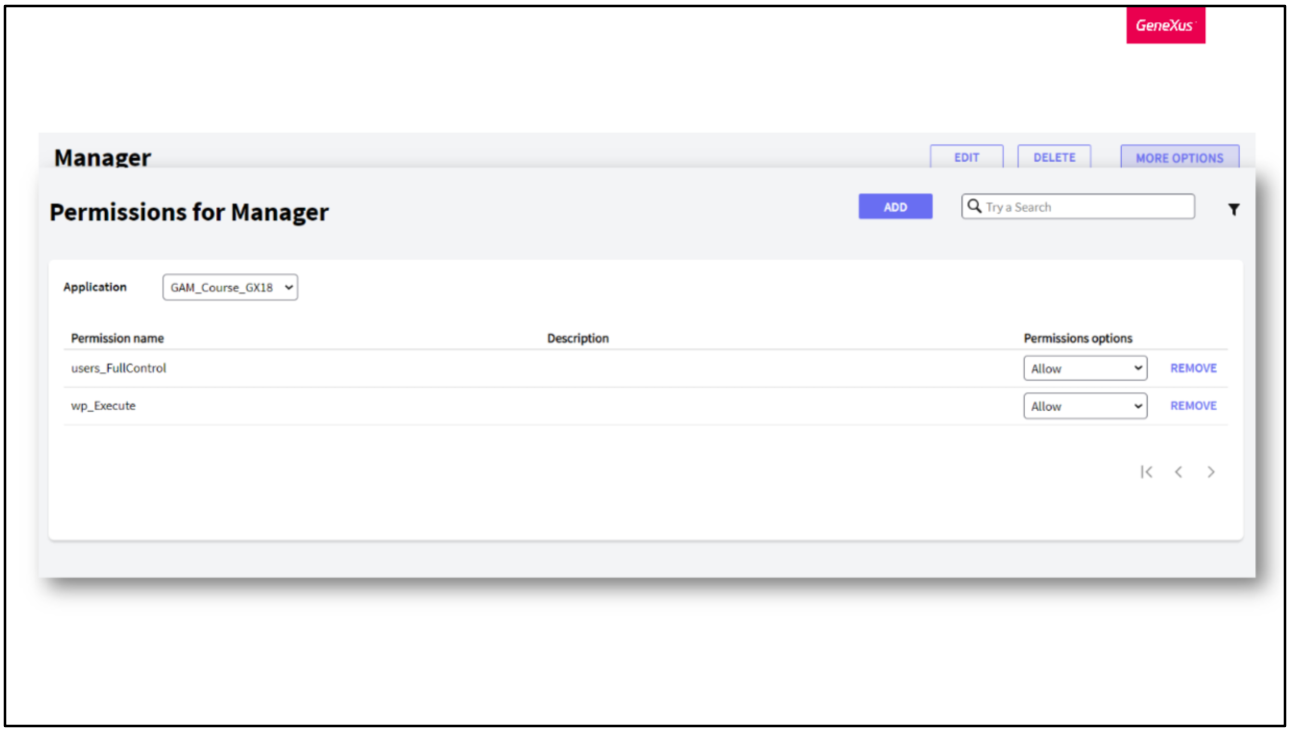
Application	GAM_Course_GX18
GUID	94654673-f42f-4018-b6ed-e724c74632f1
Name	users_FullControl
Description	View, Insert, Update and Delete Users
Access Type	Allow
Is Parent	☐

To edit a permission, the available options are to change its name, description, and access type.

The access type of a permission defines its default use (whether it is public or restricted). The available options are:

- Allow: This access type enables the permission for all users by default. Users who have this permission granted with: Access Type = Restricted or Deny, or who have any role where permission is restricted or denied, will not have this permission.

- Restricted: Users do not have this permission by default. This implies that only users who have this permission granted with Access Type = Allow, or who have some role where this permission is granted have the corresponding privileges.



GeneXus

Manager

EDIT DELETE MORE OPTIONS

Permissions for Manager

ADD

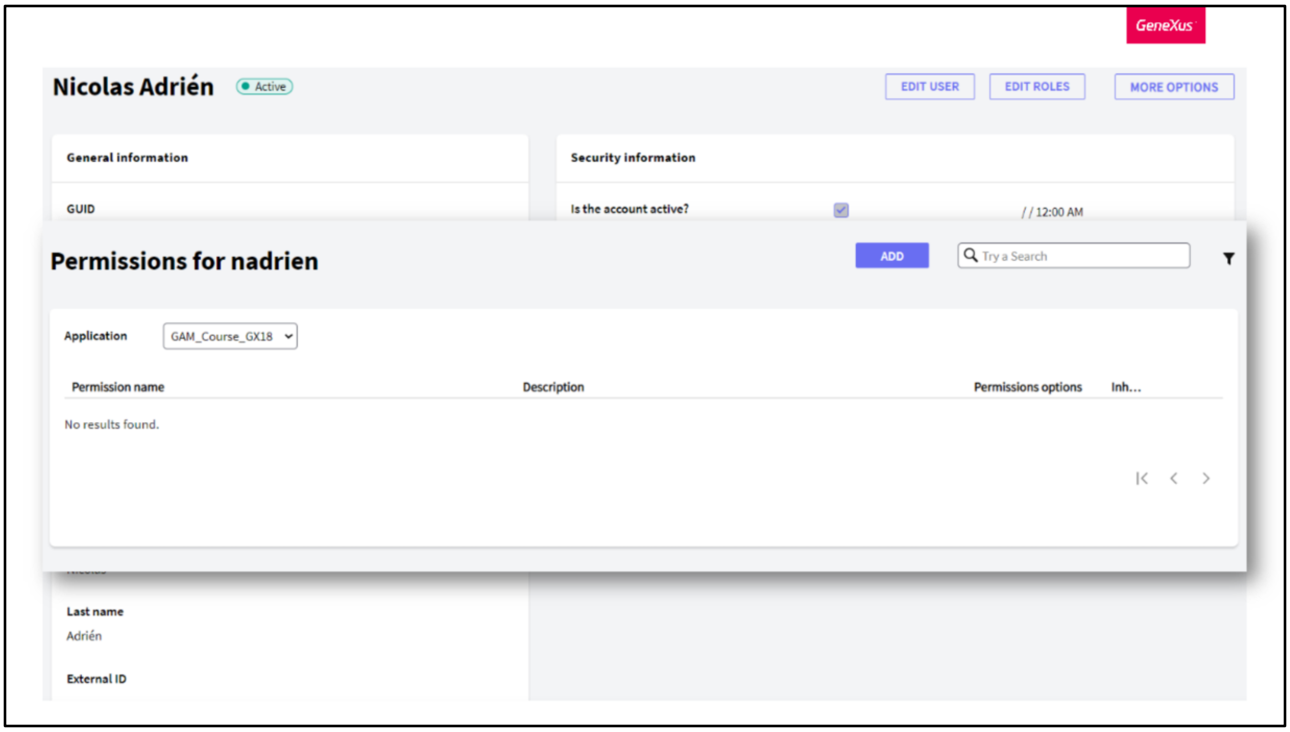
Try a Search

Application: GAM_Course_GX18

Permission name	Description	Permissions options	
users_FullControl		Allow	REMOVE
wp_Execute		Allow	REMOVE

At Role level, to add, edit, or delete permissions we must go to the Roles option; from a role we can manage its permissions through the “More options” submenu.
As we can see in the image, GAM allows us to Add or Delete a permission, modify its level as we said before (with the options Allow, Deny, and Restricted), and mark it as inherited or not.

Something to highlight is that if we add a permission to a role, transitively this permission will also be granted to all the users that have that role.


GeneXus

Nicolas Adrién Active

EDIT USER EDIT ROLES MORE OPTIONS

General information

GUID

Security information

Is the account active? ✓ / / 12:00 AM

Permissions for nadrien

ADD Try a Search

Application: GAM_Course_GX18

Permission name	Description	Permissions options	Inh...
No results found.			

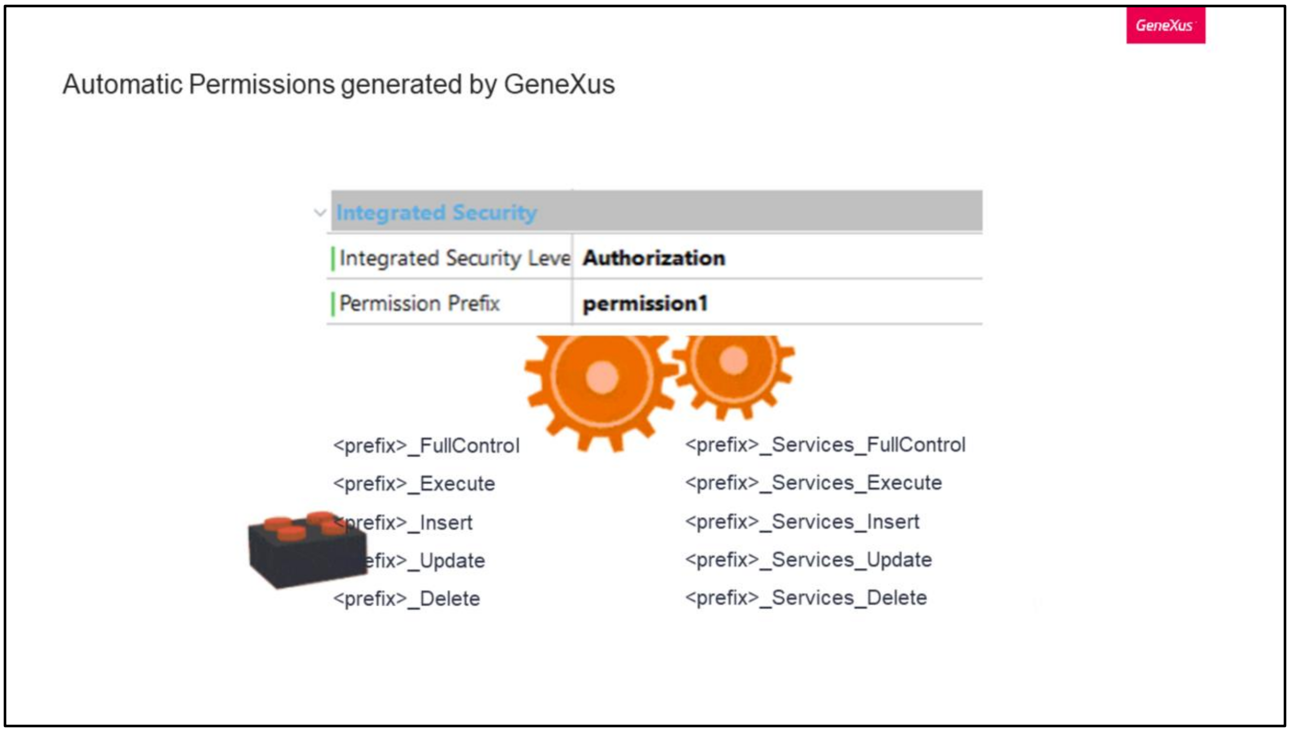
Last name

Adrién

External ID

Finally, the last level is by Users.
Just like for Roles, first we access the User. Then, when editing them, we have the option to edit permissions in the “More options” submenu.

They are handled in exactly the same way as by Roles as was just mentioned.
Note that permissions granted at this level take precedence over permissions granted at the role level; regardless of the type of access, the User level will always take precedence.



Automatic Permissions generated by GeneXus

Integrated Security	
Integrated Security Leve	Authorization
Permission Prefix	permission1

<prefix>_FullControl
<prefix>_Execute
<prefix>_Insert
<prefix>_Update
<prefix>_Delete

<prefix>_Services_FullControl
<prefix>_Services_Execute
<prefix>_Services_Insert
<prefix>_Services_Update
<prefix>_Services_Delete

Many of the permissions displayed in the previous screenshots were automatic permissions generated by GeneXus.

When a Build is done, these permissions are generated and then checked at runtime.

This assuming that the built-in Security Level property is set to the Authorization value, of course.

The code to check these permissions is included in the generated code, and the user only declares (through the Permission Prefix property) which permission is to be checked. Something positive to highlight is that, as you can see, it is not necessary to program anything; you only need to declare the necessary permissions to execute the object.

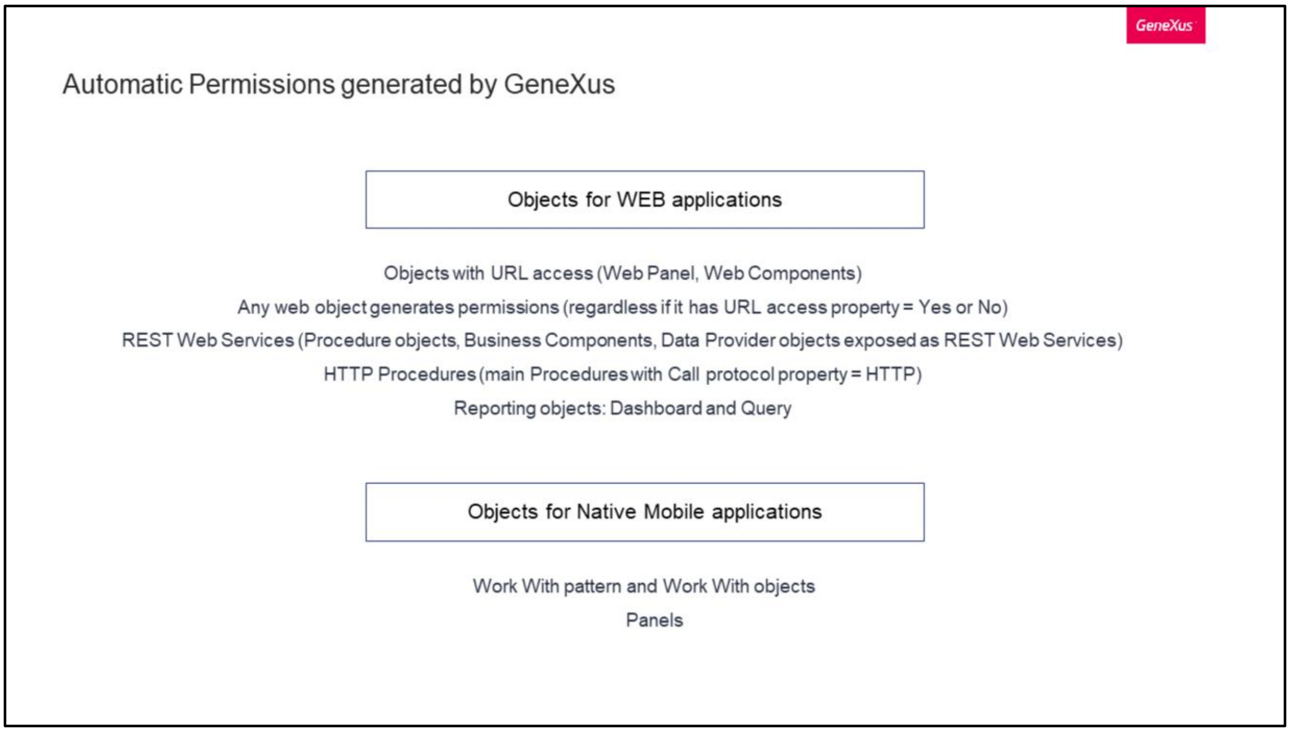
Automatic permissions can be described as follows:

First, there are execution Permissions where each object in the KB (except the Menu) exposes an access permission. Later on we will take a closer look at the objects that expose permissions.

Second, there are Permissions for executing the different modes of a transaction.
When a permission prefix is specified in any web transaction (suppose it is "prefix"), a

set of permissions is created in the GAM Repository, named as follows:
prefix.FullControl is the parent of the other permissions, and each permission is represented by the action concatenated to the prefix.

Third, there are Service Permissions.
If “prefix” is the permission prefix of a commercial component exposed as REST, the following permissions are automatically generated.



Automatic Permissions generated by GeneXus

Objects for WEB applications

Objects with URL access (Web Panel, Web Components)

Any web object generates permissions (regardless if it has URL access property = Yes or No)

REST Web Services (Procedure objects, Business Components, Data Provider objects exposed as REST Web Services)

HTTP Procedures (main Procedures with Call protocol property = HTTP)

Reporting objects: Dashboard and Query

Objects for Native Mobile applications

Work With pattern and Work With objects

Panels

In the previous slide, we said that each object of the KB (except the Menu) exposes an access permission. Let's see what these objects are.

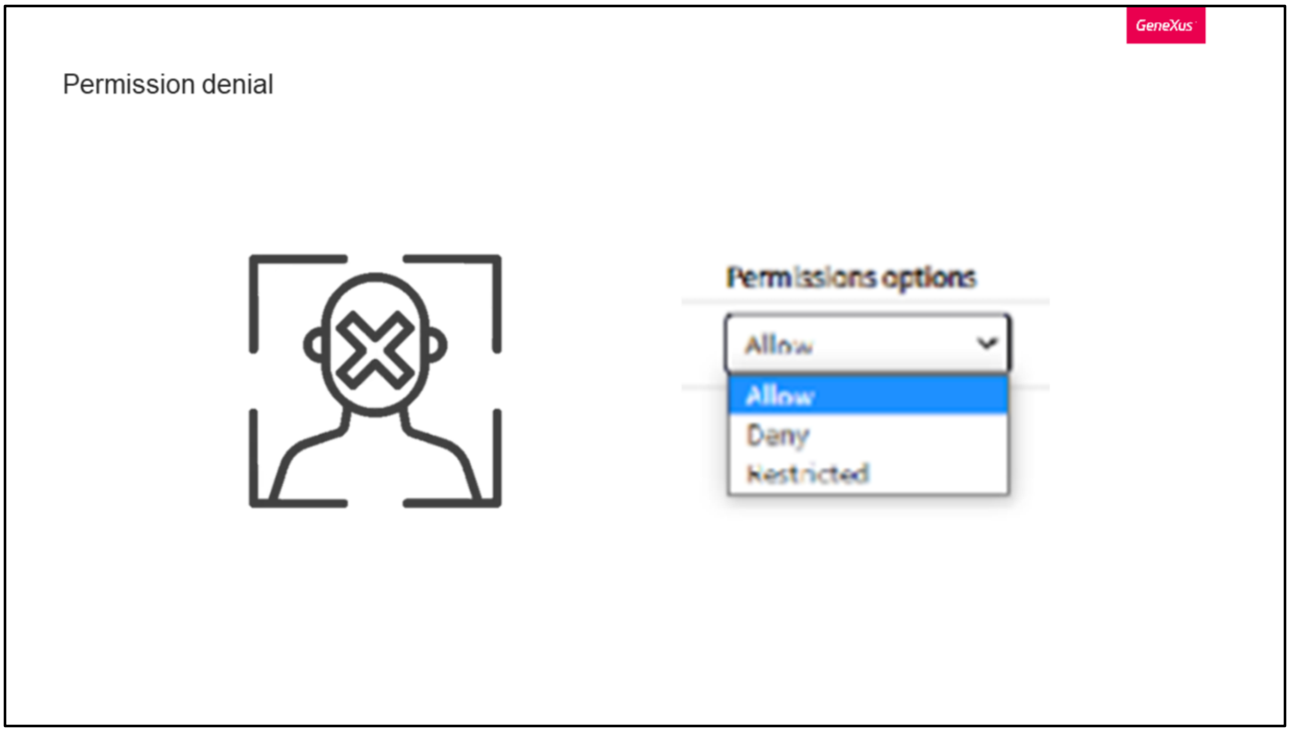
For web applications, we have:

- Web objects with URL access such as Web Panels and Web Components.
- As of GeneXus Evolution 3, any web object generates permissions, regardless of whether it has the URL access property set to Yes or No.
- REST web services, such as procedures, Business Components, or Data Providers exposed as REST web services.
- HTTP procedures, which are Main with call protocol property set to HTTP.

And finally,

- Reporting objects, such as Dashboards and Queries.

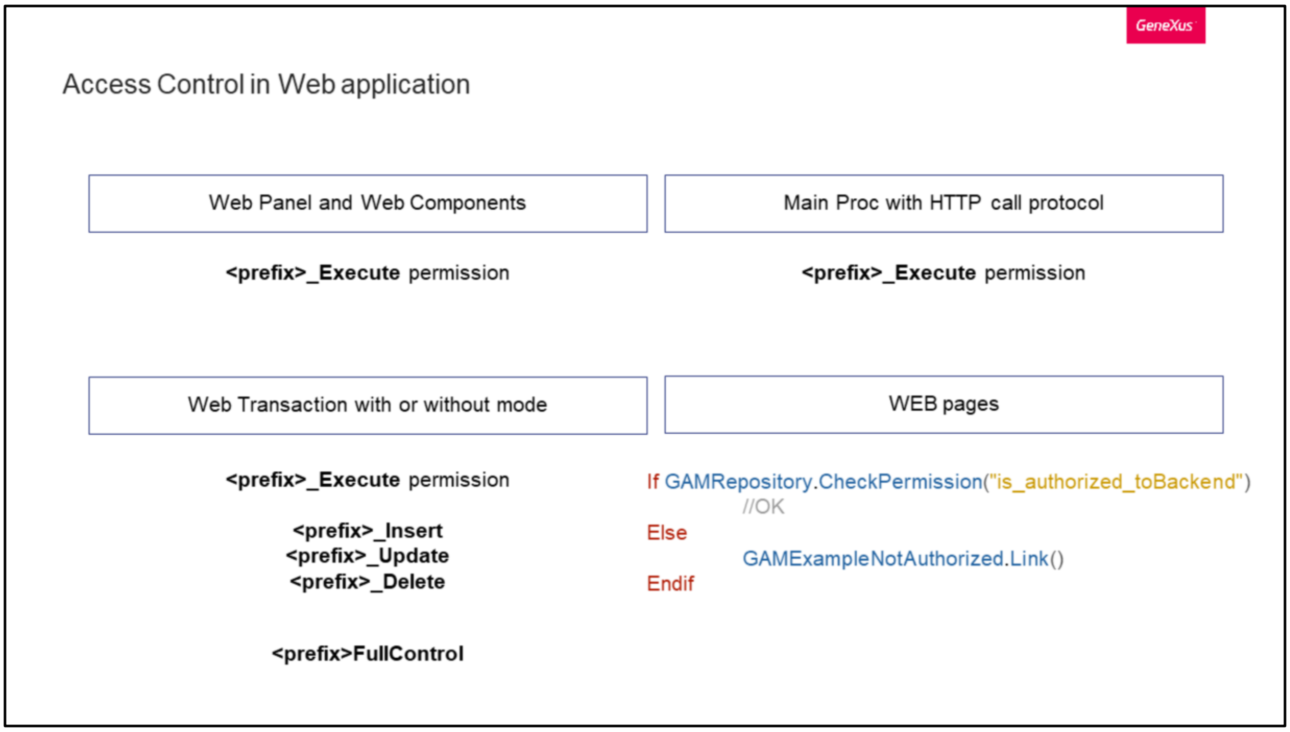
For native mobile applications we have the Work With pattern and Work With objects, as well as Panels.


Permission denial

GAM allows you to deny permissions. That is, to indicate that a permission cannot be used in a session whose user has that role.
We have already seen the option before where, in addition to the Allow and Restrict options, we also had Deny.

A user who has a role with the permission Access Type = Deny will not have this permission regardless of whether the permission is allowed at the application level (by default) or if they have another role where the permission is granted.

The only way the user can be granted this permission is with Access Type = Allow.



Access Control in Web application

Web Panel and Web Components

<prefix>_Execute permission

Main Proc with HTTP call protocol

<prefix>_Execute permission

Web Transaction with or without mode

<prefix>_Execute permission

<prefix>_Insert
<prefix>_Update
<prefix>_Delete

<prefix>FullControl

WEB pages

```
If GAMRepository.CheckPermission("is_authorized_toBackend")
            //OK
Else
            GAMExampleNotAuthorized.Link()
Endif
```

Let's see scenarios in which access control is performed.

First, we have Web Panels and Web Components (which only have access from the URL—URL access = true).
In this case, it is validated whether the user has permission to execute the object. If they don't, they should not see any of the data in the form.
To this end, GAM checks that the user has the <prefix>_Execute permission, where prefix is the permission Prefix defined for the object.
If a permission error is found, an automatic redirection to the “Not Authorized” Object will be made for web objects in the case of web applications.

There is also access to a main process with HTTP call protocol. An example of this can be a PDF report displayed in the browser.
Here we validate if the user has permission to execute this object. GAM verifies that the user has the <prefix>_Execute permission, where prefix is the Permission Prefix defined for the object.
In case of an error, the error 401 will be displayed. It is triggered by the application server and must be caught by the programmer.
In a PDF report, the “Not Authorized” object is considered for Web objects.

Third, there is access to a Web Transaction with or without mode.
First, it is checked if the user has permission to execute the object. In that case, GAM

verifies that the user has the permission <prefix>_Execute, where the prefix in this case is the one defined for the Transaction. This permission allows the user to view the transaction data (in view mode only).
If the user executes an action on the transaction—either Commit or Delete—other permissions will be required, as shown on the screen.
Actually, we also have a permission that groups all permissions together and can also be used.
If an error occurs, a GeneXus error message will be displayed if the Transaction does not receive KEY and mode as parameters. This can be seen in more detail in the GeneXus Wiki.

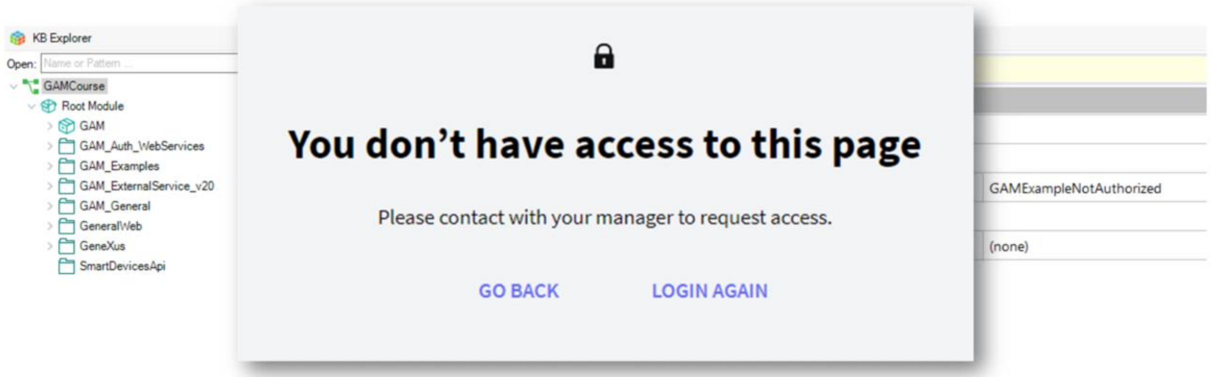
Finally, we have one last item. Restricted access to a group of WEB pages.
In some cases, the authorization level required is only used to allow or deny a group of users access to a set of web pages of the application.

For example, if we divide our application into front-end and back-end modules, probably only some authorized users will be able to access the back end.
One way to do this is to use a Master Page for the pages and program the following in its Start event.
In addition, the application property “Require access permissions” must be set to true.

The other option is to define automatic permissions as children of the permission “is_authorized_toBackend” and this would be enough.

Access Control in Web application

Not Authorized Object


KB Explorer

Open: Name or Pattern ...

GAMCourse
- Root Module
  - GAM
  - GAM_Auth_WebServices
  - GAM_Examples
  - GAM_ExternalService_v20
  - GAM_General
  - GeneralWeb
  - GeneXus
  - SmartDevicesApi

You don't have access to this page

Please contact with your manager to request access.

GO BACK   LOGIN AGAIN

GAMExampleNotAuthorized

(none)

In the previous slide, we often mentioned the Not Authorized object. Let's see how we can configure it.
At version level, we find the Not Authorized Object property for both Web and Mobile. There we can configure which object of the knowledge base we want to define so that the application redirects to it when the user is not authorized.

For Web, the GAM example will be used by default. You can use this one and even modify its design.
If you use one of your own, the built-in security level property must be set to "None."

GeneXus™ by Globant

training.genexus.com
wiki.genexus.com