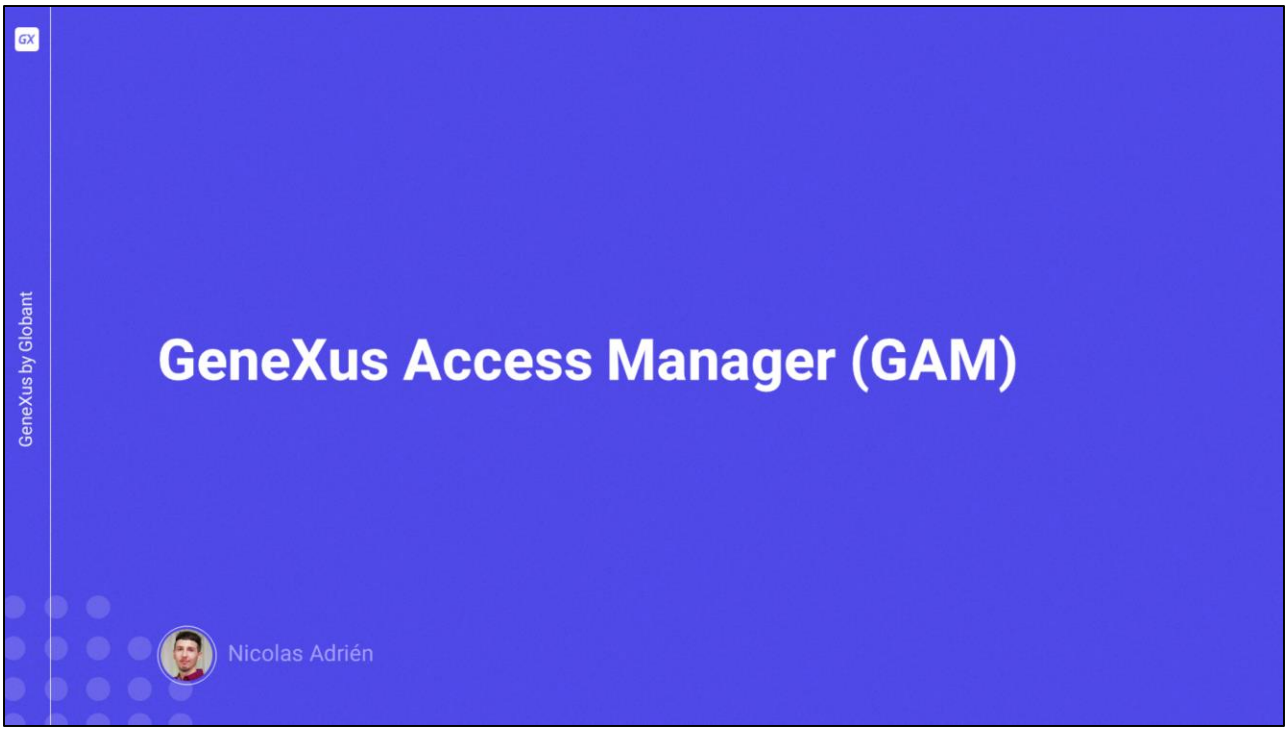
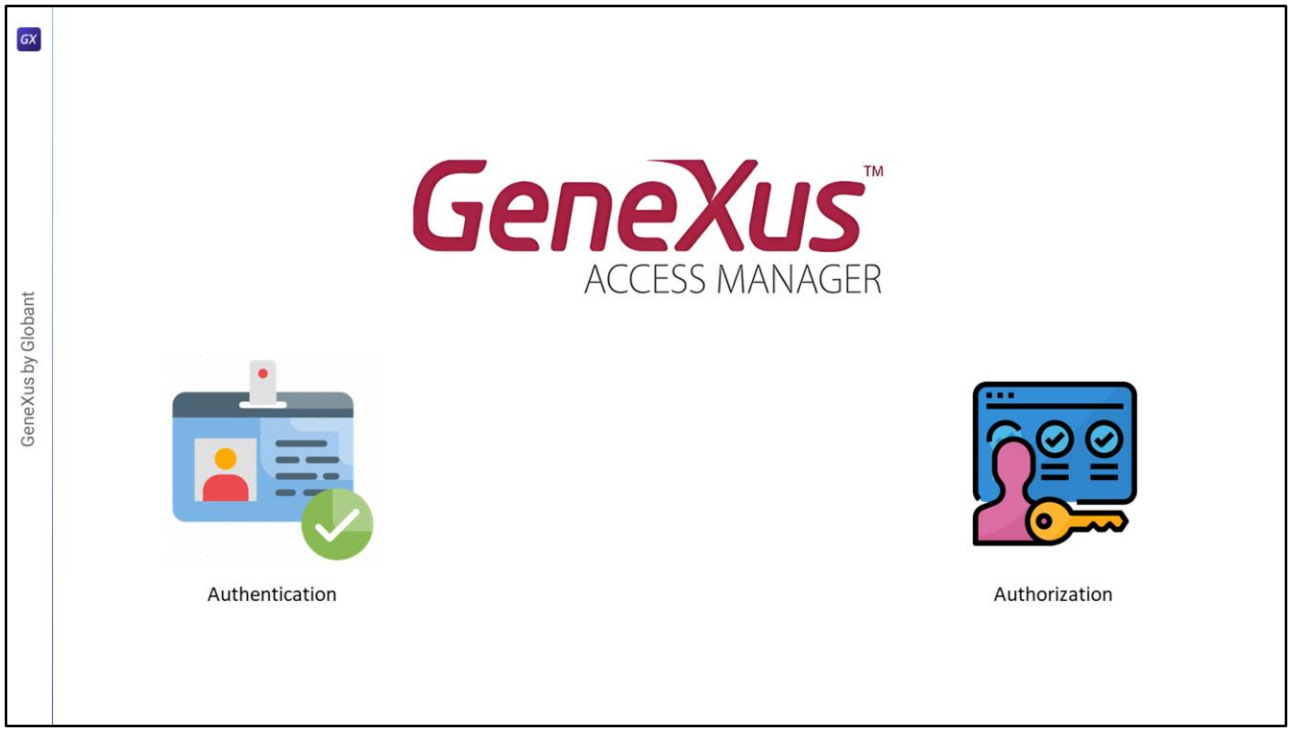
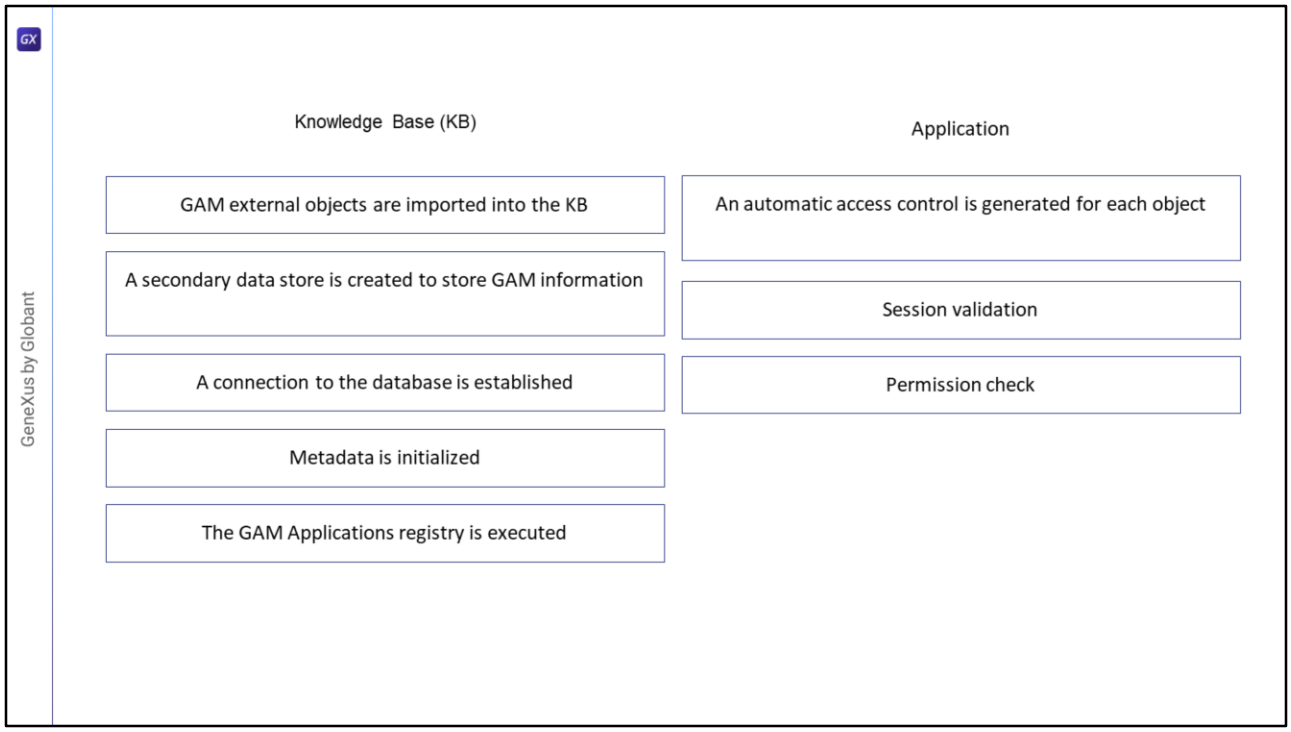

GeneXus by Globant
GeneXus™
by Globant
training.genexus.com



O módulo de segurança de qualquer aplicação GeneXus (tanto aplicações web como móveis) é fornecido pelo GeneXus Access Manager, ou comumente chamado GAM.



Este foi criado com a finalidade de resolver as funcionalidades de autenticação e autorização das aplicações.
Os controles de segurança são realizados automaticamente ao habilitar a segurança integrada na aplicação.

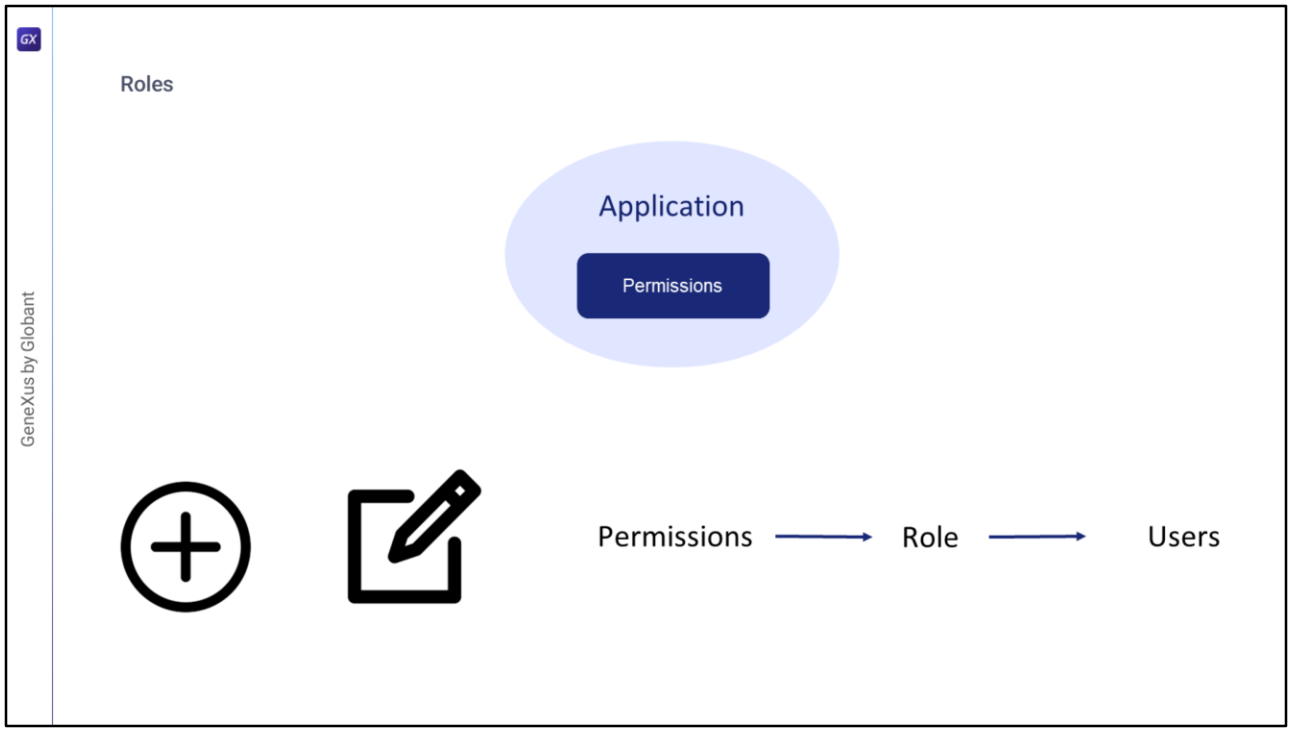


O que acontece no nível da KB depois de habilitar a segurança integrada?

- São importados para a KB os objetos externos GAM.
- É criado um armazenamento de dados secundário para armazenar informações GAM.
- É estabelecida uma conexão com a base de dados.
- São inicializados os metadados.
- É executado o registro de Aplicações GAM.

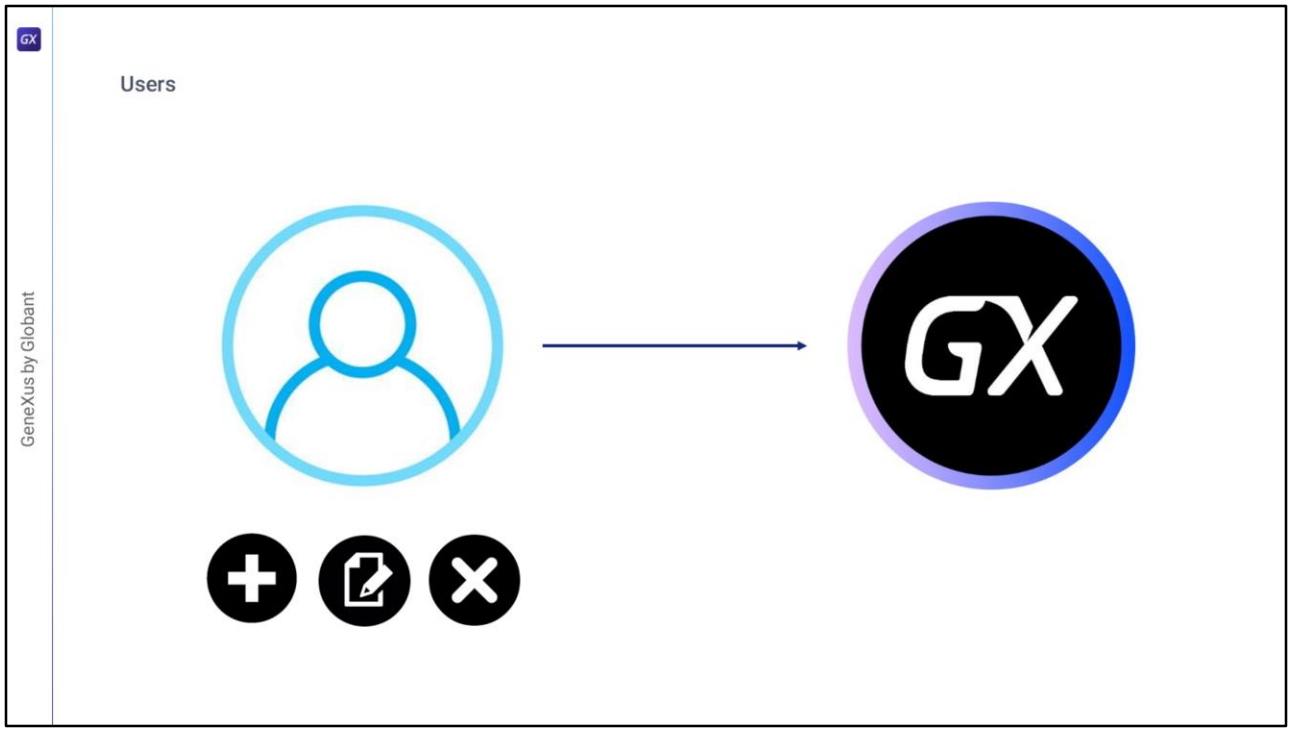
Agora, o que acontece no nível de aplicação depois de habilitar a segurança integrada?

- Neste caso, é gerado um controle de acesso automático em cada objeto que tem definida a propriedade de Nível de Segurança Integrado.
- Antes de executar qualquer objeto, o código gerado verifica se a sessão é válida; caso contrário, é executado um Objeto de login para Web ou móvel, para efetuar login.
- Além disso, são verificadas automaticamente as permissões na inicialização (este último ocorre quando a propriedade Nível de segurança integrado está com o valor “Autorização”).



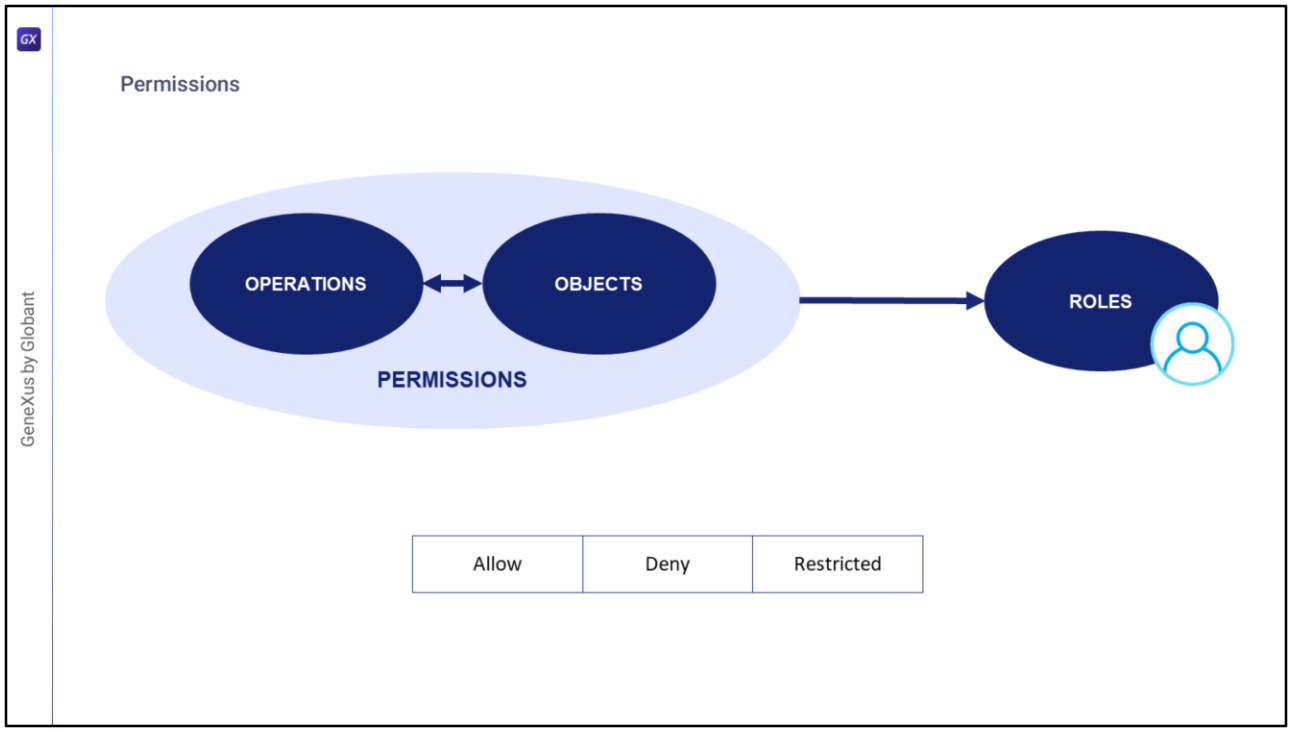
Indo para os elementos próprios do GAM, temos novamente as Roles.
Uma role no GAM é a maneira de agrupar permissões em uma aplicação e estão organizadas em hierarquias de roles.

Podem ser criadas novas roles, editar existentes e adicionar permissões a roles já definidas em uma aplicação a partir do Backoffice web do GAM.
Se uma role estiver associada a um usuário, as permissões da role estão associadas a esse usuário. Isto significa que quando são verificadas as permissões de usuário em tempo de execução, são levadas em consideração as permissões associadas ao usuário através das roles. No entanto, as permissões associadas diretamente com o usuário têm prioridade.



Quanto aos usuários, estes são aqueles indivíduos que utilizam habitualmente uma aplicação GeneXus. Com GAM podem ser adicionados, editados e excluídos usuários.

Por sua vez, um usuário do GAM pode ter várias Roles associadas e uma Role Principal.



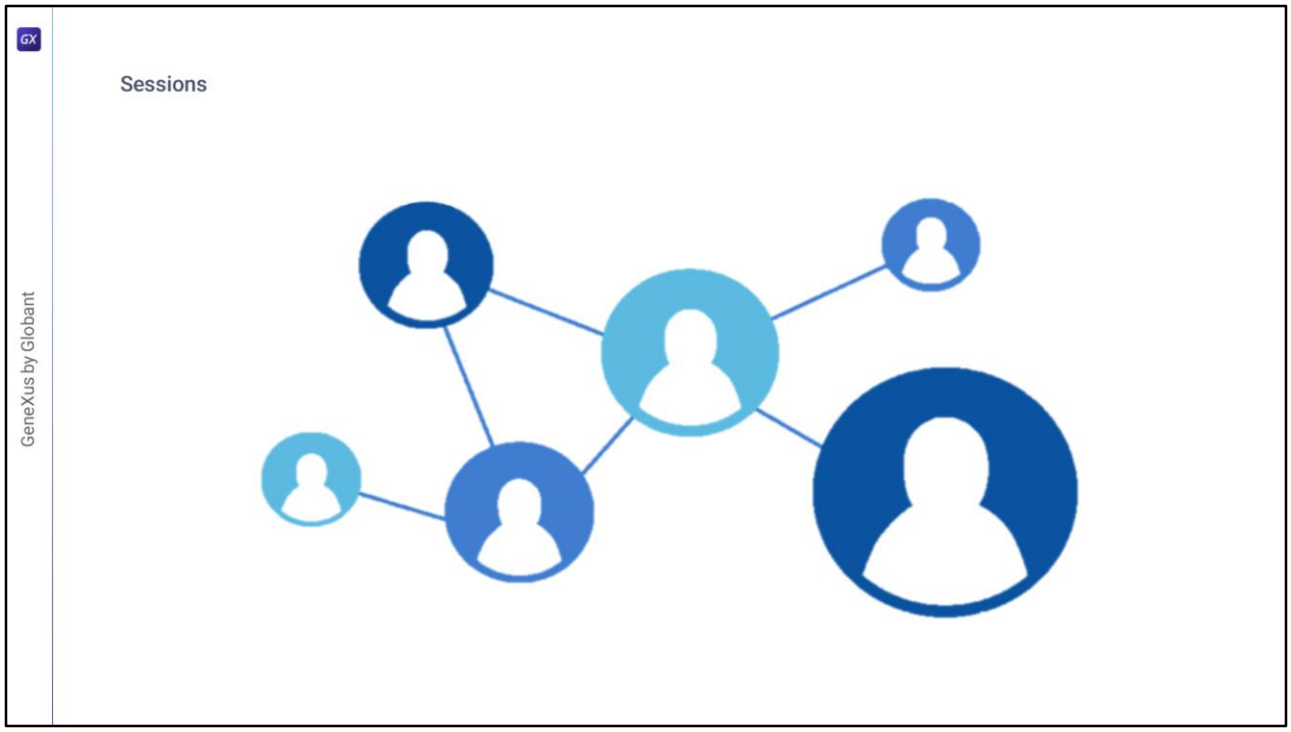
As permissões, como dissemos anteriormente, descrevem a possibilidade de realizar operações sobre objetos.
Estas podem estar associadas a uma ou mais roles.

A permissão tem um tipo de ação predeterminada que especifica se a permissão está restrita ou não: Allow ou Restricted. Por padrão, está habilitado para todos os usuários.

As permissões são por aplicação e, ao atribuí-la a uma Role, é possível especificar sua ação para essa Role específica. As opções são Allow, Deny ou Restricted.
Quando é atribuído Allow, é uma permissão positiva. Se quisermos negar uma permissão, temos as outras duas opções: Deny ou Restricted.

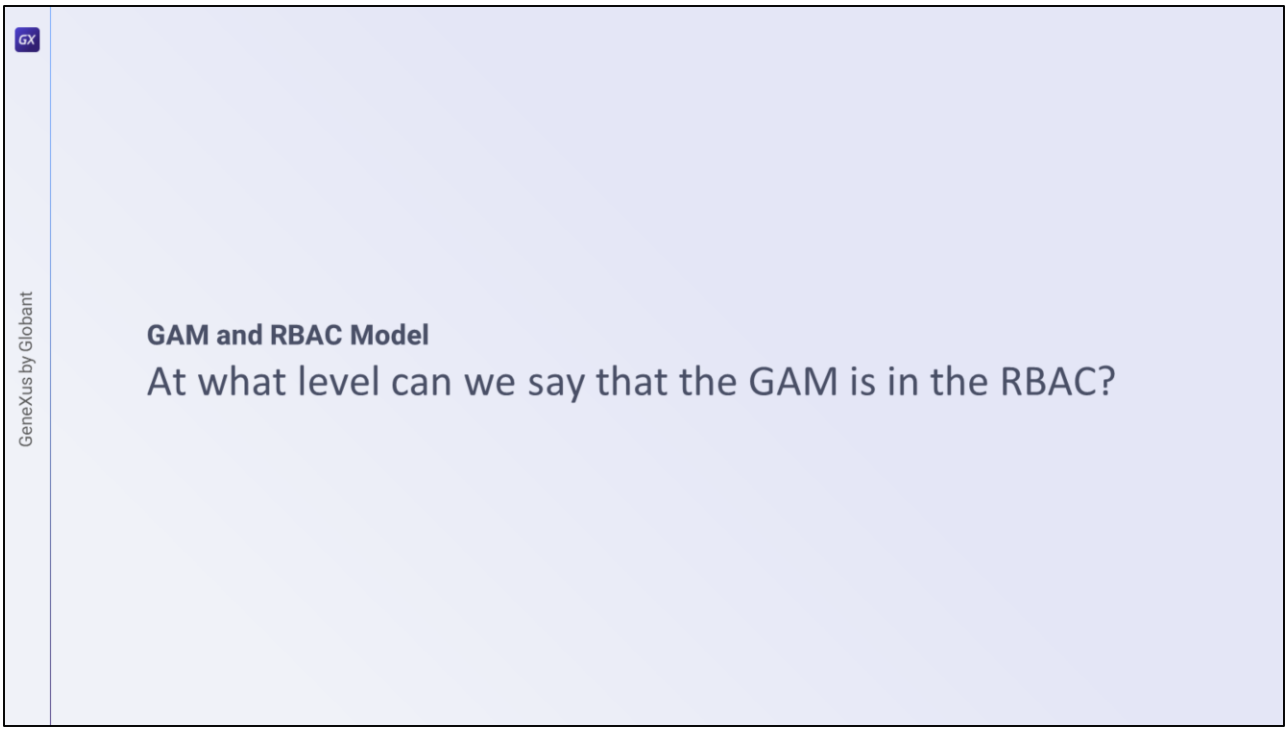
No momento de criação de uma sessão para um usuário, GAM obtém as roles associadas a ele. Para verificar se um Usuário tem uma permissão válida, GAM verifica todas essas Roles: Se uma delas tiver uma permissão Restricted e outra tiver a mesma permissão Allow, ganha a Allow, mas se uma delas tiver uma permissão Deny e outra tiver a permissão Allow, ganha a Deny, onde inclusive esta é mais forte que a Restricted.

Finalmente, a permissão é atribuída diretamente a um usuário.

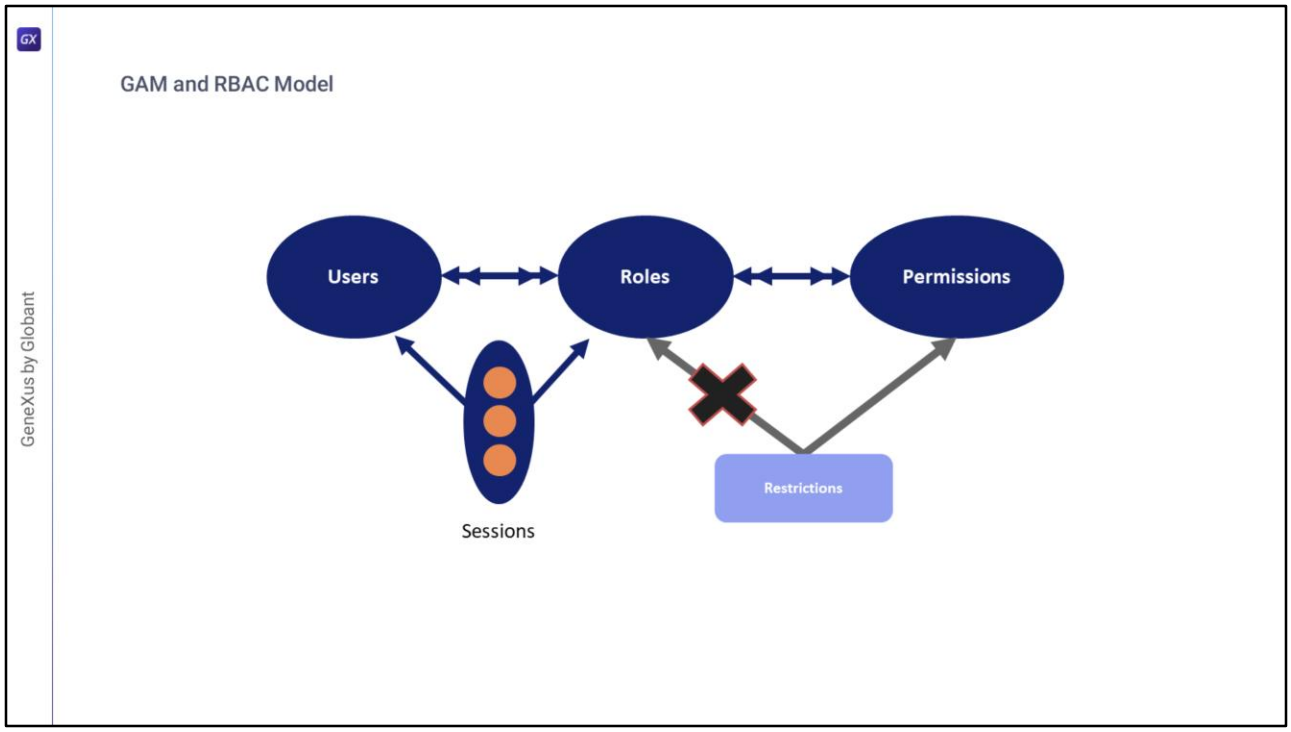


Assim como no RBAC temos as Sessões, que realizam o mapeamento entre os usuários e suas roles ativas.

Conceitualmente, são tratadas exatamente da mesma forma que as descrevemos no RBAC.
Cada sessão corresponde a cada usuário e é tratada automaticamente pelo GAM. Contêm um token de identificação e expiram de acordo com as políticas de segurança definidas.

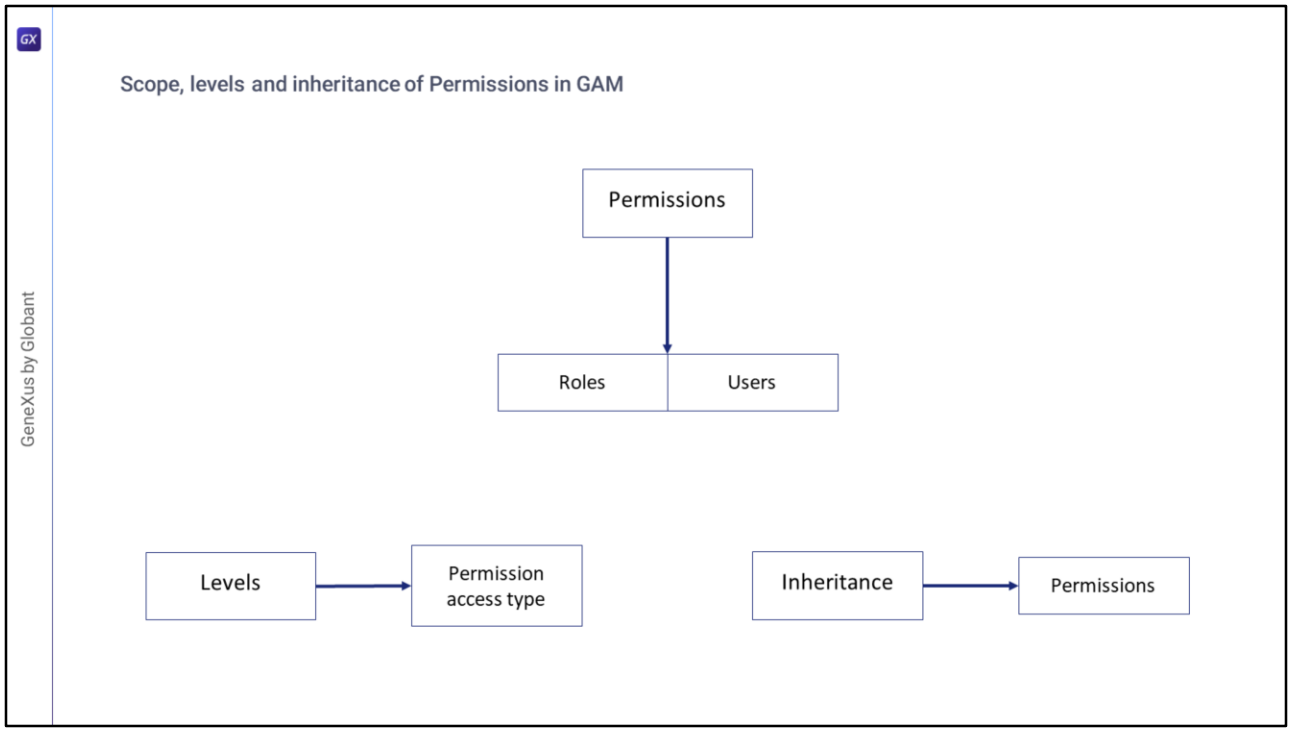


Em que nível podemos dizer que está o GAM no RBAC?



O GAM está localizado no nível 3, onde são aplicados os mecanismos de separação por roles, com determinadas permissões para cada usuário, que se nos recordamos, era o que correspondia a este nível.

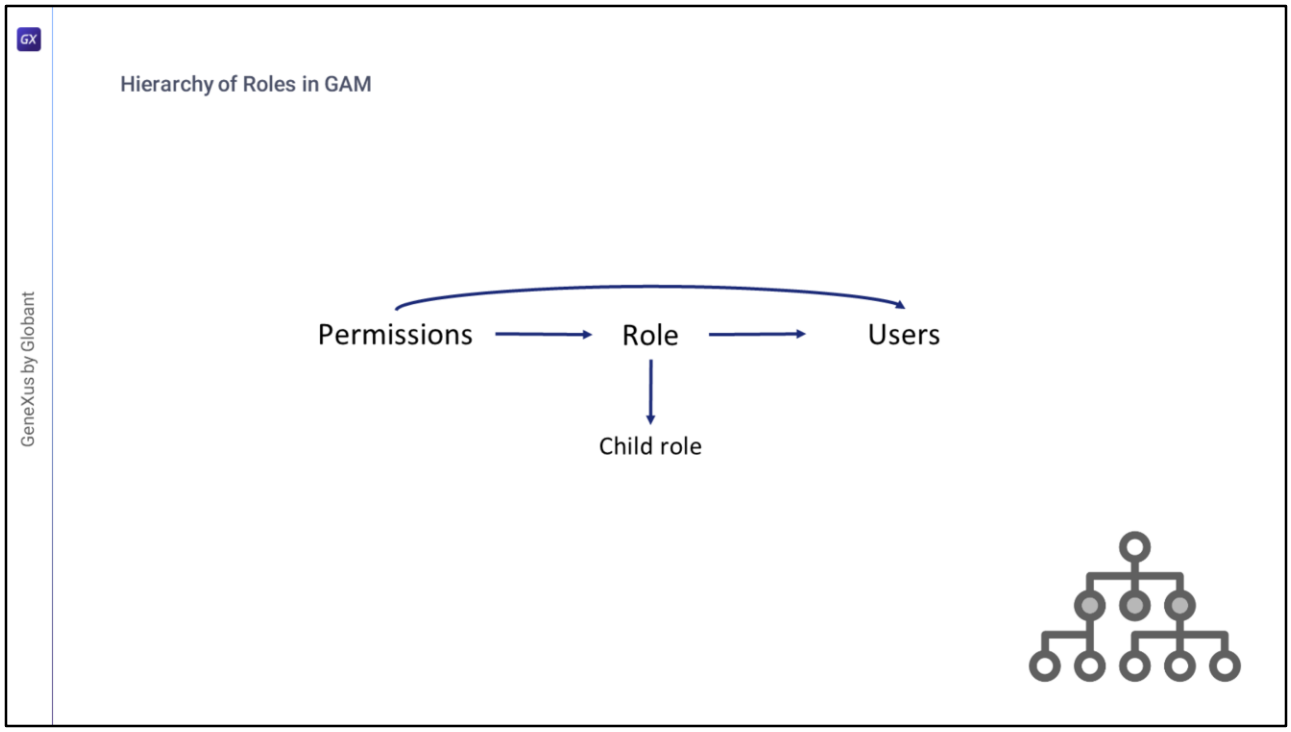
No caso das Restrições, embora existam restrições no nível de permissões, não existem no nível de roles.



As Permissões no GAM existem no âmbito das Aplicações e são atribuídas a Roles e a Usuários no Repositório GAM.

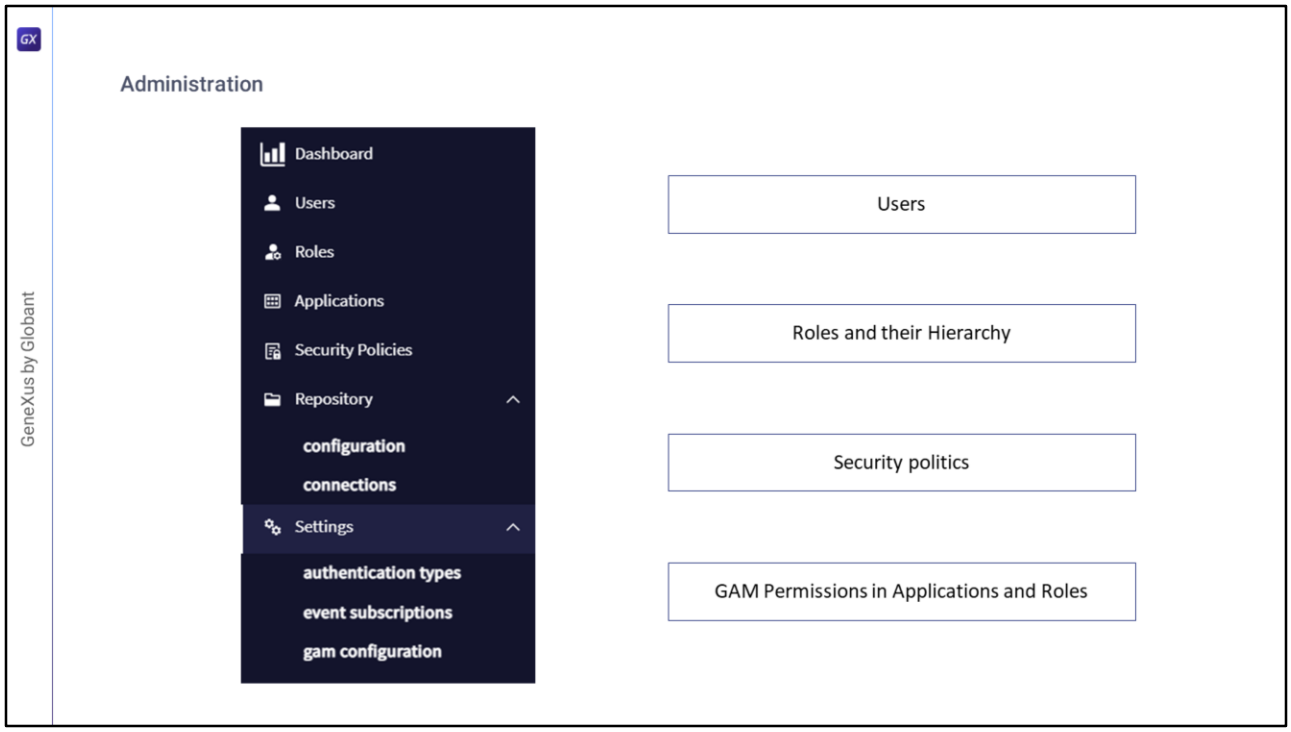
Quanto aos níveis, o nível de permissões que tem um usuário em tempo de execução depende do tipo de acesso de permissão.

Finalmente, quando é adicionada qualquer permissão de controle total a uma role, por padrão, todas as permissões secundárias deste controle total também são adicionadas à role, a menos que tenha sido perdida a herança.



Conforme explicado anteriormente, se uma role está associada a um usuário, as permissões da role estão associadas indiretamente com esse usuário. Isto significa que quando são verificadas as permissões de usuário em tempo de execução, são levadas em consideração as permissões associadas ao usuário através das roles.

Além disso, se a role tem filhas, as permissões das roles filhas também são associadas ao usuário.



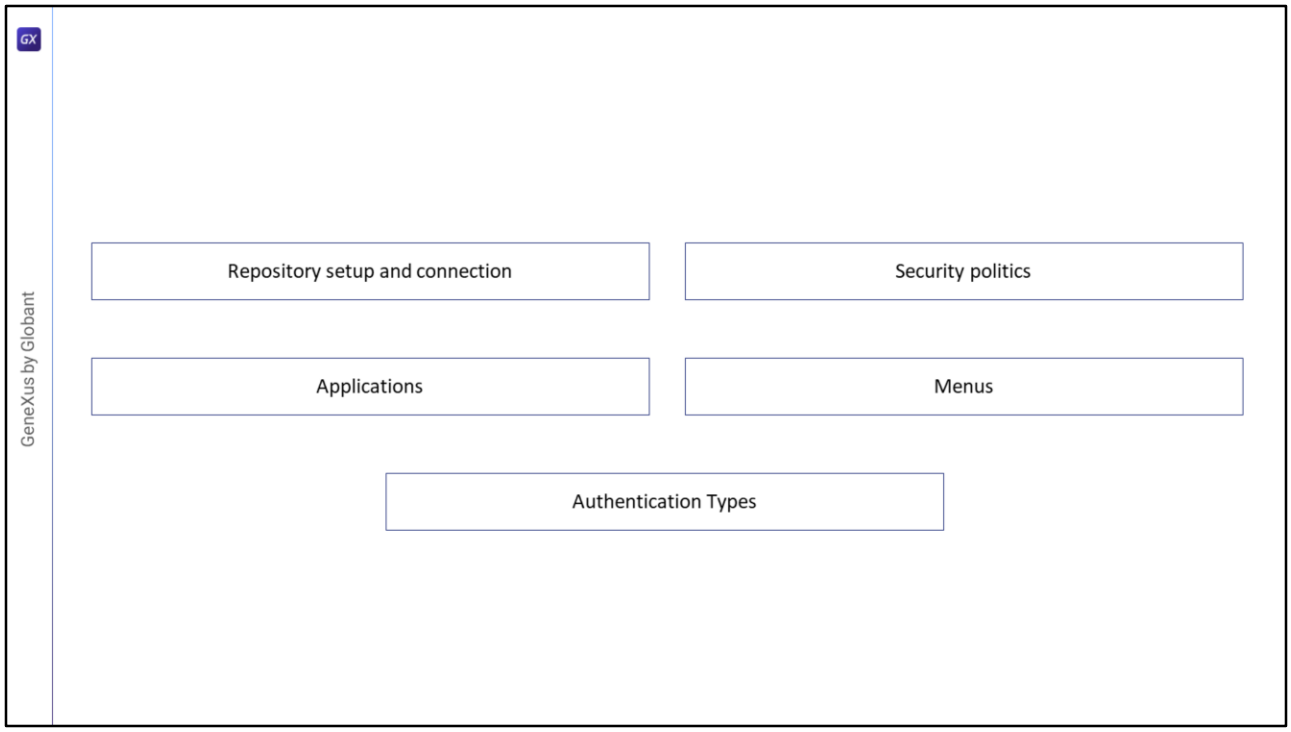
O backend do GAM é uma aplicação web que permite ao administrador do GAM gerenciar todo o sistema de forma completa.
Entre estas funções, encontra-se tudo o que mencionamos anteriormente relacionado ao RBAC:

- Usuários
- Roles e sua hierarquia
- Políticas de Segurança
- Permissões GAM em Aplicações e Roles

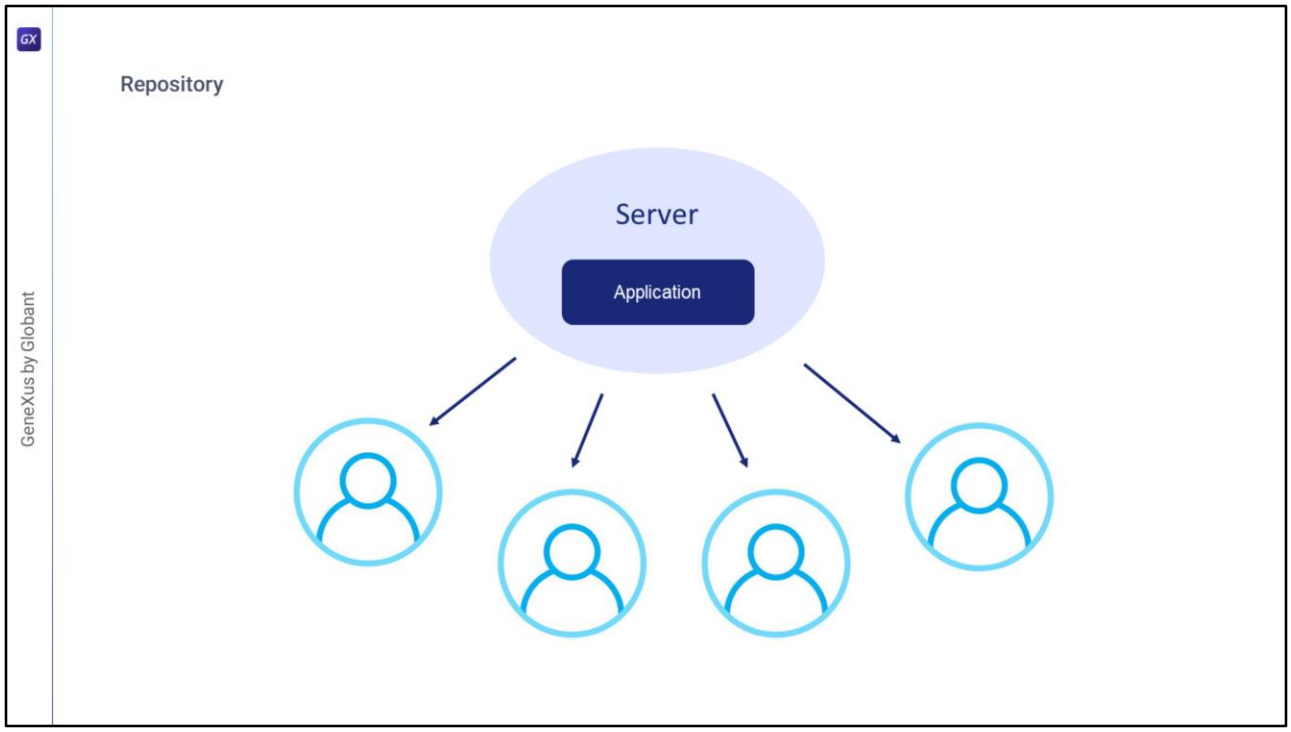
Por sua vez, também oferece a possibilidade de gerenciar as aplicações, configurar o repositório e suas conexões, alterações de senha, tipos de autenticação, entre outros, que veremos a seguir…

GeneXus by Globant

Other fundamental concepts of GAM



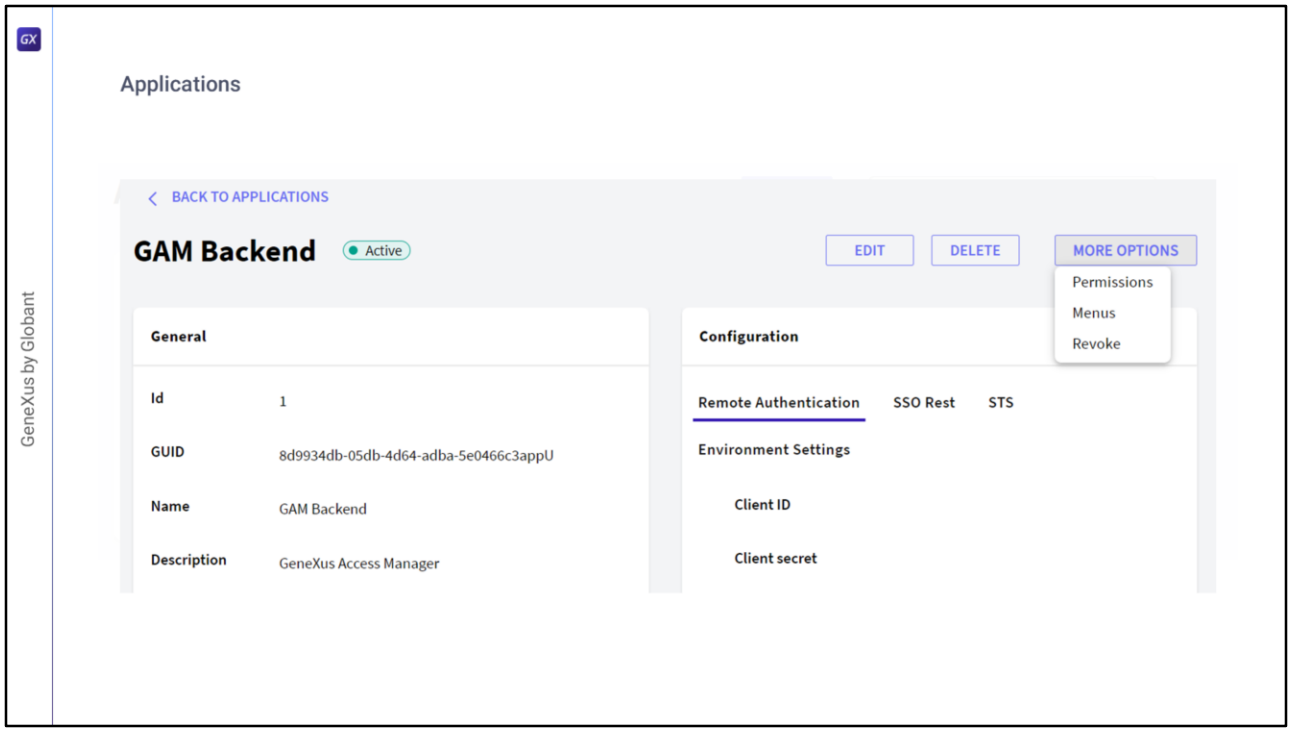
Além dos conceitos mencionados até o momento, GAM oferece a possibilidade de gerenciar mais opções, como a configuração e conexão do repositório, suas aplicações e políticas de segurança, os menus e os tipos de autenticação existentes.



Primeiro vejamos o Repositório.

Um repositório é uma entidade do GAM que suporta uma arquitetura onde uma única instância da aplicação é executada em um servidor e atende a múltiplos usuários.

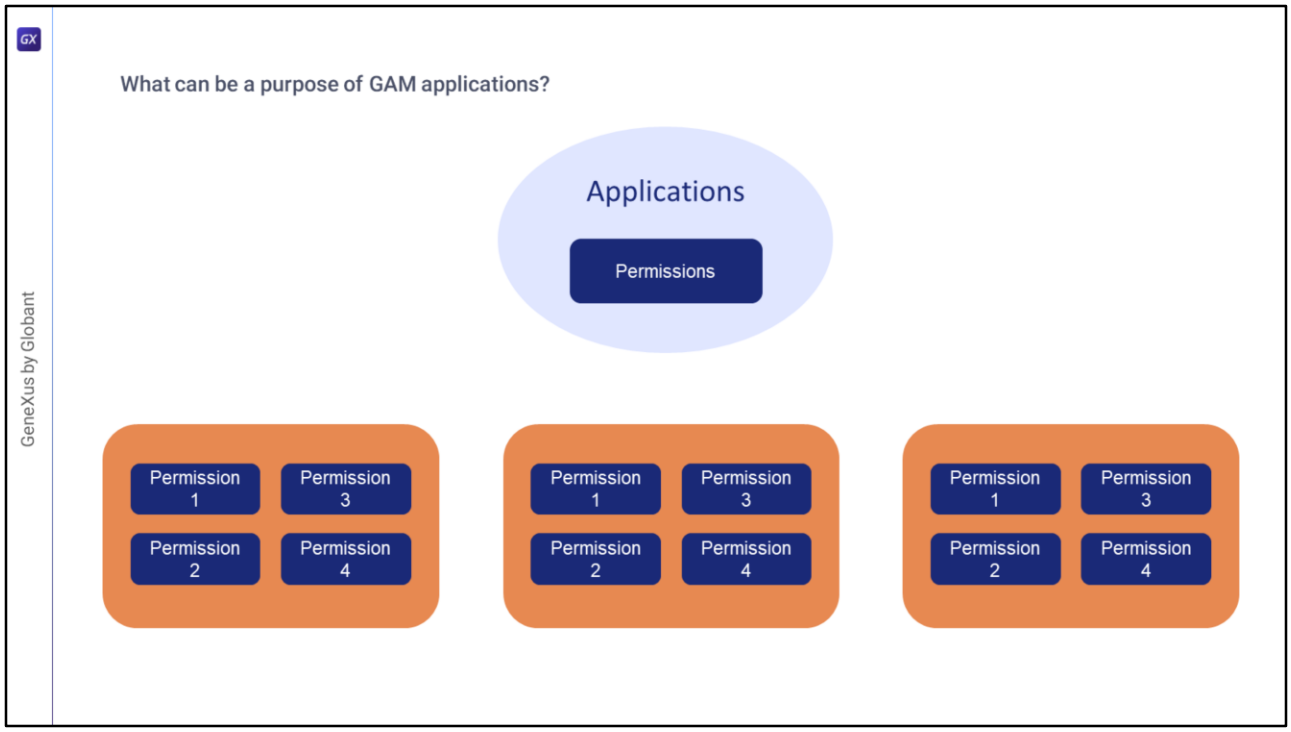
Neste cenário, estes usuários devem usar a mesma base de dados GAM e diferentes repositórios GAM dentro da base de dados. Isto é comumente chamado de aplicação multiusuário e permite a cada usuário um conjunto exclusivo de roles, aplicações e políticas de segurança do GAM.



O que são as aplicações GAM?
GeneXus possui um ambiente web, e conta com uma aplicação web GAM, que pode ser identificada mediante um GUID.
Além disso, a aplicação tem um nome (o nome da KB) e inclui as permissões de todos os objetos web da KB

Em relação às Aplicações móveis:
As aplicações GAM Mobile agrupam as permissões de todos os objetos mobile principais da KB. Cada objeto principal destes, determina a existência de uma aplicação GAM.

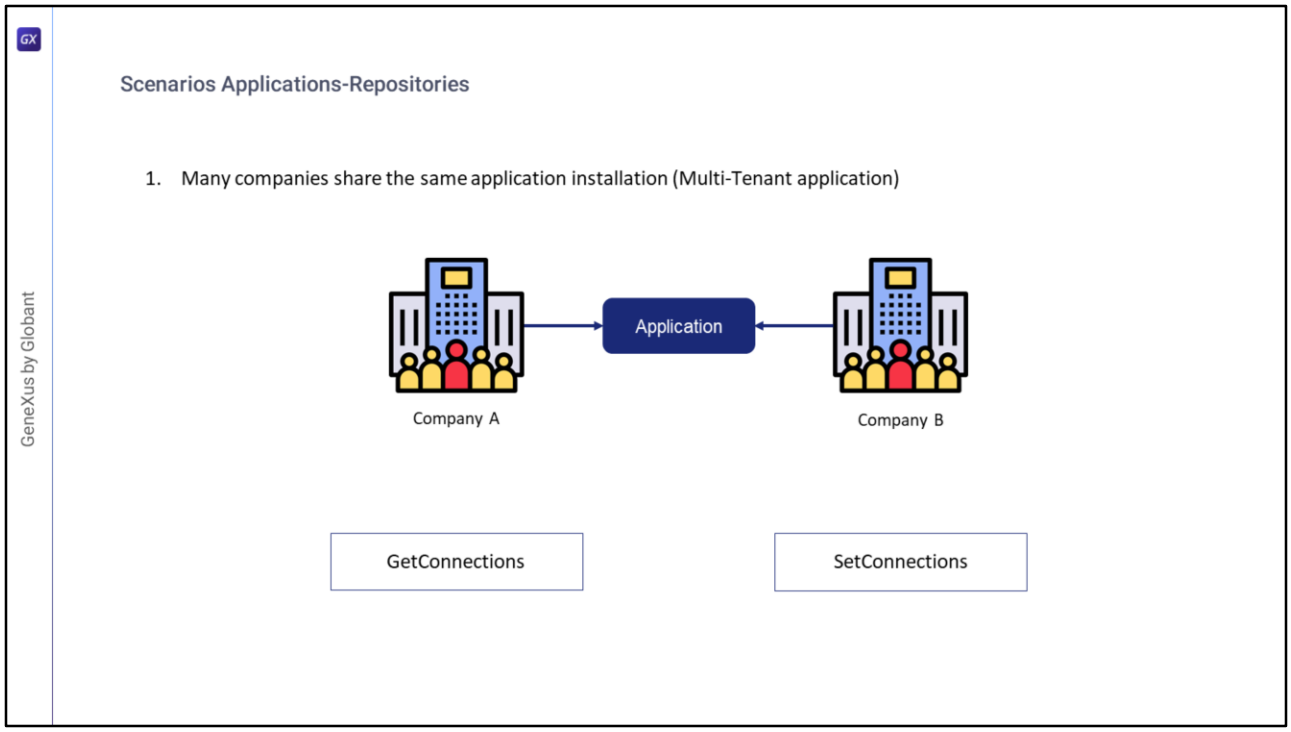
As aplicações GAM são definidas dentro de um repositório e como já dissemos, cada um deles pode conter mais de uma aplicação.



Qual pode ser um propósito das aplicações GAM?

Um deles pode ser associar permissões a estas aplicações e formar grupos de permissões.

Como as permissões são verificadas tendo em conta a aplicação que está sendo executada, então, quando o usuário faz login em um repositório e é necessária uma permissão para executar uma ação, a permissão deve estar definida na aplicação GAM que está executando (e deve ter uma role onde seja autorizada essa permissão).



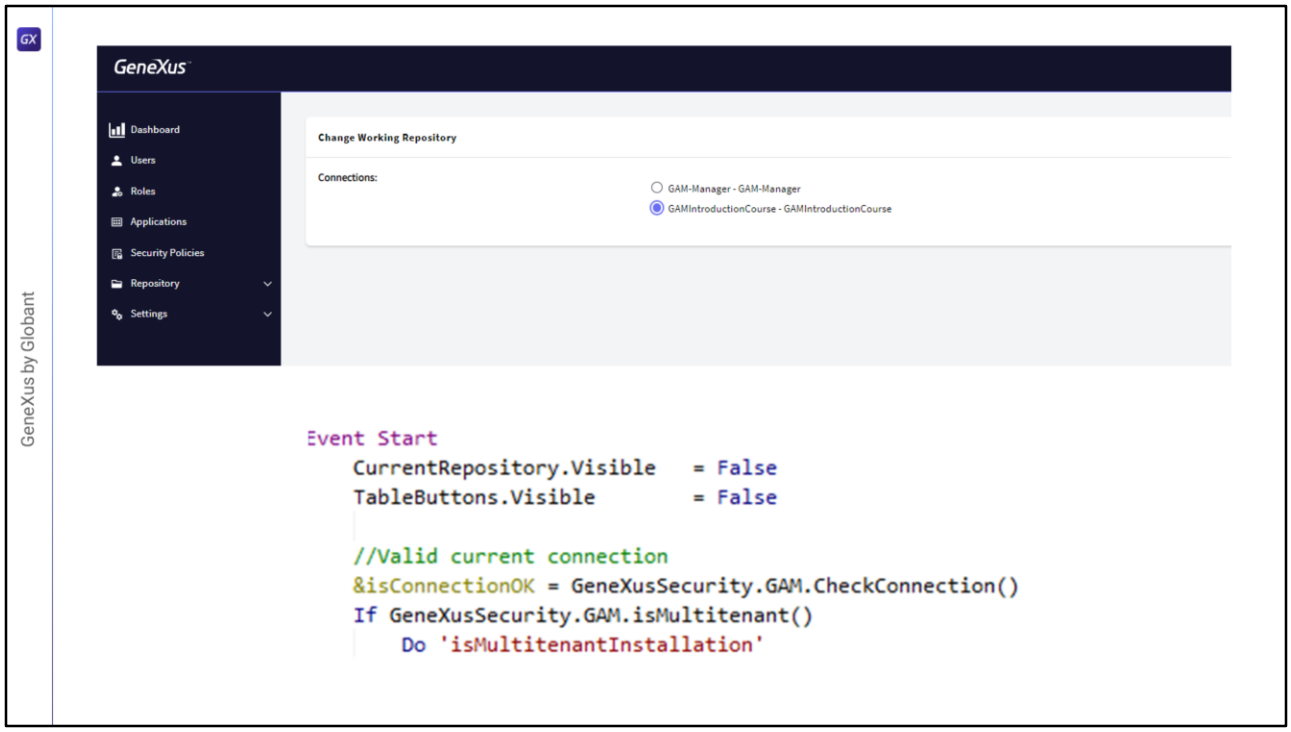
Vamos colocar exemplos de cenários com Aplicações e Repositórios.

O primeiro cenário é composto por muitas empresas que compartilham a mesma instalação da aplicação (o que é definido como aplicação Multi-Tenant)
Como dissemos antes, o desenho do modelo do GAM permite conectar-se a múltiplos Repositórios para resolver muitos cenários em que um único Repositório GAM não seria suficiente.
Neste caso, cada Repositório terá seus próprios usuários administradores, e a informação de um Repositório não será acessível a partir dos demais.

É o caso de uma aplicação multiusuário em que uma única instância do software é executada em um servidor, atendendo a múltiplas organizações de clientes.

Colocando este cenário em prática, temos dois métodos de GAM para poder utilizar: **GetConnections** e **SetConnections**.
O método GAM.GetConnections retorna em uma coleção uma lista de conexões que contém a chave armazenada no arquivo connection.gam.
O método GAM.SetConnection retorna verdadeiro se a conexão foi estabelecida corretamente. Isto significa que todos os métodos GAM acessarão o novo conjunto de repositórios.

Vejamos um exemplo disto.



Antes de aprofundar nele, vejamos um conceito que é utilizado na implementação disto e é o GAM Manager Repository.
Este é um Repositório particular que é utilizado para gerenciar os demais Repositórios, e são os usuários deste Repositório, em conjunto com os usuários de repositórios que tenham habilitada a propriedade de Manager, os únicos que podem criar novos Repositórios e gerenciá-los.

Não entraremos em detalhes neste tema, pelo que recomendamos consultar a documentação dele na Wiki de GeneXus para compreendê-lo a fundo. Nós simplesmente assumiremos que já o temos criado e configurado.

Podemos mostrar isto com um dos exemplos que nos fornece GAM. Neste caso, utilizaremos GAMExampleLogin.
Vemos que no evento Start, é verificada a conexão atual com o Repositório, e se estamos perante um caso de Multitenant, é realizada uma chamada a uma sub-rotina.

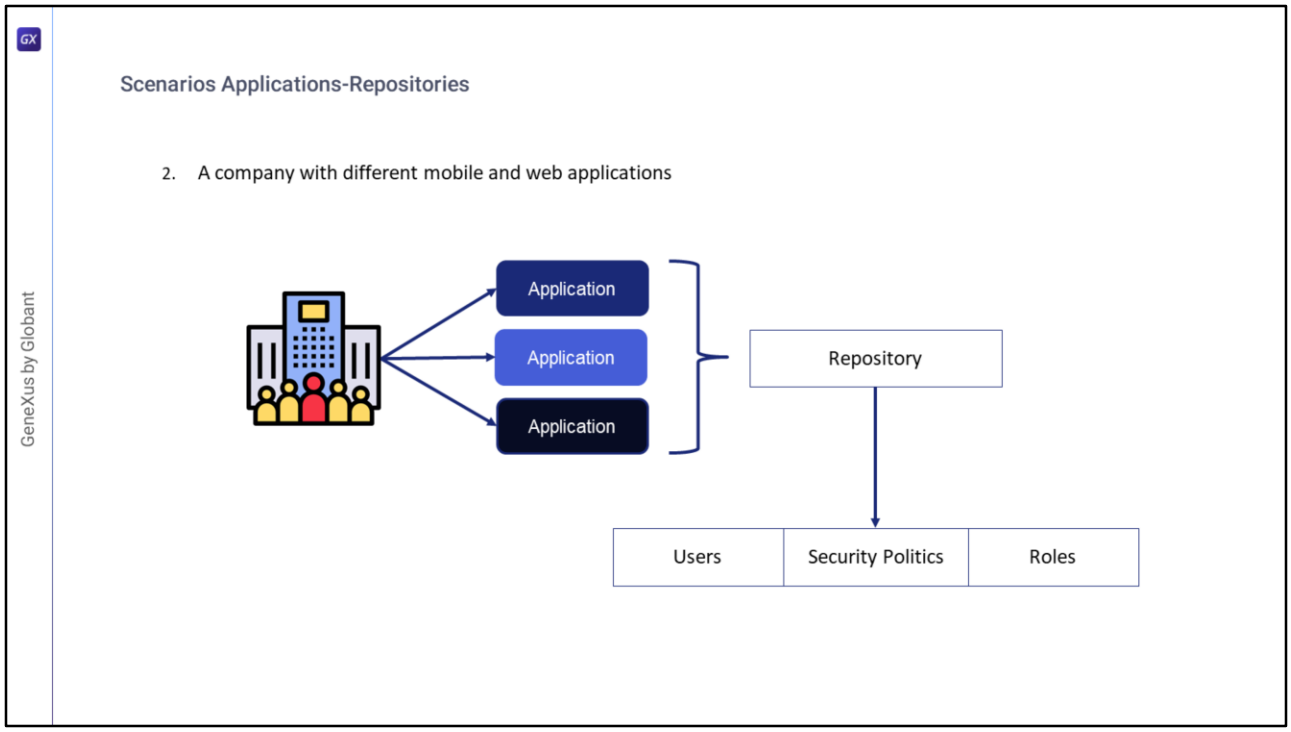
```
Sub 'isMultitenantInstallation'
    //Read Curent Repository
    &GAMRepository = GeneXusSecurity.GAMRepository.Get()
    //Check if the current repository uses an authentication master repository
    If not &GAMRepository.AuthenticationMasterRepositoryId.IsEmpty()
        &isConnectionOK = GeneXusSecurity.GAM.SetConnectionByRepositoryId(&GAMRepository.AuthenticationMasterRepositoryId, &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
```

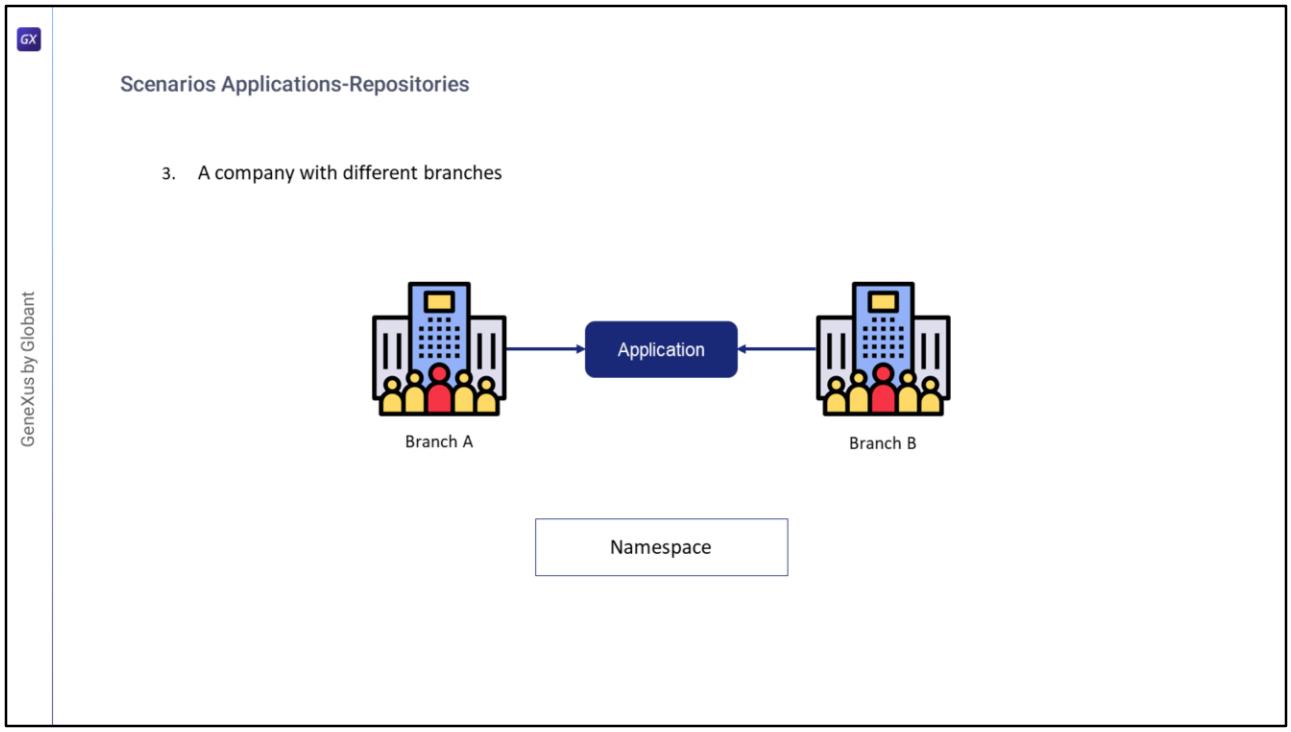
Na sub-rotina, vemos que a primeira coisa que é trazida é o repositório atual, para depois verificar se é master ou não e conectar-se.
Se a conexão não foi bem-sucedida, pergunta se o repositório atual é o padrão para configurar uma conexão com ele. Caso não seja, utiliza os métodos que vimos anteriormente: GetConnections e SetConnections.
Primeiro faz o Get obtendo a lista de conexões, e fica com a primeira.

Vamos ficar com essa parte do código. Neste caso, fica com o primeira, pois é um exemplo de implementação básica. Em um caso personalizado, o desenvolvedor pode escolher a qual repositório deseja se conectar de acordo com as condições que queira ou deseja expor.
Lembremos que estas implementações são de exemplo e guia para que realizem suas próprias implementações.

Com esses dois métodos, temos todo o controle das conexões aos repositórios. O resto do código já não vem ao caso.



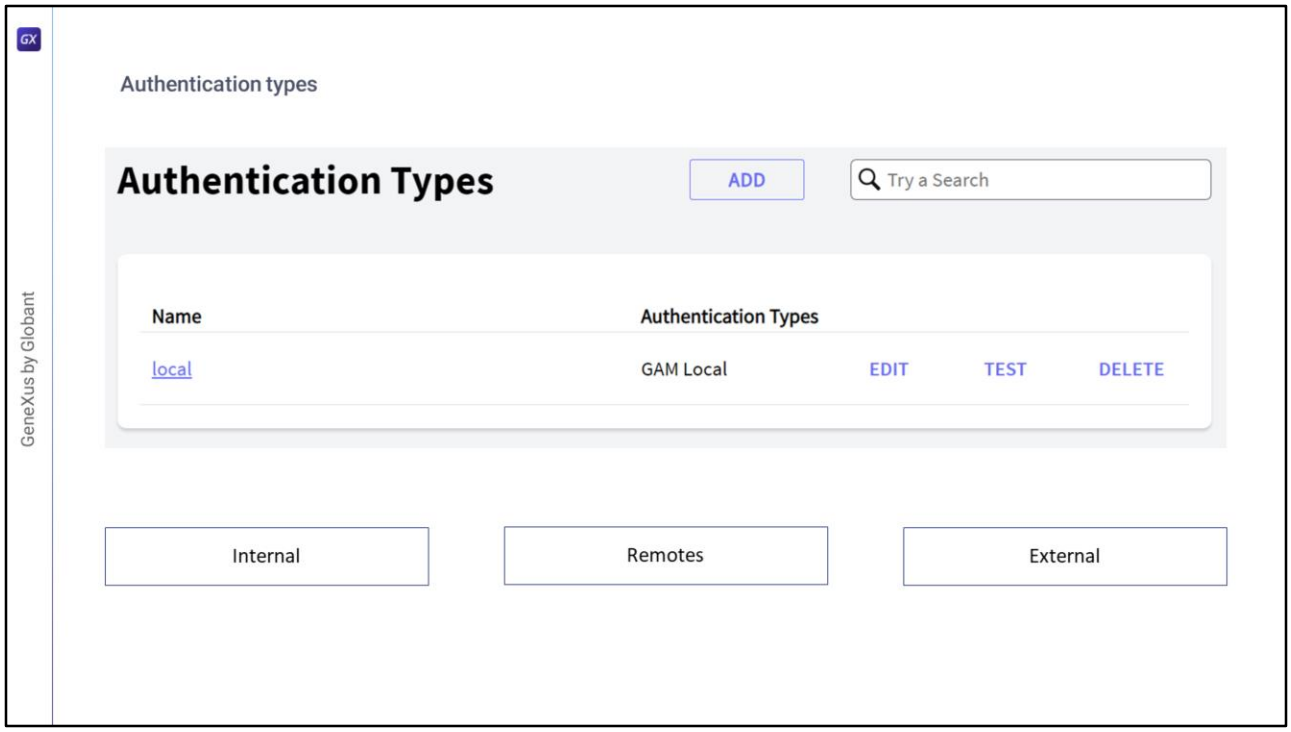
No segundo cenário temos uma empresa com diferentes aplicações móveis e web
Neste cenário de uso, a empresa possui diferentes aplicações, e são compartilhados os usuários, políticas de segurança e roles, tendo um único repositório.
Além disso, cada aplicação contém suas próprias permissões.



No terceiro e último cenário, temos uma empresa com diferentes filiais.
Neste cenário de uso, a empresa possui uma única aplicação que é usada por todas as diferentes filiais que existem na empresa. Ou seja, cada filial é representada por um repositório.
Ao contrário do cenário 1, neste caso os usuários são os mesmos para todas as filiais, embora tenham a ressalva de que cada um pode ter diferentes políticas de segurança, roles e permissões, dependendo da filial da aplicação à qual estão conectados.

Dado o desenho do modelo de repositórios no GAM, os usuários poderiam ser habilitados em muitos repositórios se tiverem o mesmo namespace que o do repositório no qual estão habilitados.
O Namespace dos Repositórios GAM, tem como finalidade agrupar em um contêiner abstrato todos os Repositórios que pertencem a uma mesma Empresa.

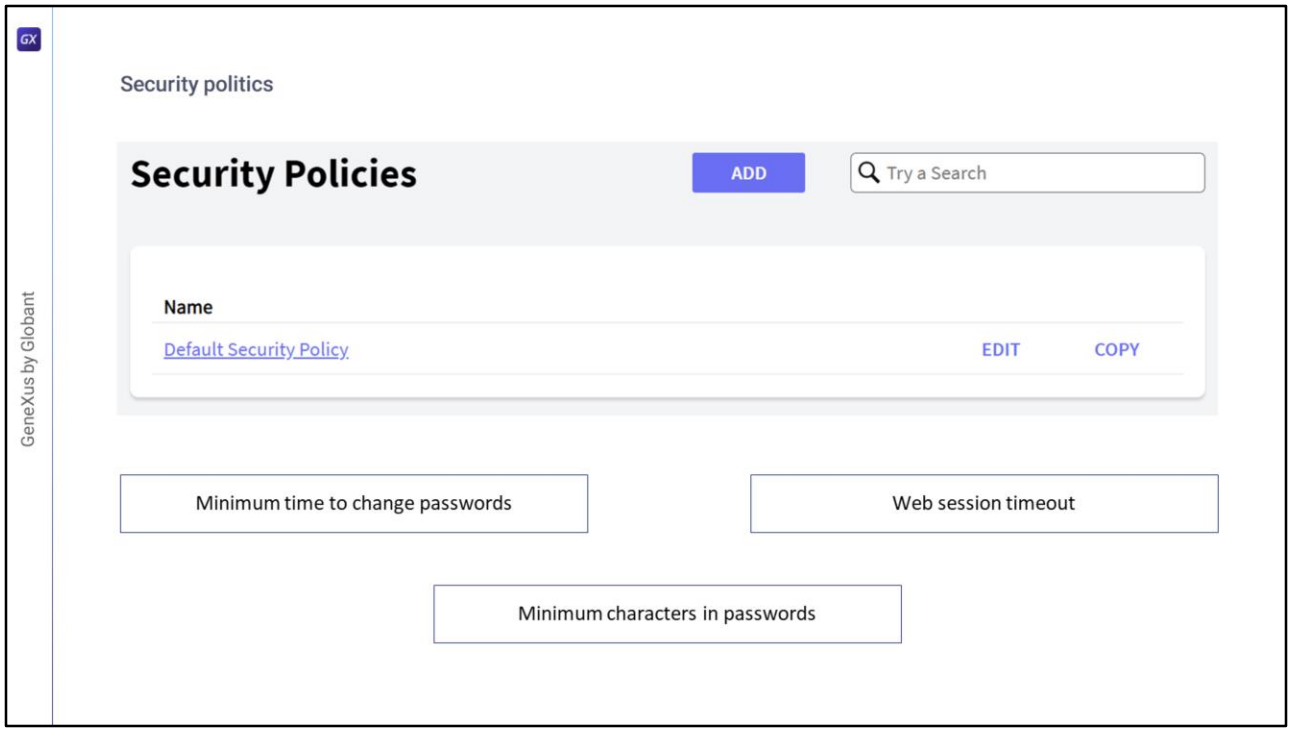
Algo a destacar é que os três cenários são resolvidos com apenas uma base de dados, sem necessidade de criar outras para cada filial ou aplicação.



Tipos de Autenticação.
A autenticação é o ato ou processo de confirmar que algo (ou alguém) é quem diz ser.

GAM oferece diferentes tipos de autenticação, sejam internos, remotos e externos (como pode ser serviços web, redes sociais ou Google).

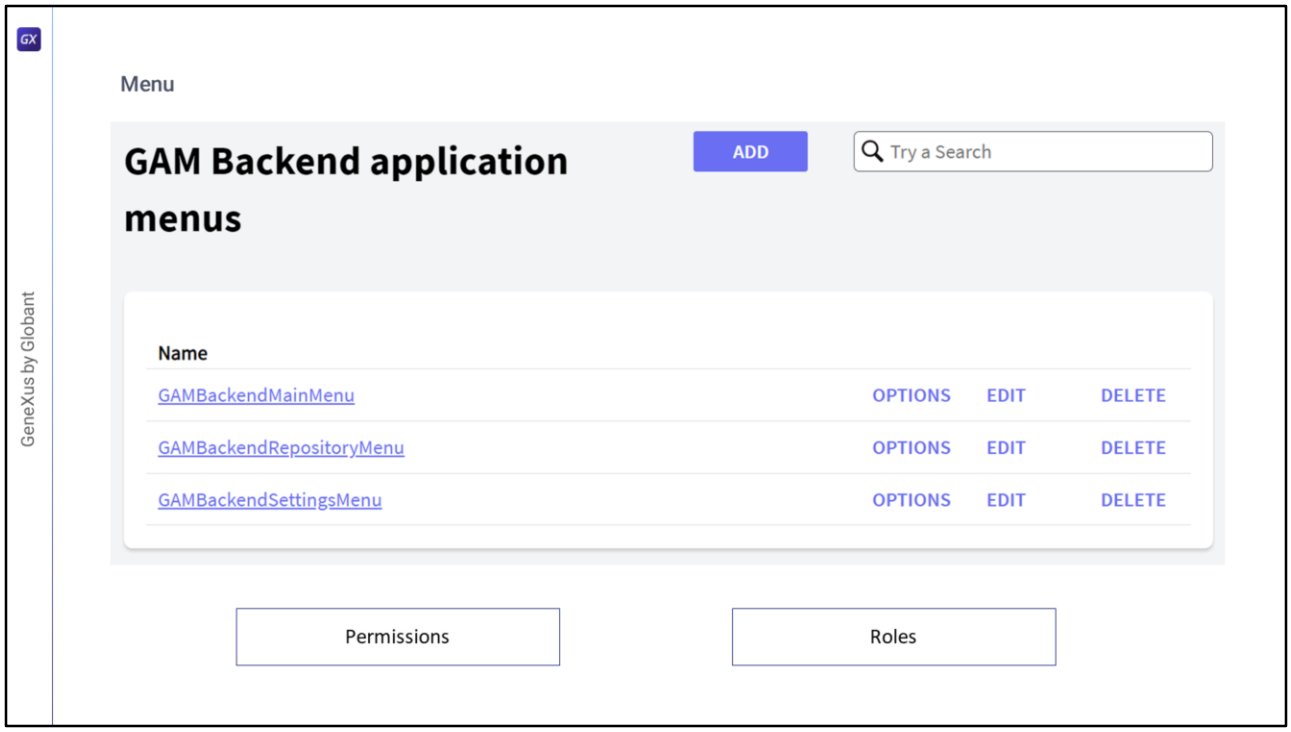
Isto veremos com detalhes mais adiante no curso.



As políticas de segurança são um conjunto de regras, normas e protocolos de atuação que se encarregam de zelar pela segurança das roles e usuários no GAM.

Podem ser definidas utilizando o Backoffice web do GAM ou programaticamente usando a API do GAM.

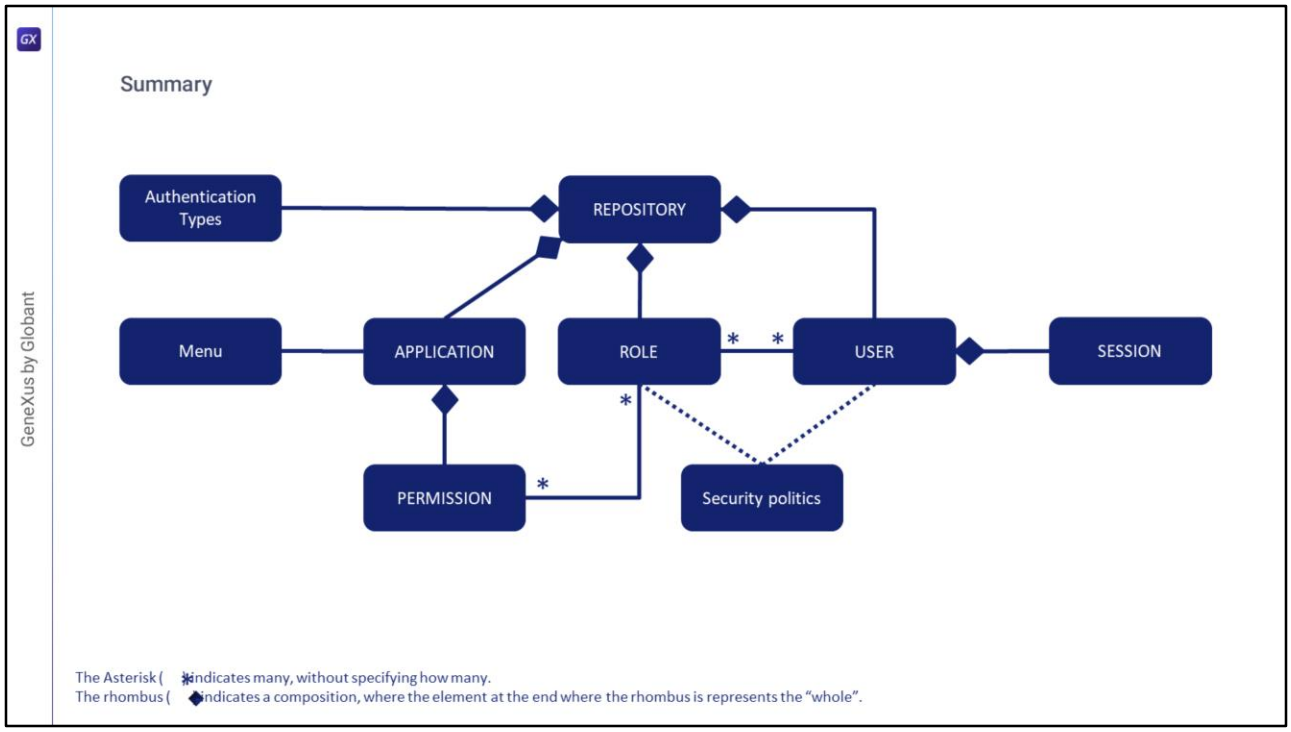
Entre as que utiliza GAM, podemos destacar o Tempo mínimo para trocar senhas, o Tempo limite da sessão web, o Mínimo de determinados caracteres em senhas (como são os caracteres numéricos, especiais, maiúsculos e minúsculos), entre outros.



Utilizando GAM, é possível definir (dinamicamente) um menu de sua aplicação web ou móvel em tempo de execução, com base nas permissões e roles GAM do usuário logado.

GAM retorna a estrutura do menu, dependendo das permissões do usuário, para que esta estrutura possa ser carregada em tempo de execução com qualquer Controle de usuário.

O menu deve ser definido para uma aplicação GAM em particular e podem ser definidos quantos menus quiser dentro de uma aplicação GAM.
Uma opção de menu terá uma permissão e um recurso associado.



Resumindo tudo o que vimos até agora, no diagrama que vemos na tela, temos todos os componentes que mencionamos do GAM.

Antes de aprofundar no diagrama, em relação à nomenclatura utilizada, como se indica no slide, o asterisco indica muitos, e o losango indica uma composição, onde faz referência a que, por exemplo, não podem existir sessões se não houver um usuário que a contenha.

Agora, indo para o conteúdo do diagrama, podemos detalhar que o conjunto de Permissões GAM se enquadra no escopo de uma única Aplicação e Repositório. Portanto, as Aplicações GAM podem ser associadas a *n* Permissões GAM, e estas são determinadas por uma Aplicação em um Repositório.
Então, as roles são definidas dentro de um Repositório e estão associadas a Permissões onde estas podem ser usadas em muitas Roles.
Os usuários podem ser habilitados em um ou mais repositórios do GAM e são o conteúdo das sessões. Além disso, podem ter muitas roles e indiretamente atribuir-se uma ou mais permissões.

Os Tipos de Autenticação e Políticas de Segurança estão dentro do escopo de um Repositório. Os menus estão dentro do escopo da aplicação, que por sua vez, estas estão dentro do escopo de um Repositório


GeneXus by Globant
GeneXus™
by Globant
training.genexus.com