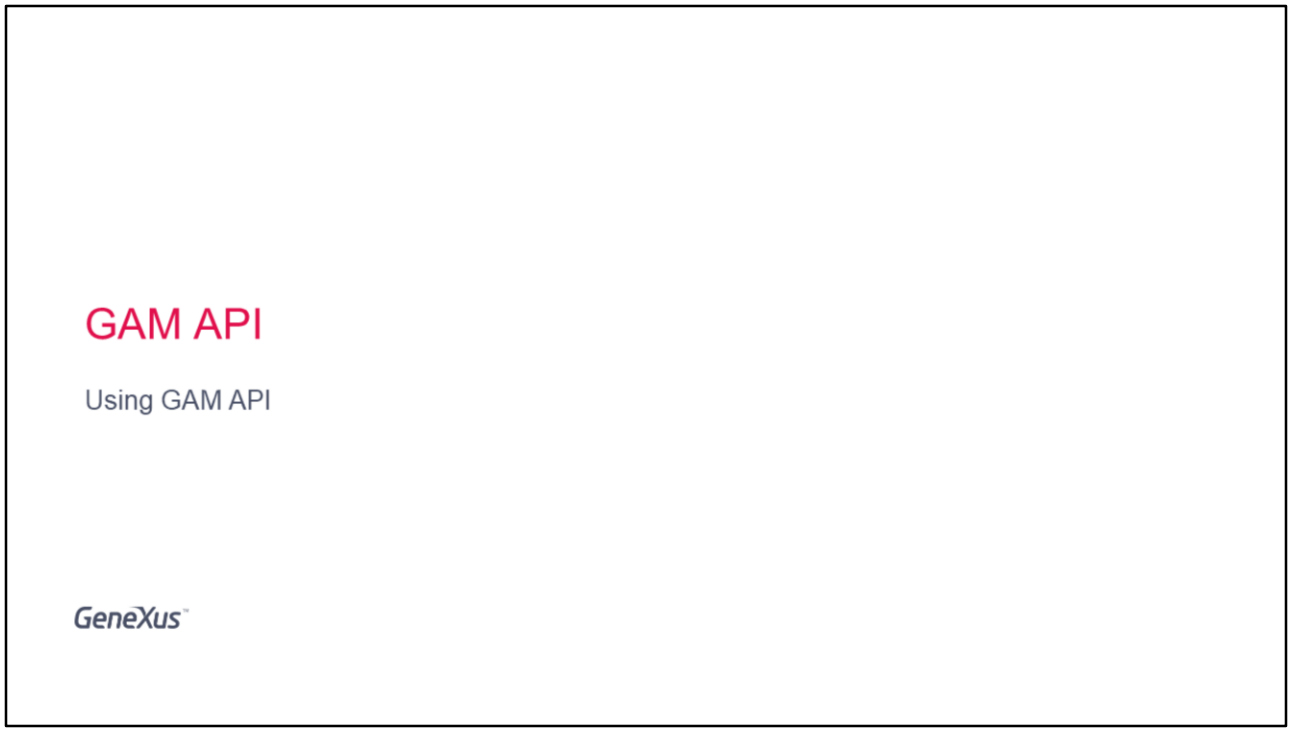

GeneXus™
by Globant

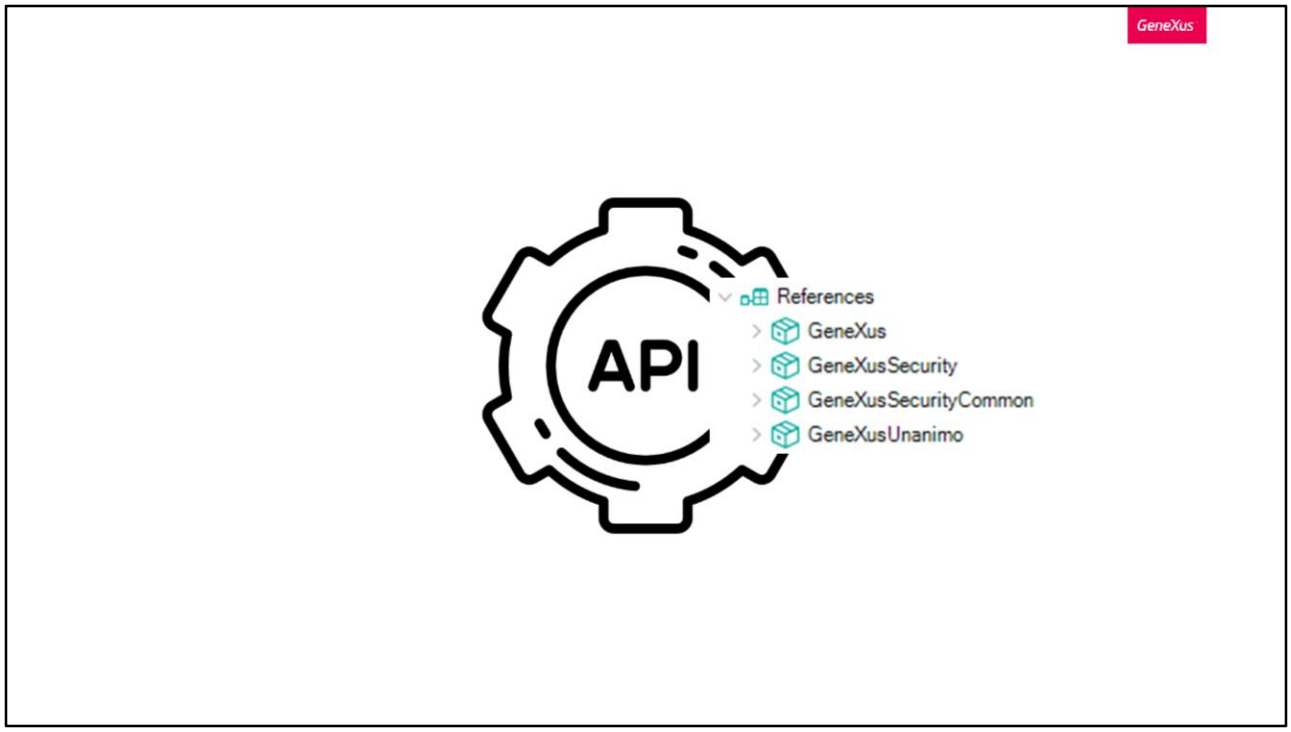
API

Nicolas Adrién

GeneXus



Neste vídeo explicaremos em que consiste a API do GAM, com seus métodos e implementações de referência, tipos de autenticação, serviços REST, entre outras.



GAM oferece uma API que permite aos usuários lidar com diferentes tipos de dados e métodos para adicionar segurança às aplicações GeneXus, tanto aplicações Web como aplicações Móveis.

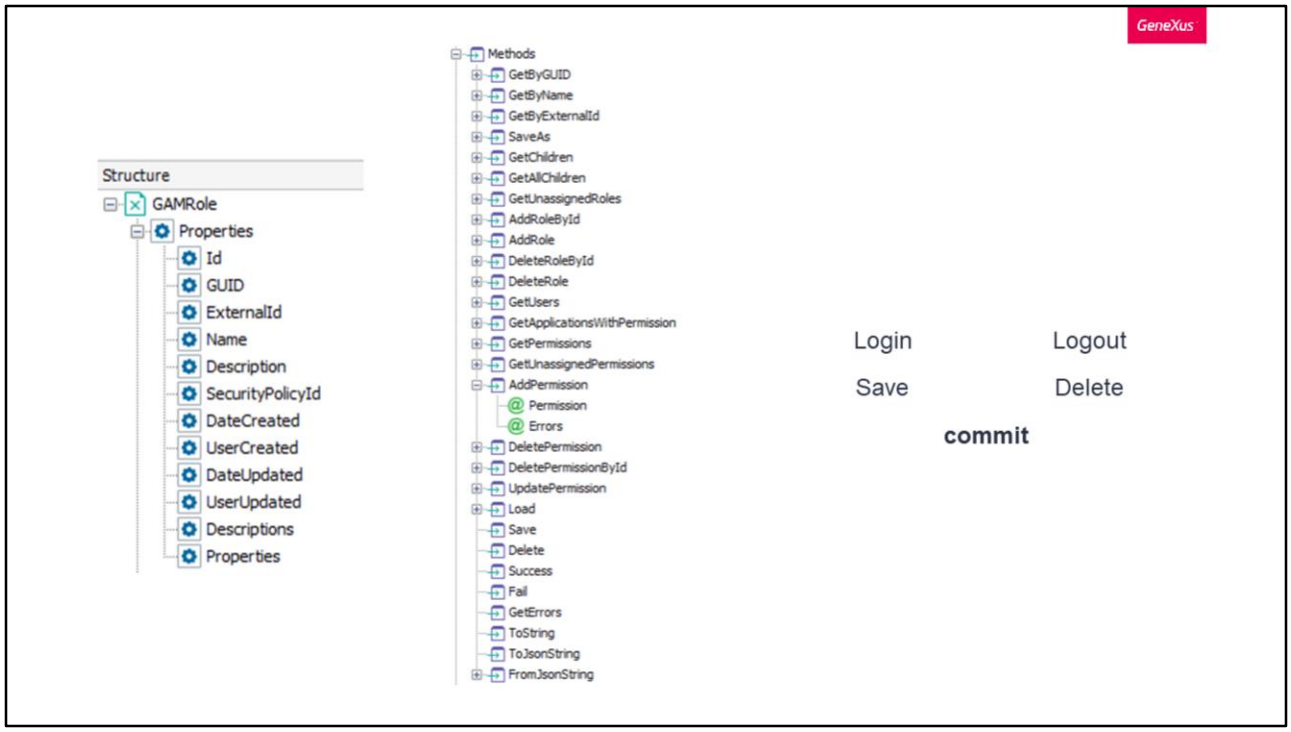
Quando a segurança integrada é habilitada na KB, são incorporados determinados External Objects para permitir que o usuário interaja com a API do GAM, onde estes objetos fazem parte dela.
Todos eles podem ser encontrados no módulo chamado GeneXusSecurity. Seus domínios estão distribuídos no módulo GeneXusSecurityCommon.

No curso introdutório, vimos a lista desses External Objects fornecidos pelo GAM e exemplificamos o uso de um. Agora vamos ver isto em detalhes.

Estes objetos possuem propriedades e métodos e, por sua vez, alguns deles implementam os mesmos métodos que os Business Component, como Load, Save, Delete, Fail e Success.
Se for alterada alguma propriedade nesses objetos GAM, deve ser chamado o método Save() e executar o comando commit.

Os objetos que se comportam como Business Component são: GAMRepository, GAMApplication, GAMUser, GAMSecurityPolicy, GAMEventSubscription e todos os tipos de autenticação.

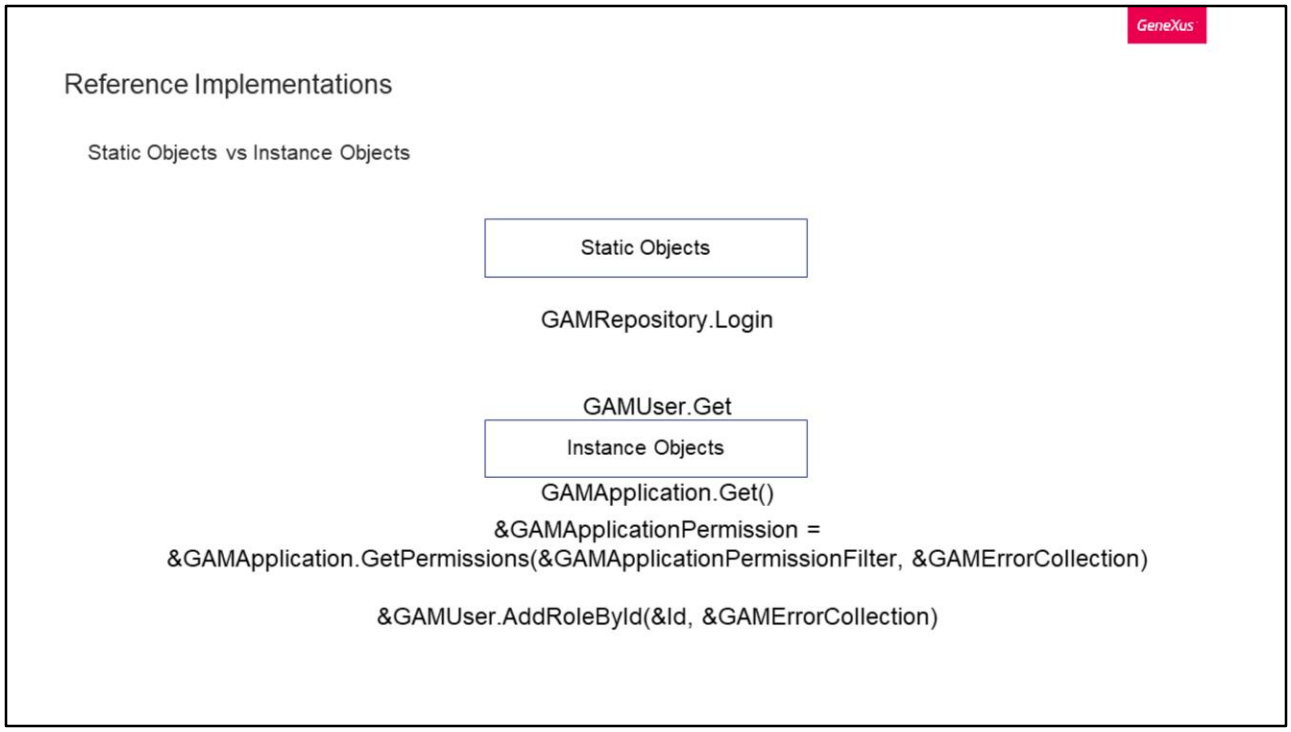
Como dissemos antes, todos esses objetos podem ser encontrados no módulo GeneXusSecurity.



Como mencionamos, estes objetos são compostos de propriedades e métodos.

Por sua vez, os objetos GAM também possuem outros métodos implementados para criar, atualizar ou excluir objetos.
Ao utilizá-los, o comando Commit deve ser usado após o método ter sido executado corretamente. Este comando, como nos Business Components, também é utilizado após um Save ou Delete.

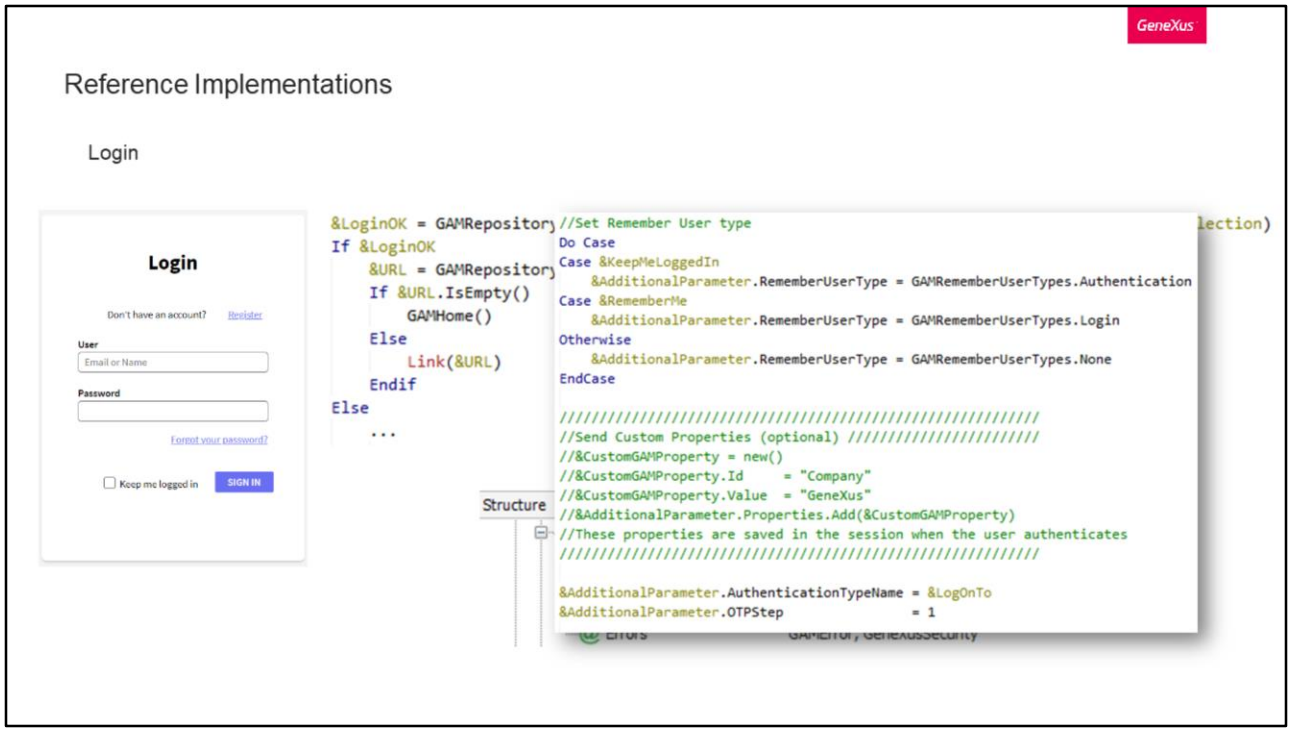
Os únicos métodos GAM que executam um commit implícito são os relacionados ao início e fim de sessão, e são executados em uma nova unidade lógica de trabalho, onde não é necessário realizá-lo já que internamente o GAM produz os diferentes inserts e faz o commit.



Dentro do que são as implementações de referência, podemos encontrar usos de Objetos Externos estáticos e objetos de instância.
A diferença entre eles é que os estáticos são métodos que se aplicam quando no objeto não é colocado o &.

Alguns exemplos de objetos estáticos podem ser os seguintes:
GAMRepository.Login ==> Com isto, podemos fazer Login no repositório atual
GAMUser.Get ==> Com este podemos obter a informação do usuário atualmente logado
GAMApplication.Get() ==> Com o qual podemos obter a informação da aplicação atual

Exemplos de objetos de instância podem ser:
&GAMApplication.GetPermissions ==> Onde tendo o & é indicado que o método será aplicado sobre a instância do objeto carregado, que neste caso é GAMApplication. Com isto, obtemos as permissões de uma aplicação
&GAMUser.AddRoleById, onde recebe um identificador e uma coleção de erros, que com isso adicionamos uma Role ao usuário carregado previamente (que foi instanciado)



Vejamos agora duas implementações de referência próprias, como Login e Logout. Vamos começar com o Login. Para isto, temos o objeto GAMExampleLogin.

No momento de pressionar o botão de iniciar sessão, a implementação é baseada em executar o método Login do External Objects: GAMRepository.
Como vemos, está sendo utilizado o método Login deste objeto já fornecido pelo GAM. Os parâmetros incluídos na chamada são:
Nome de usuário e senha, obtidos a partir do WebPanel.
Parâmetros adicionais: Estes podem ser a opção de manter a sessão iniciada, lembrar do usuário ou outro caso. Por sua vez, temos a possibilidade de criar propriedades Custom e associá-las à sessão do usuário como vemos em tela. Também podemos enviar-lhe o tipo de autenticação usado, informação relacionada com OTP, etc.
Coleção de erros: Se houver algum erro, são carregados na referida variável para depois revisar.

O resultado da operação é armazenado na variável booleana LoginOK e, dependendo desse valor, é a ação que se realiza.
Se este for verdadeiro, é armazenado na variável URL o valor resultante do método GetLastErrorsURL. Este retorna a URL onde ocorreu o último erro do GAM e, uma vez que é executado, é removido seu valor. Isto é útil para quando um usuário tenta acessar um objeto com autenticação e não consegue porque ainda não está logado, portanto é redirecionado primeiro para o login e depois a partir daqui é levado de volta

para o objeto em questão, agora sim com uma sessão válida.
Caso contrário, simplesmente é redirecionado para a Home definida na aplicação.

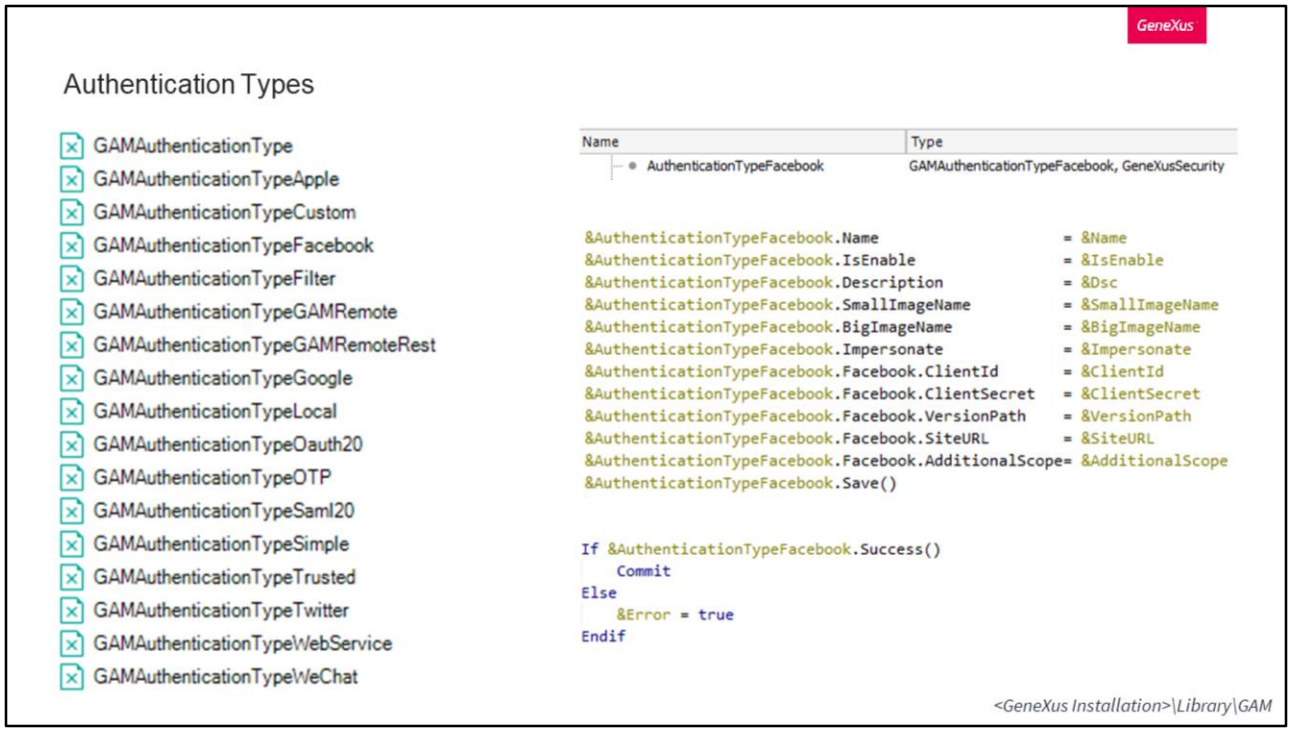
GeneXus

Reference Implementations

Logout

```
Event 'Logout'
    GAMRepository.Logout(&Errors)
    GAMExampleLogin()
EndEvent
```

Logout.
Uma implementação para isto é supersimples, pois poderia ser realizada da seguinte forma.
Com a primeira instrução termina a sessão atual e no caso de existirem erros, são salvos no SDT &Errors.
A segunda instrução simplesmente encerra o fluxo levando o usuário ao Web Panel GAMExampleLogin, ou na ausência deste, ao Web Panel responsável pelo login.
Esta última instrução só deve ser realizada nos casos em que esta aplicação NÃO seja cliente de um IDP. Se for, nada precisa ser chamado após o Logout para que seja realizado o encerramento de sessão no IDP.

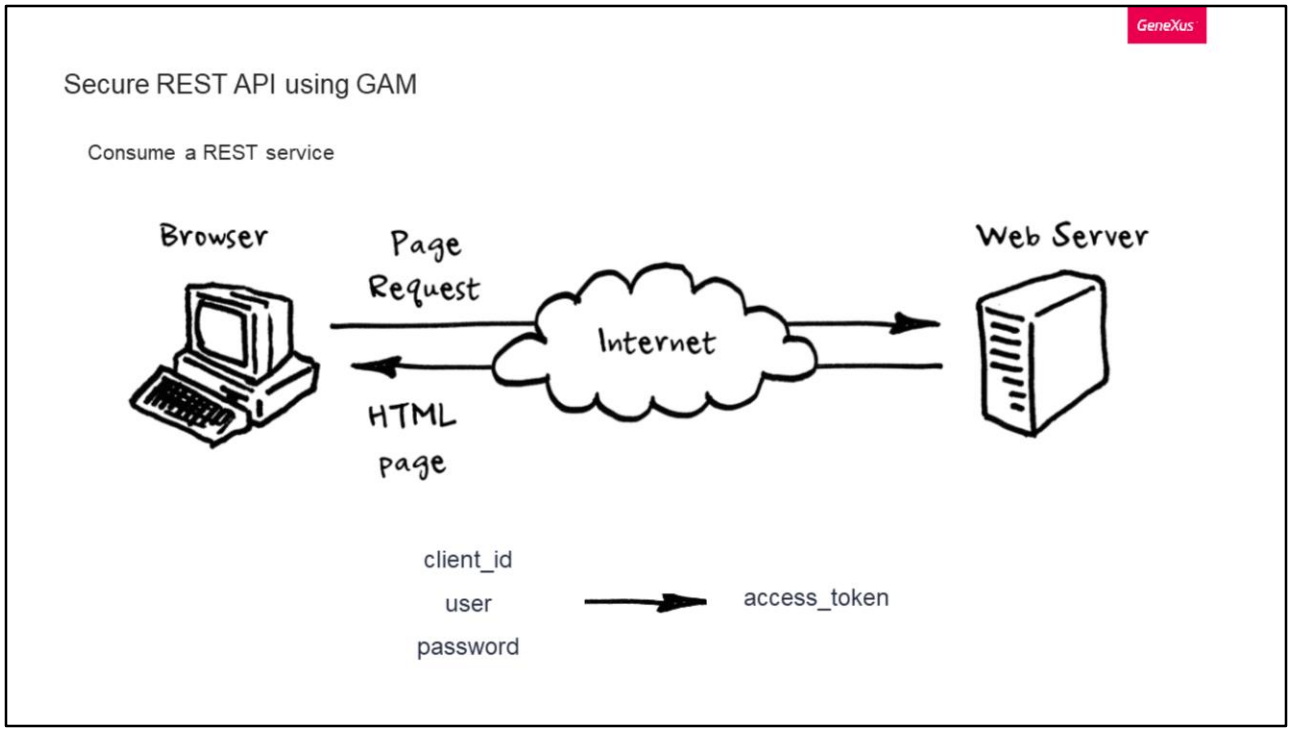


No curso introdutório também abordamos os tipos de autenticação a partir do backoffice web do GAM, mas mencionamos que isto por sua vez é possível por meio da API do GAM.
Para isso, temos os External Objects que vemos em tela, um para cada tipo de autenticação suportado pelo GAM.

Vamos pegar o exemplo do tipo de autenticação com Facebook.
Se quiséssemos criar um destes, a primeira coisa que precisamos fazer é criar uma variável deste tipo.
Posteriormente, começamos a preencher os diferentes atributos do tipo de autenticação, de acordo com nossos dados.
Se o tipo de autenticação for bem-sucedido, realizamos um commit para persistir os dados.

Isso é tudo.

Nos exemplos da API do GAM podem ser encontrados este e todos os outros exemplos para cada tipo de autenticação, tanto para a inserção, como para a modificação e exclusão.
Estes exemplos estão localizados na pasta de instalação do GeneXus, dentro do diretório Library/GAM.



Serviços REST.

É muito comum que as aplicações exponham serviços REST, portanto, a segurança é muito importante quando a privacidade dos dados é uma preocupação.
No GeneXus, como já dissemos, a solução para isto é utilizar GAM, onde por trás dele utiliza a tecnologia OAuth, facilitando para o usuário a complexidade que ele contém.

A API do GAM fornece uma maneira de restringir o acesso dos usuários aos serviços REST definidos na aplicação.

Para consumir um serviço REST no GeneXus, primeiro deve ter o client_Id da Aplicação, e o usuário e senha com permissões de acesso a esta aplicação.
Antes de consumir o serviço web, primeiro deve obter um access_token.

Vejamos uma possível implementação de um procedimento que o consome.

GeneXus

Secure REST API using GAM

Consume a REST service

```
&addstring = "client_id=xxx&grant_type=password&scope=FullControl&username=test&password=test"

&getstring = &urlbase + "/oauth/access_token"

&httpclient.AddHeader("Content-Type", "application/x-www-form-urlencoded")
&httpclient.AddString(&addstring)
&httpclient.Execute("POST", &getstring)

&httpstatus = &httpclient.StatusCode
//String response
&result = &httpclient.ToString()
//JSON response
&GAMOauth20AccessToken.FromJsonString(&result)
```

A primeira string é composta do seguinte:

- Client_id da aplicação
- Grant_type do fluxo
- Scope da conta de usuário que deseja acessar
- Nome de usuário da conta que deseja acessar e sua senha

A URL a consumir será a base da aplicação, seguida de /oauth/access_token.
É necessário adicionar o header Content-Type.
E executamos uma requisição POST com a referida informação, através do httpclient.

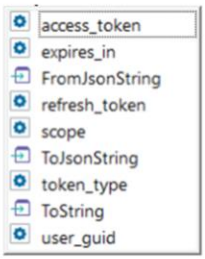
GeneXus

Secure REST API using GAM

Consume a REST service

Response


```
{
    "access_token": "85a3006c-0606-41d2-980e
        -223f88463ec2!c2ed363379e5de5dd33cd84627f452a074d332413
        65c6932e9abdc6a728e4d66be2f5b63ba6de8",
    "token_type": "Bearer",
    "expires_in": 180,
    "refresh_token": "001l4yQ6r1jb4r7eD9TUfMscq6R9fCXmYua97bh",
    "scope": "gam_user_data+gam_user_roles",
    "user_guid": "1a651f1a-d211-4c48-a5db-218d23986d1d"
}
```

Para ler a resposta, podemos fazê-lo tanto em uma string quanto em um JSON. No caso deste último, temos os seguintes atributos para poder ler da resposta programaticamente.

No caso em que a informação esteja correta, a resposta da requisição deverá parecer da seguinte forma, nos concedendo em um JSON os dados que vemos em tela.

Agora suponhamos que já temos o access_token que vimos anteriormente, como procedemos para consumir realmente o serviço que queremos?

Para fazer isto, cada chamada deve conter o access_token incluído no cabeçalho Authorization de nossas solicitações, depois da palavra OAuth como vemos em tela.

Um exemplo de código pode ser o seguinte, onde é mostrado como obter uma lista de produtos.
Indicamos a url, adicionamos os cabeçalhos onde um é o que dissemos antes com o access_token.
E finalmente executamos a solicitação, que neste caso é um GET a DPProduct, que, como dissemos, representa uma lista de produtos exposta como serviço web REST, onde este é um serviço seguro graças ao fato de GAM estar habilitado em sua KB.

Para fechar a parte teórica deste tema, veremos os External Objects mais comuns e utilizados nas aplicações GeneXus.
Vamos começar com GAMRepository.

Este objeto é bastante extenso, tanto em suas propriedades quanto em seus métodos. As opções mais comuns que fornece, é tudo o relacionado diretamente a um repositório, Login, usuários, sessões, roles, aplicações, etc.

Vejamos alguns dos métodos mais comuns.

Em primeiro lugar temos GetAuthenticationTypes. Este método serve para obter os tipos de autenticação de um repositório.
Então temos GetUsersOrderBy que obtém todos os usuários de acordo com as preferências que passamos a ele como filtros, onde também podemos definir uma ordem para a lista de retorno (que correspondem ao domínio GAMUserListOrder).
GetRoles e GetApplications correspondem a métodos para obter as roles e aplicações, respectivamente, de um repositório.
Finalmente temos GetSessionLogs com o qual podemos obter um log de todas as sessões de um repositório, tendo a possibilidade, por exemplo, de filtrar pelas sessões ativas.

Como vemos, em todos os métodos sempre são incluídos os mesmos parâmetros:

Aqueles que terminam em Filter são úteis para passar filtros para a consulta a ser realizada na base de dados. Cada um corresponde a outro External Object com sua especificação, os quais podem ser revisados antes de utilizá-los com finalidade de definir uma filtragem adequada às nossas necessidades. O outro parâmetro corresponde aos diferentes erros que pode causar o método ao ser utilizado, os quais depois podemos ler para verificar o resultado da operação.

GAMApplication.

Com este objeto teremos controle total sobre as aplicações definidas no GAM.

Vejamos quatro de seus métodos mais comuns.
Em primeiro lugar, temos GetPermissions, que nos permite obter a lista de permissões definidas em uma aplicação. Em seu primeiro parâmetro indicaremos a filtragem que queremos realizar na lista de resultado.
Os três restantes servem para adicionar, modificar ou excluir uma permissão de uma aplicação. Neste método devemos indicar através do primeiro parâmetro os dados da permissão à qual aplicaremos uma destas funções.

GAMUser.

Com este objeto temos o controle total de um usuário do GAM. Entre seus métodos, destaca-se poder obter toda a informação de cada um, e poder controlar suas roles, permissões e sessões,

Entre os mais comuns temos o clássico GetRoles para obter as roles de um usuário.
Então temos GetUnassignedRoles que seria o método inverso do anterior. Este nos retorna as roles que o usuário não tem atribuídas.
GetPermissions, que retorna as permissões associadas ao usuário, e como no método anterior, temos a variável de filtragem.
E finalmente PhysicalDelete, que exclui fisicamente o usuário e todos os seus dados relacionados.

GAMRole.

Com este objeto podemos gerenciar as roles do GAM. Entre seus métodos, destaca-se a capacidade de obtê-las, adicionar e excluir, controlar suas permissões, entre outros.

Os métodos mais comuns são os seguintes:
Com GetPermissions obteremos todas as permissões associadas à role que temos na variável GAMRole.
Com GetChildren, obteremos todas as roles filhas que contém a role identificada.
Em ambos os métodos podemos utilizar a filtragem que comentamos em todos os outros métodos.

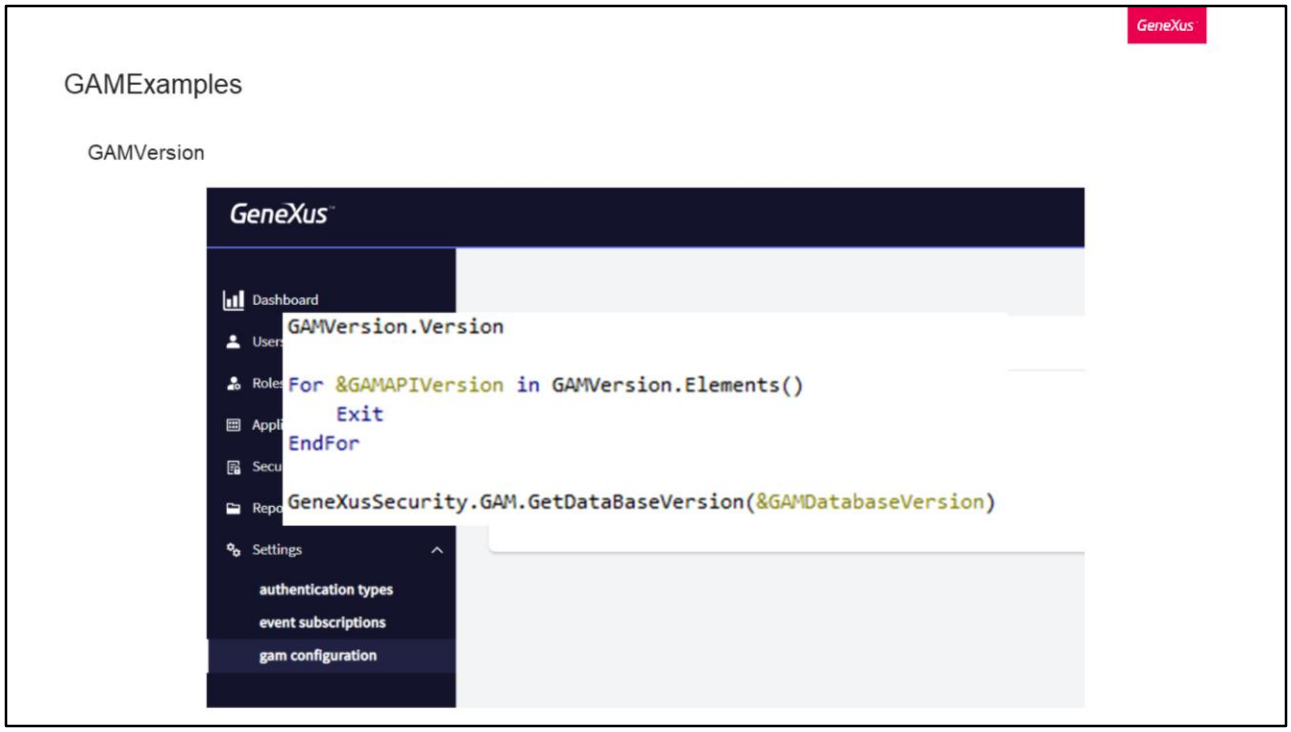
Em seguida temos adicionar e remover role por identificador, cuja função é basicamente o que diz o nome do método com a exceção de que trabalham com as roles filhas da role para a qual é instanciado. No parâmetro Id passamos o identificador da role a adicionar ou excluir.

Vamos continuar com GAMSession.

Com este objeto temos o controle de todas as sessões. Podemos obter todos os atributos de uma sessão, suas roles, entre outras coisas.

Vejamos dois de seus métodos:
Se quisermos carregar os dados da sessão atual, fazemos da seguinte maneira utilizando GAMSession.Get() e armazenando o resultado na variável &GAMSession. Esta variável é exclusivamente para a sessão atual.
Se, por exemplo, quiséssemos ver o histórico de todas as sessões, devemos utilizar o objeto &GAMSessionLog.

Da mesma forma que carregamos os dados da sessão atual, podemos carregar as roles que uma sessão possui utilizando o método GetRoles.



Por último temos GAMVersion, que embora seja um domínio e não um objeto, pode ser útil conhecê-lo.

Para obter a versão de GAM, temos duas maneiras de fazer isso:
A primeira é através do Backoffice web, como vemos em tela, na opção GAM Configuration dentro de Settings.
A segunda forma é através de código GeneXus, e para isto temos diferentes opções:

- Se quisermos saber a versão da API, podemos fazer o seguinte:
  1. Primeiro podemos consultá-la com o domínio, da seguinte forma.
  2. Outra opção é realizando o seguinte For, onde na variável GAMAPIVersion do tipo GAMVersion teremos a versão buscada.

- Se quisermos saber a versão em que está a estrutura da base de dados, podemos realizar uma consulta à base de dados como a que vemos em tela, isto retorna um booleano indicando o resultado e na variável GAMDatabaseVersion fica registrado o número de versão.

Um detalhe disso que devem levar em consideração é que a versão da base de dados pode ser mais nova que a versão da API do GAM.

GeneXus™
by Globant

training.genexus.com
wiki.genexus.com