GeneXus Access Manager (GAM)

Nicolas Adrién | GeneXus Training

The security module of any GeneXus application (both web and mobile applications) is provided by the GeneXus Access Manager, which is commonly referred to as GAM.

GeneXus™ ACCESS MANAGER

Authentication

Authorization

It was created in order to implement authentication and authorization functionalities in applications.
Security controls are automatically performed by enabling the application's built-in security.

Knowledge Base (KB)	Application
GAM external objects are imported into the KB	An automatic access control is generated for each object
A secondary data store is created to store GAM information	Session validation
A connection to the database is established	Permission check
Metadata is initialized	
The GAM Applications registry is executed	

What happens at the KB level when integrated security is enabled?

- External GAM objects are imported into the KB.
- A secondary data store is created to store GAM information.
- A connection to the database is established.
- Metadata is initialized.
- The GAM Application registry is executed.

What happens at the application level after integrated security is enabled?

- In this case, an automatic access control is generated on each object that has the Integrated Security Level property defined.
- Before executing any object, the generated code validates whether the session is valid; otherwise, a Login Object for Web or mobile is executed to start the session.
- In addition, permissions are automatically checked at startup (this happens when the Integrated Security Level property is set to “Authorization”).

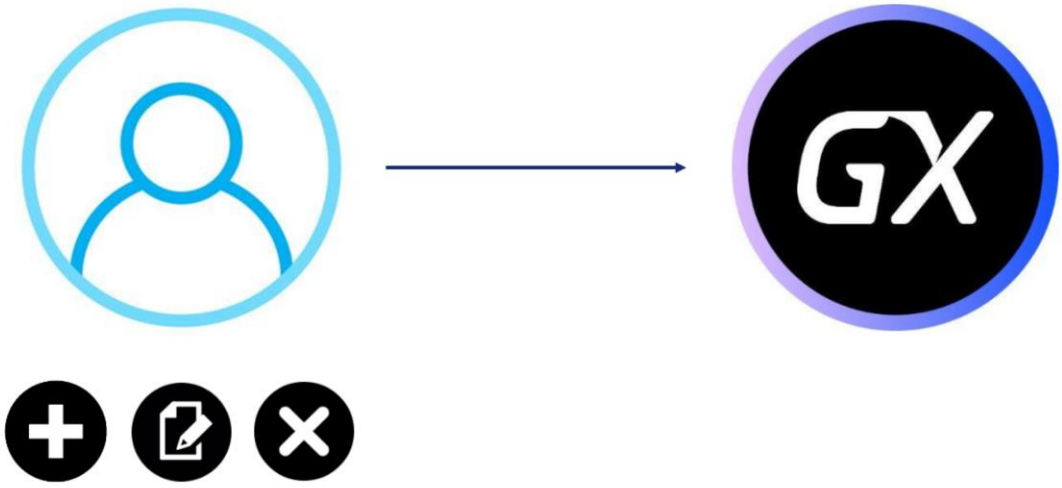
Roles

Permissions → Role → Users

Among GAM's elements, we find Roles again.
A role in GAM is the way to group permissions in an application, and they are organized in role hierarchies.

You can create new roles, edit existing ones, and add permissions to roles already defined in an application from the GAM Web Back office.
If a role is associated with a user, the role's permissions are associated with that user. This means that when user permissions are checked at runtime, the permissions associated with the user across roles are taken into account. However, the permissions directly associated with the user take precedence.

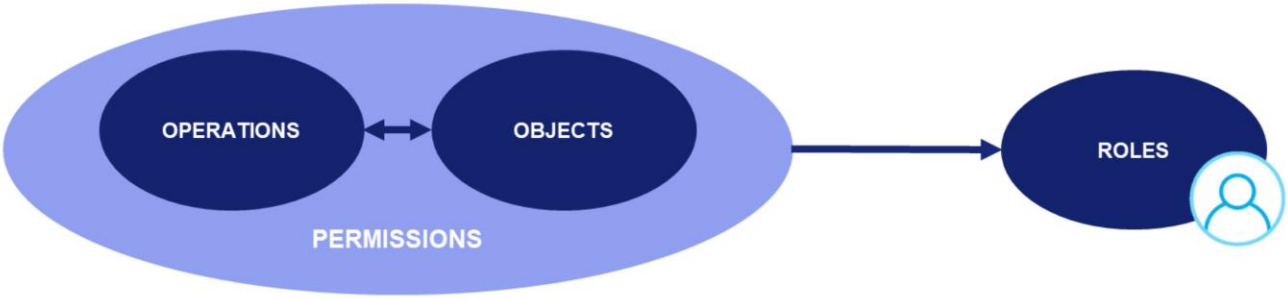
Users



Users are people who use a GeneXus application on a regular basis.
With GAM you can add, edit, and delete users.

In turn, a GAM user can have several associated Roles and a Main Role.

Permissions



Allow	Deny	Restricted

Permissions, as said before, describe the possibility of performing operations on objects.
They can be associated with one or more roles.

A permission has a default action type that specifies whether the permission is restricted or not: Allow or Restricted. By default, it is enabled for all users.

Permissions are granted per application, and when assigning them to a Role you can specify their action for that specific Role. The options are Allow, Deny, or Restricted.
When Allow is assigned, the permission is granted. To deny a permission, we can use the other two options: Deny or Restricted.

When a session is created for a user, GAM obtains the roles associated with it. To check if a User has a valid permission, GAM checks on all those Roles: If one of them has a Restricted permission and another has the same permission set to Allow, Allow takes precedence. However, if one of them has a Deny permission and another has the Allow permission, Deny takes precedence because it is stronger than a Restricted permission.

Lastly, the permission is assigned directly to a user.

Sessions

Just like in RBAC we have Sessions, which map users to their active roles.

Conceptually, they are handled in exactly the same way as described in RBAC.
Each session corresponds to each user and is handled automatically by GAM. They contain an identification token, and expire according to the security policies defined.

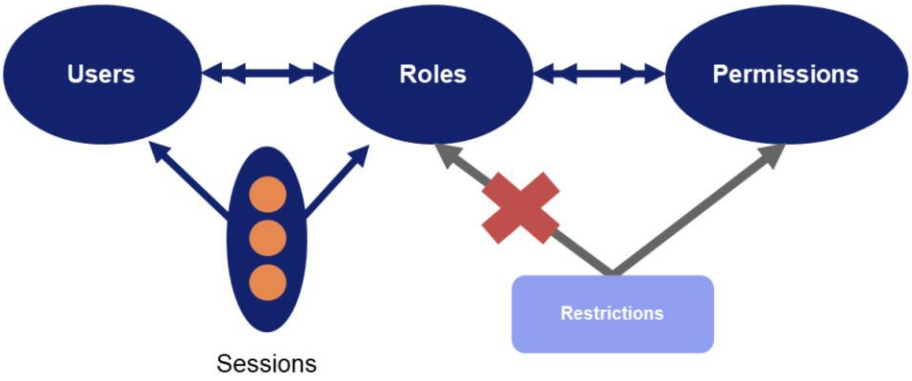
GAM and RBAC Model

At what level can we say that the GAM is part of RBAC?

GeneXus™

At what level can we say that the GAM is part of RBAC?

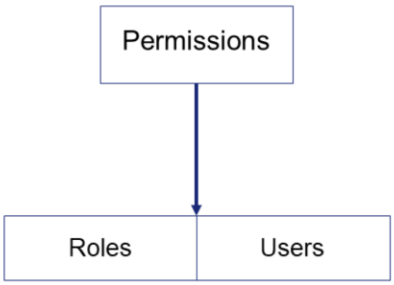
GAM and RBAC Model



The GAM is located at level 3, where role separation mechanisms are applied, with certain permissions for each user. Remember that it corresponds to this level.

As for Restrictions, although there are restrictions at permission level, they do not exist at role level.

Scope, levels, and inheritance of Permissions in GAM

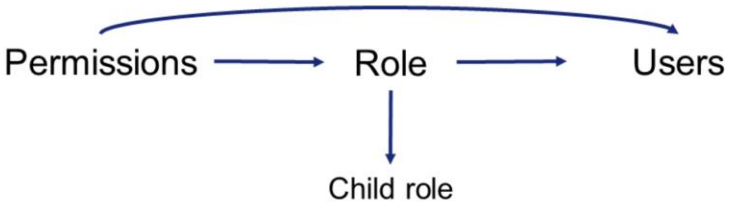


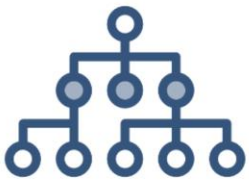
Permissions in GAM exist within the scope of the Applications and are assigned to Roles and Users in the GAM Repository.

As for Levels, the level of permissions a user has at runtime depends on the type of permission access.

Lastly, when any full control permission is added to a role, by default all the secondary permissions of this full control are also added to the role, unless inheritance has been lost.

Role Hierarchy in GAM

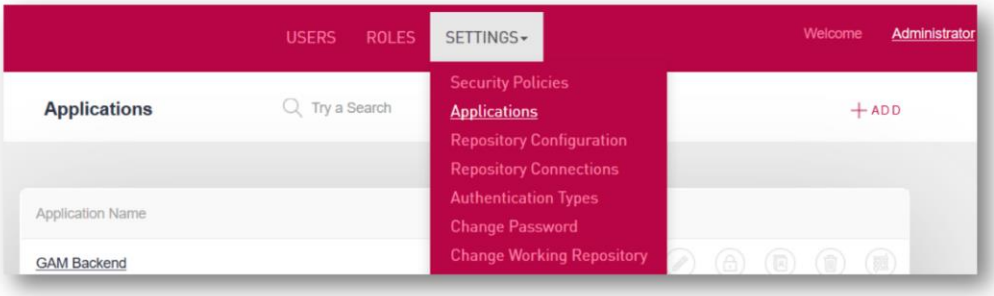




As said before, if a role is associated with a user, the role's permissions are indirectly associated with that user. This means that when user permissions are checked at runtime, the permissions associated with the user across roles are taken into account.

In addition, if the role has children, the permissions of the child roles are also associated with the user.

Administration



Users

Roles and their Hierarchy

Security policies

GAM Permissions in Applications and Roles

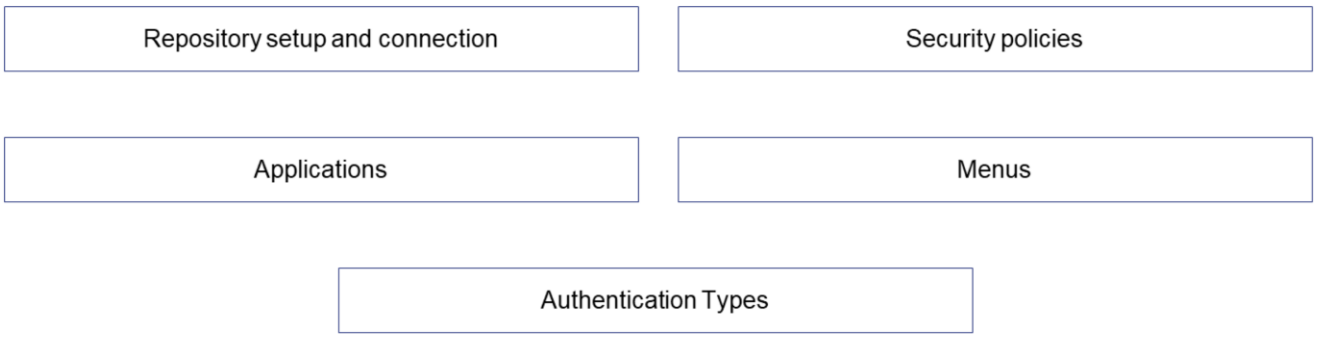
The GAM backend is a web application that allows the GAM administrator to manage the entire system.
These functions include everything we mentioned before in relation to RBAC:

- Users
- Roles and their Hierarchy
- Security Policies
- GAM Permissions in Applications and Roles

It also allows managing applications, configuring the repository and its connections, changing passwords, authentication types, etc. We will discuss them next...

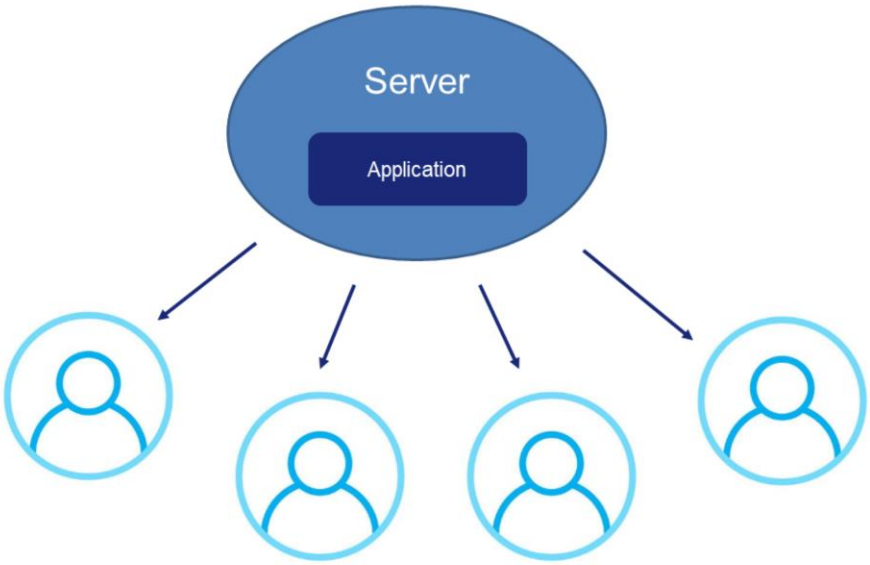
Other fundamental concepts of GAM

GeneXus™



In addition to the concepts mentioned so far, GAM also allows you to manage more options, such as the configuration and connection of the repository, its applications and security policies, menus, and the types of authentication available.

Repository

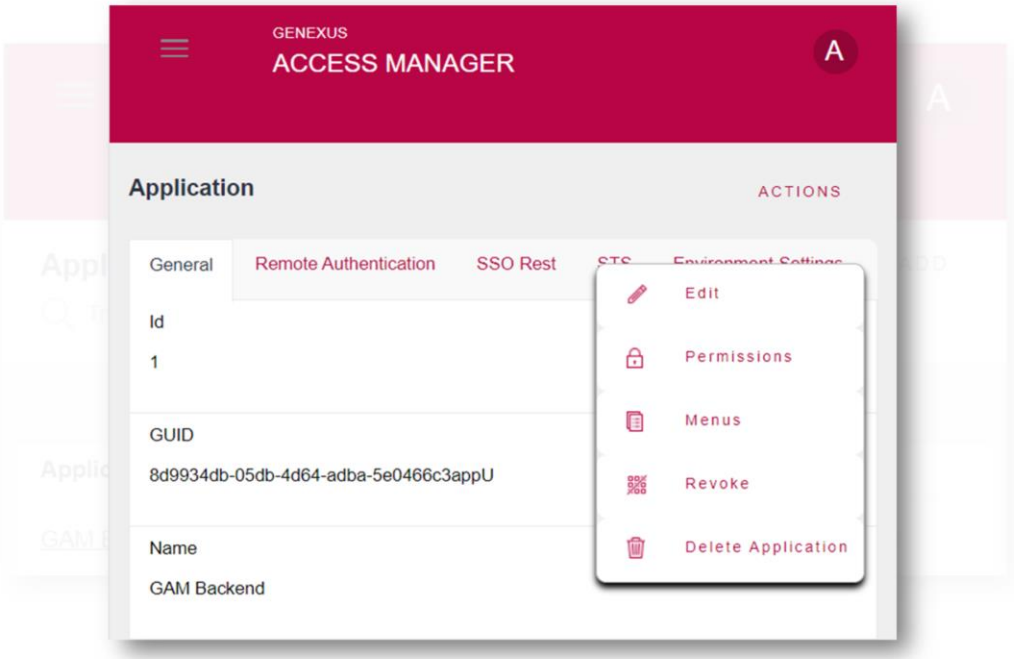


First, let's see the Repository.

A repository is a GAM entity that supports an architecture where a single instance of the application runs on a server and serves multiple users.

In this scenario, these users must use the same GAM database and different GAM repositories in the database.
This is commonly referred to as a multi-user application, and provides each user with a unique set of GAM roles, applications, and security policies.

Applications

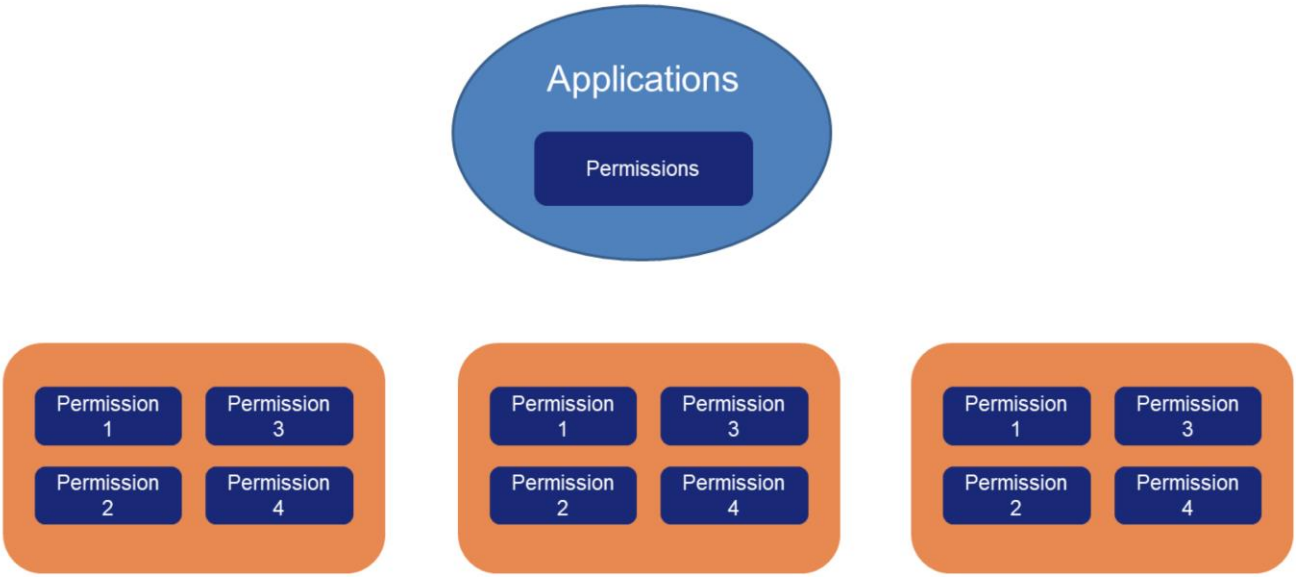


What are GAM applications?
GeneXus has a web environment and a GAM web application, which can be identified by a GUID.
In addition, the application has a name (the name of the KB) and includes the permissions of all web objects in the KB.

About Mobile Applications:
GAM Mobile applications group the permissions of all the main mobile objects in the KB. Each of these main objects determines the existence of a GAM application.

GAM applications are defined within a repository and, as we said, each repository can contain more than one application.

What can be a purpose of GAM applications?



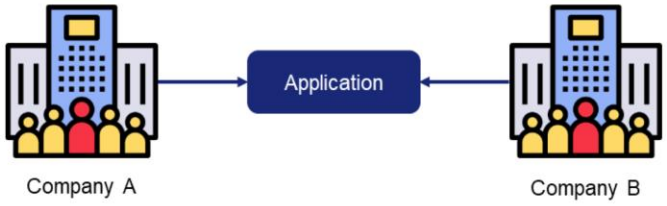
What can be a purpose of GAM applications?

One of them may be to associate permissions with these applications and form permission groups.

Since permissions are checked against the running application, **when the user logs into a repository and a permission is needed to perform an action, the permission must be defined in the GAM application that is running (and must have a role where this permission is granted).**

Scenarios of Applications-Repositories

1. Many companies share the same application installation (Multi-Tenant application)



GetConnections

SetConnections

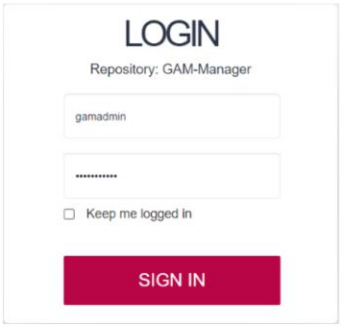
Let's consider some examples of scenarios with Applications and Repositories.

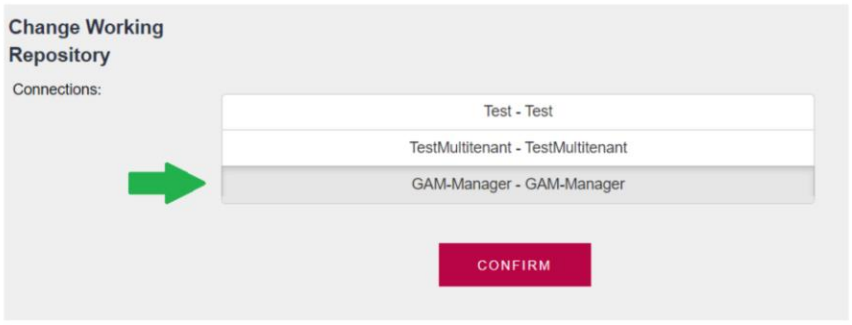
The first scenario is composed of many companies sharing the same application installation (defined as Multi-Tenant application).
As we said before, the design of the GAM model allows connecting to multiple Repositories to solve many scenarios where a single GAM Repository would not be enough.
In this case, each Repository will have its own administrator users, and the information from one Repository will not be accessible from the others.

This is the case of a multi-user application where a single instance of the software runs on a server, serving multiple client organizations.

Putting this scenario into practice, we have two GAM methods to use: **GetConnections** and **SetConnections**.
The GAM.GetConnections method returns in a collection a list of connections containing the key stored in the connection.gam file.
The GAM.SetConnection method returns true if the connection was successfully established. This means that all GAM methods will access the new set of repositories.

Let's see an example of this.






```
Event Start
    CurrentRepository.Visible    = False
    TableButtons.Visible         = False

    //Valid current connection
    &isConnectionOK = GeneXusSecurity.GAM.CheckConnection()
    If GeneXusSecurity.GAM.isMultitenant()
        Do 'isMultitenantInstallation'
```

Before we delve into it, let's look at a concept used in implementing this: the GAM Manager Repository.
It is a particular Repository used to manage the rest of the Repositories. The users of this Repository, and the users of repositories that have the Manager property enabled, are the only ones that can create new Repositories and manage them.

We will not go into detail on this topic, so we recommend reading the documentation on the GeneXus Wiki to understand it fully. We will simply assume that it is already created and configured.

We can explain this with one of the examples provided by GAM.
In this case, we will use GAMExampleLogin.
Note that in the Start event, the current connection with the Repository is checked, and if we are in a Multi-tenant case, a call to a subroutine is made.

```
Sub 'isMultitenantInstallation'
    //Read Curent Repository
    &GAMRepository = GeneXusSecurity.GAMRepository.Get()
    //Check if the current repository uses an authentication master repository
    If not &GAMRepository.AuthenticationMasterRepositoryId.IsEmpty()
        &isConnectionOK = GeneXusSecurity.GAM.SetConnectionByRepositoryId(&GAMRepository.AuthenticationMasterRepositoryId, &Errors)
    Endif
    If not &isConnectionOK
        If GeneXusSecurity.GAM.GetDefaultRepository(&RepositoryGUID)
            &isConnectionOK = GeneXusSecurity.GAM.SetConnectionByRepositoryGUID(&RepositoryGUID, &Errors)
        Else
            &ConnectionInfoCollection = GeneXusSecurity.GAM.GetConnections()
            If &ConnectionInfoCollection.Count > 0
                //The first connection found is established by default
                &isConnectionOK = GeneXusSecurity.GAM.SetConnection(&ConnectionInfoCollection.Item(1).Name, &Errors)
            EndIf
        Endif
    Endif

    If &isConnectionOK
        &GAMRepository = GeneXusSecurity.GAMRepository.Get()
        //Check if the current repository uses an authentication master repository
        If not &GAMRepository.AuthenticationMasterRepositoryId.IsEmpty()
            &isConnectionOK = GeneXusSecurity.GAM.SetConnectionByRepositoryId(&GAMRepository.AuthenticationMasterRepositoryId, &Errors)
            &GAMRepository  = GeneXusSecurity.GAMRepository.Get()
        Endif
        CurrentRepository.Caption = "Repository: " + &GAMRepository.Name
        CurrentRepository.Visible = True
    Endif
EndSub
```

In the subroutine, the first thing that is brought is the current repository, to then check whether it is a master one and connect.

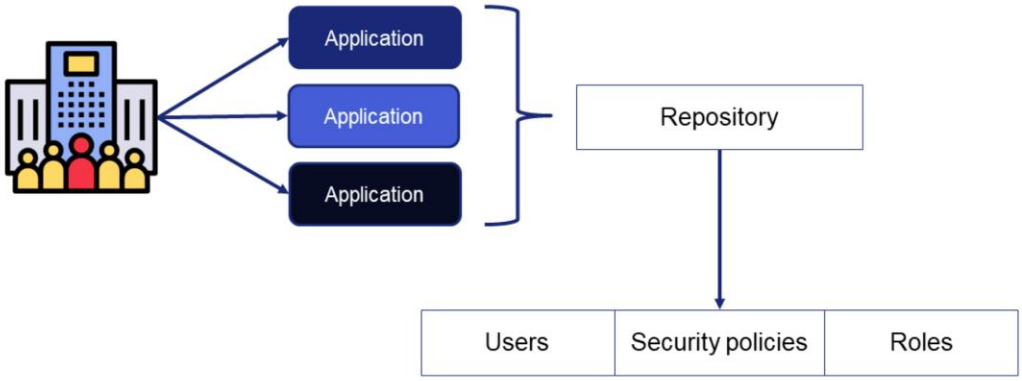
If the connection was not successful, it asks whether the current repository is the default repository to set a connection with it. If it is not, the methods that we saw before are used: GetConnections and SetConnections.
First, it runs the Get to obtain the list of connections, and keeps the first one.

Let's keep that part of the code. In this case, it keeps the first one because it is an example of basic implementation. In a customized case, the developer can choose which repository to connect to according to the conditions to be exposed.
Remember that these implementations are an example and a guide for you to make your own implementations.

With those two methods, we have full control of the connections to the repositories. The rest of the code is not relevant here.

Scenarios of Applications-Repositories

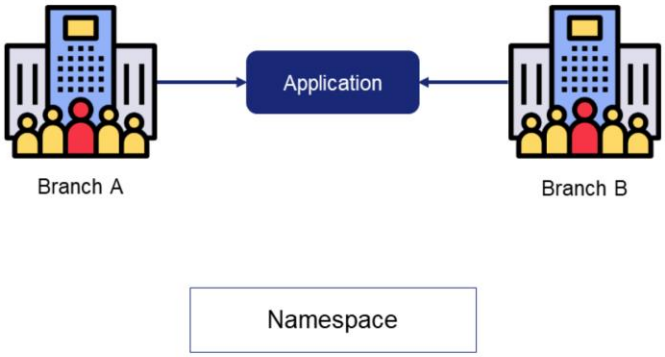
2. A company with different mobile and web applications



In the second scenario, we have a company with different mobile and web applications.
In this usage scenario, the company has different applications, and they share users, security policies and roles, and have a single repository.
In addition, each application contains its own permissions.

Scenarios of Applications-Repositories

3. A company with different branches

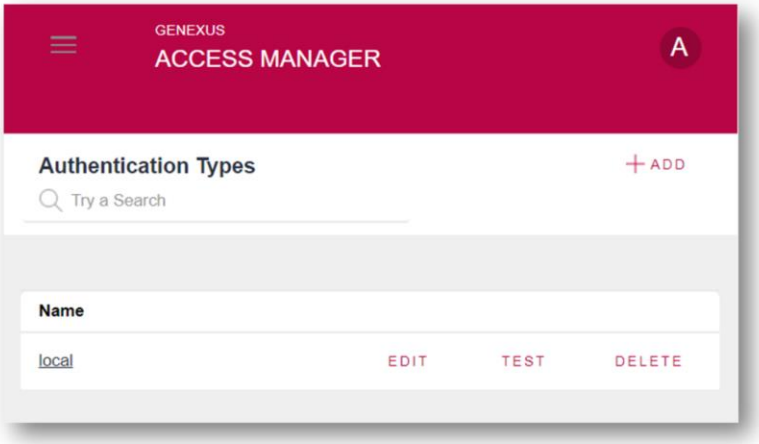


In the third and last scenario, we have a company with different branches.
In this usage scenario, the company has a single application that is used by all the different branches existing in the company. That is, each branch is represented by a repository.
Unlike scenario 1, in this case users are the same for all branches, with the caveat that each user may have different security policies, roles and permissions, depending on the branch of the application to which they are connected.

Given the design of the repository model in GAM, users could be enabled in many repositories if they have the same namespace as that of the repository in which they are enabled.
The purpose of the GAM Repository Namespace is to group in an abstract container all the repositories that belong to the same company.

It should be emphasized that the three scenarios are solved with only one database, without the need to create others for each branch or application.

Authentication Types



Internal

Remote

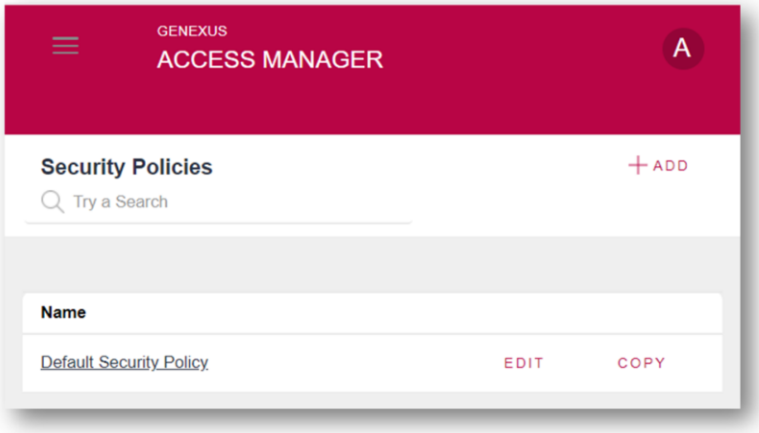
External

Authentication Types
Authentication is the act or process of confirming that something (or someone) is who they claim to be.

GAM offers different types of authentication, including internal, remote, and external (such as web services, social networks or Google).

This will be covered in detail later in the course.

Security policies



Minimum time to change passwords

Web session timeout

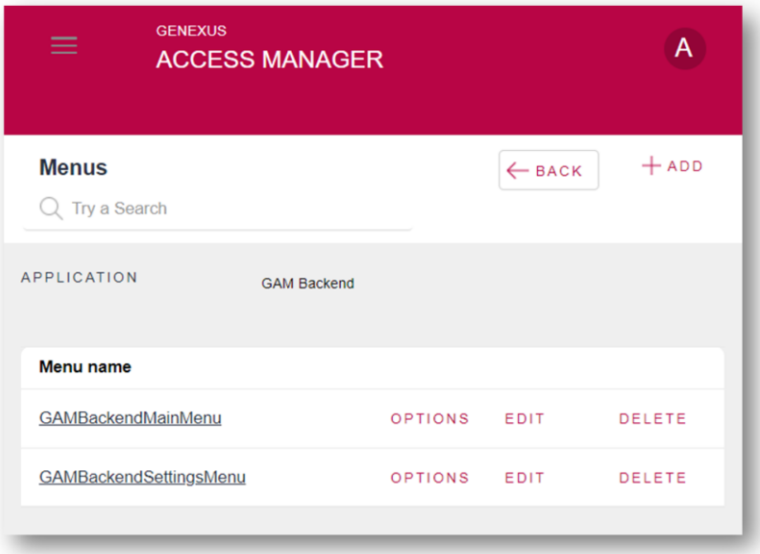
Minimum number of characters in passwords

Security policies are a set of rules, standards, and protocols that ensure the security of roles and users in GAM.

They can be defined using the GAM Web Back office, or programmatically using the GAM API.

Among those used by GAM, we can highlight the Minimum time to change passwords, the Web session timeout, the Minimum number of certain characters in passwords (such as numerical characters, special characters, upper and lower case), among others.

Menu



Permissions

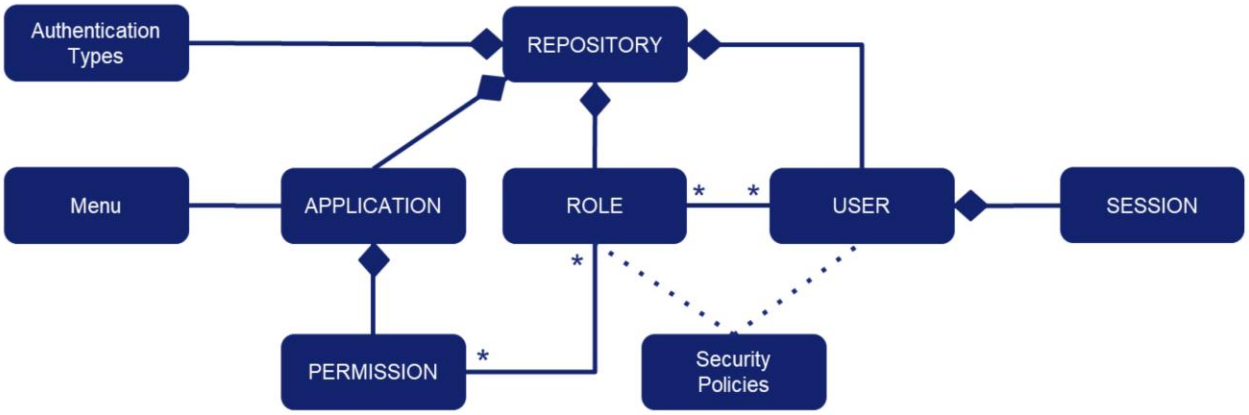
Roles

With GAM you can define (dynamically) a menu of the web or mobile application at runtime, based on the GAM permissions and roles of the logged in user.

GAM returns the menu structure, depending on the user's permissions so that this structure can be loaded at runtime with any User Control.

The menu must be defined for a particular GAM application. You can define as many menus as you want within a GAM application.
A menu item will have an associated permission and resource.

Summary



The Asterisk (*) indicates many, without specifying how many.
The rhombus (◆) indicates a composition, and the element at the end where the rhombus is placed represents the whole.

Summarizing everything discussed so far, in the diagram displayed on the screen we have all the components we mentioned about the GAM.

Before looking at the diagram in more detail, regarding the naming convention used, as indicated in the slide, the asterisk indicates many. In turn, the rhombus indicates a composition, which means that, for example, there can be no sessions if there is no user that contains it.

Moving on to the content of the diagram, we can specify that the set of GAM Permissions falls within the scope of a single Application and Repository.
Therefore, GAM Applications can be associated with *n* GAM Permissions, and these are determined by an Application in a Repository.
Roles are then defined within a Repository, and associated with Permissions where these can be used in many Roles.
Users can be enabled in one or more GAM repositories, and they are the content of the sessions. In addition, they can have many roles and indirectly be assigned one or more permissions.

Authentication Types and Security Policies are within the scope of a Repository.
Menus are within the scope of the applications, which in turn, are within the scope of a Repository.

GeneXus™

training.genexus.com
wiki.genexus.com
training.genexus.com/certifications