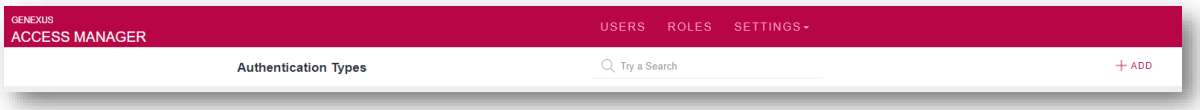
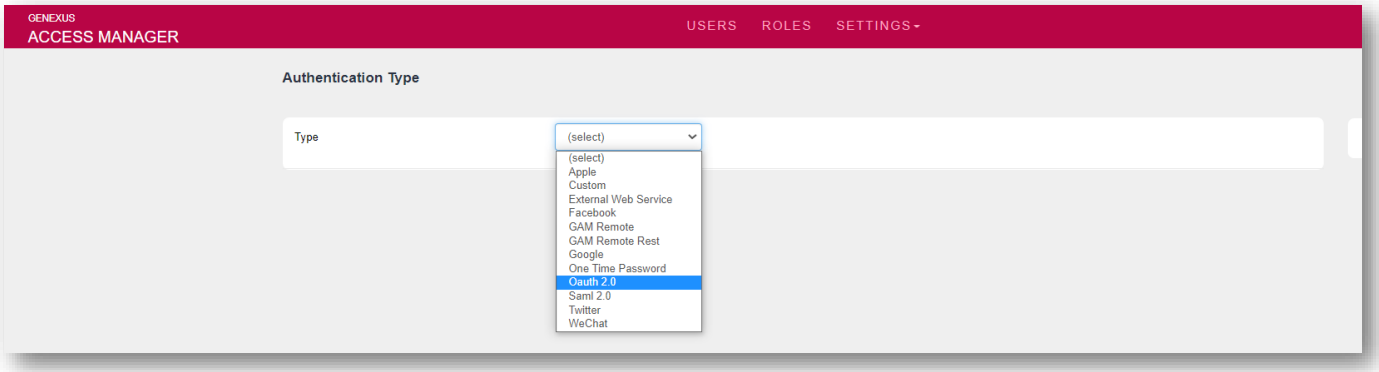
DEMO: OpenID Connect

Primeira demo: OpenID Connect

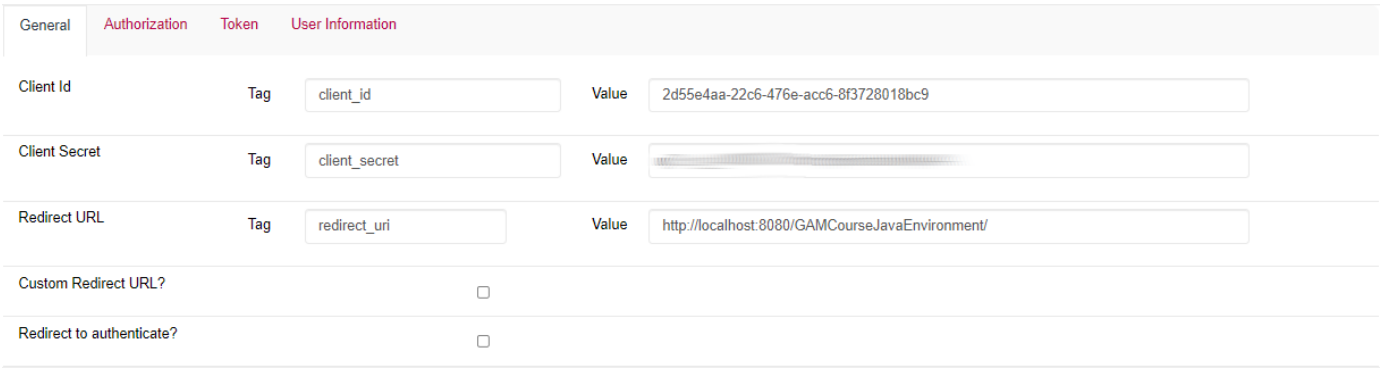




Para esta demo, utilizaremos o protocolo Oauth 2.0 no GAM. Nosso provedor de identidade será Azure Active Directory por meio da Microsoft.

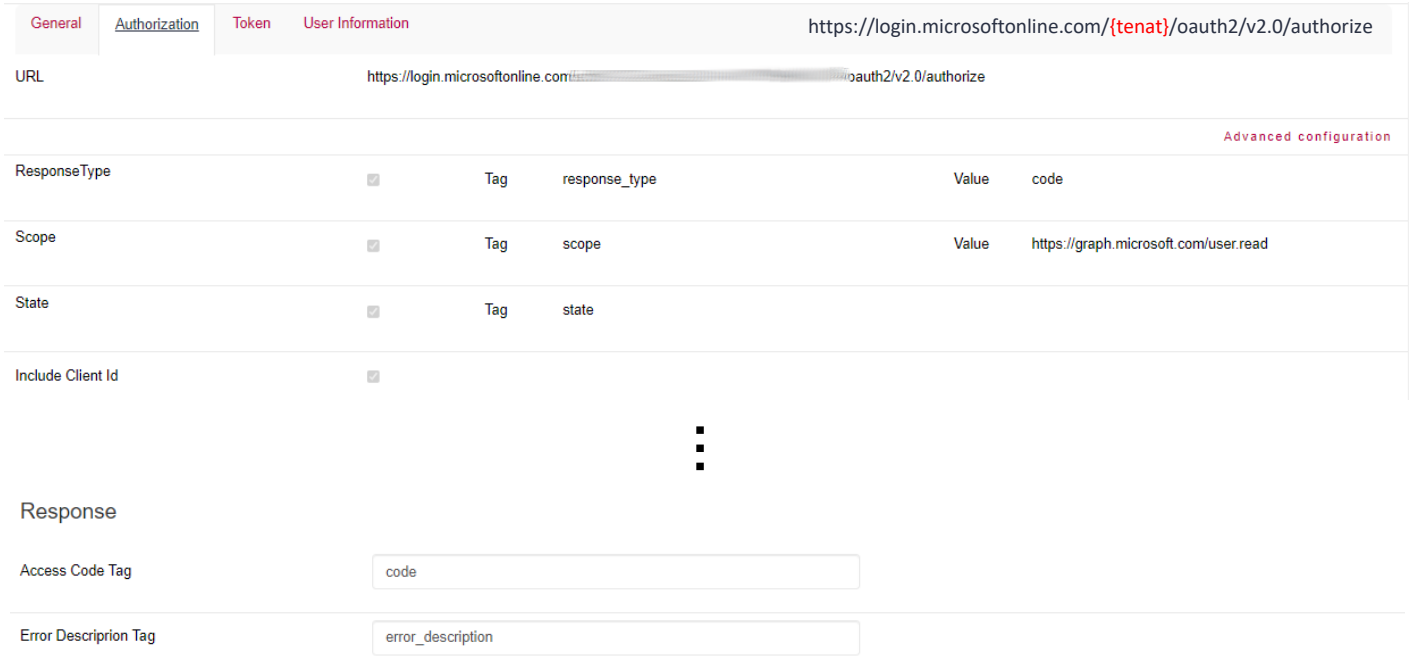
Assumiremos que a configuração do lado do Azure já foi realizada corretamente e não entraremos em detalhes sobre isso. Para ver como realizá-la, você pode encontrar na Wiki de GeneXus um artigo detalhado sobre isto.

Em primeiro lugar, devemos criar um novo Tipo de autenticação GAM Oauth 2.0 e definir os conceitos básicos, como o Nome, Descrição, etc.



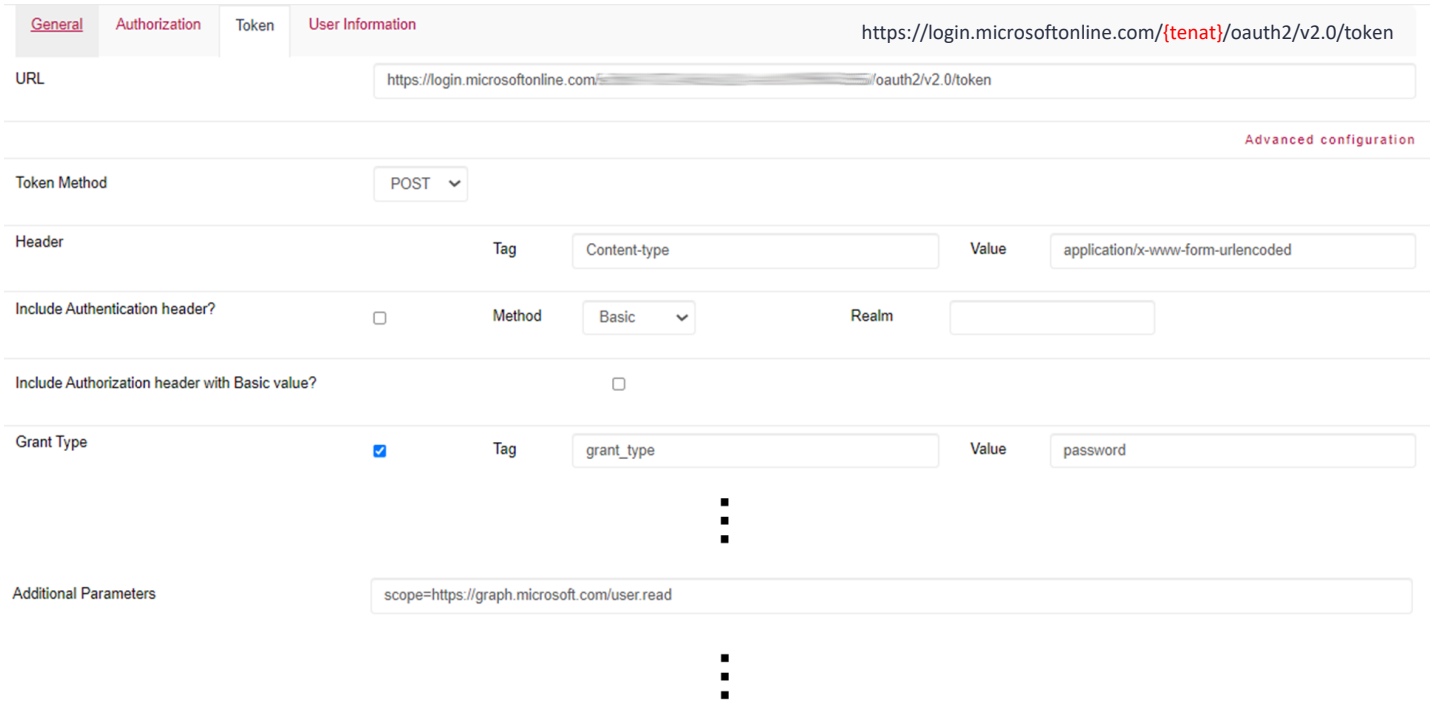
Na aba General, deve ser definido o seguinte:
Primeiro, definimos o Client ID e Client Secret que obtemos do Azure.
A URL de redirecionamento deve ser a URL Base do Backend de nossa aplicação.

Como comentamos no vídeo anterior, não marcaremos a opção Redirecionar para autenticar, pois queremos fazer login a partir do próprio GAM.



Agora vamos para a aba Authorization.
Aqui devemos definir a URL do Azure obtida a partir de seu portal, a qual se apresenta da seguinte forma.
A segunda coisa a modificar deve ser o Scope, que deve conter a URL que vemos em tela.

O restante fica tudo por padrão.

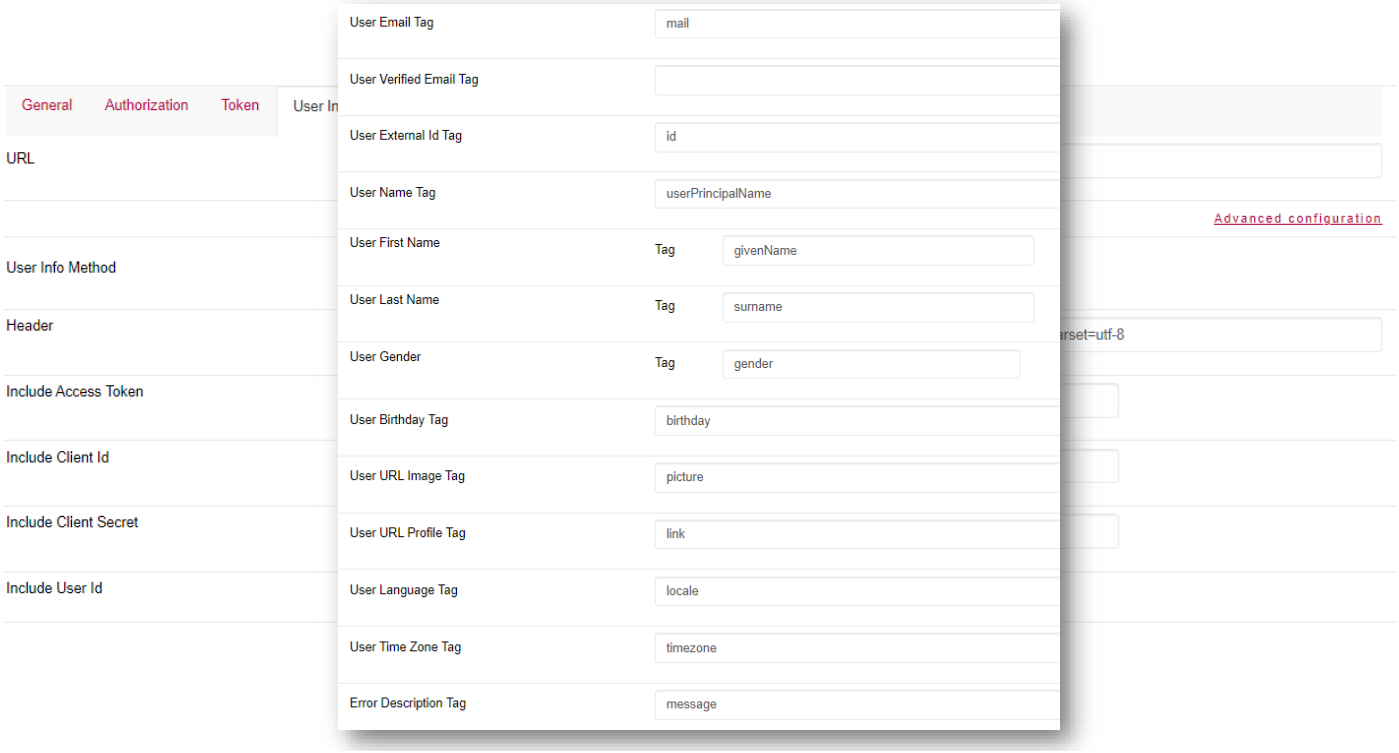


Na aba Token, novamente definimos a URL do Azure obtida a partir de seu portal, a qual se apresenta da seguinte forma.

O restante fica por padrão, exceto os campos Grant Type e Additional Parameters, os quais definimos com o que vemos em tela.

Vale esclarecer que este último só deve ser alterado quando queremos que não seja redirecionado no momento de fazer login e seja realizado a partir do login do GAM.
Caso contrário, o Grant Type deve ficar com o valor padrão (que é *authorization_code*) e sem parâmetros adicionais.

As demais opções ficam configuradas por padrão.

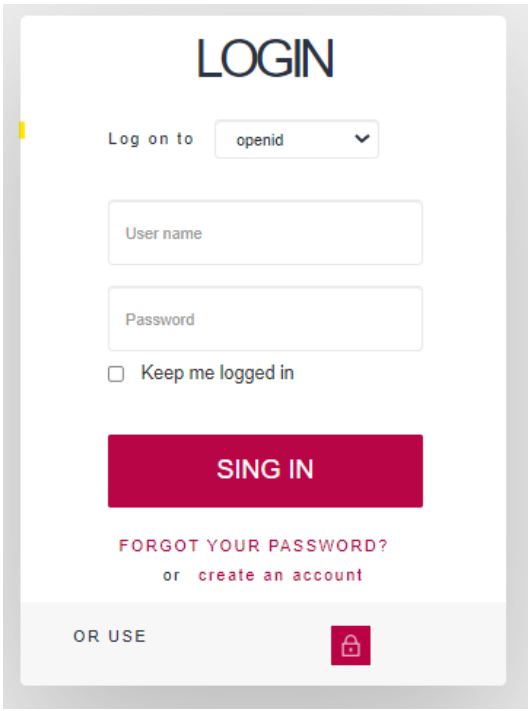


Finalmente, na aba User Information, aqui simplesmente definimos a URL que vemos em tela (também obtida a partir do Azure) e não incluímos nada.

Desta forma é como é criado o usuário no GAM local, e é de onde é mapeada a obtenção da informação do usuário de acordo com o IDP configurado.
O IDP deve retornar um identificador único, que deve ser configurado em "User External Id Tag", e por este, é que nos sucessivos logins de GAM se tem certeza de que está sendo autenticado o mesmo usuário.

O restante dos parâmetros, definimos como vemos em tela.

Com isto concluímos a configuração.

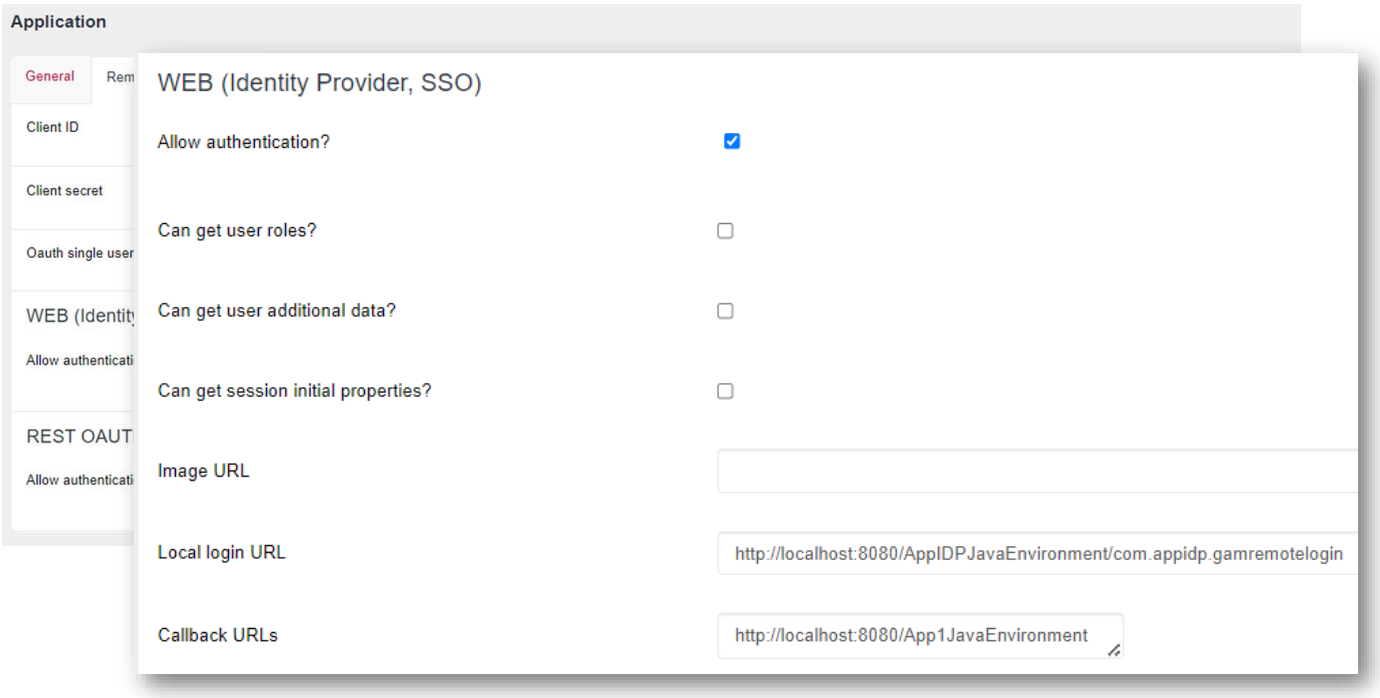


Agora simplesmente basta ir em Login, selecionar fazer login com o tipo de autenticação recém-criado e inserir as credenciais de um usuário definido no Azure.

Isso é tudo.

DEMO: IDP

Segunda demo: IDP.

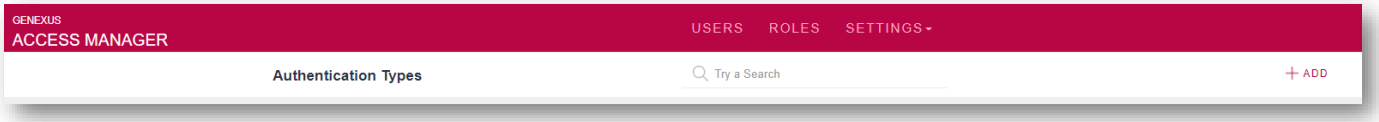


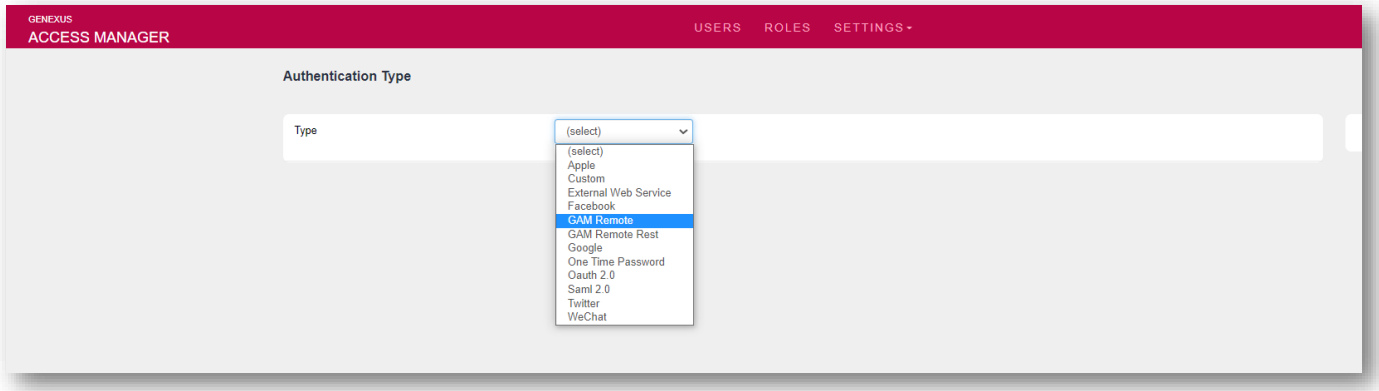
Para esta demo, voltaremos a utilizar o protocolo Oauth 2.0. O GAM através dele, será nosso provedor de identidade.
Em primeiro lugar, devemos configurar nossa aplicação GAM definida no servidor do IDP que irá interagir como o provedor.

Para isto, devemos acessar a aba “Remote Authentication” nas configurações da aplicação a partir do Backend do GAM.
Aqui salvamos o Client ID e Client Secret para defini-los posteriormente na aplicação cliente.
Em seguida, devemos marcar a opção de permitir a autenticação na seção WEB (Identity Provider, SSO). Lá, pode ser indicado se deseja compartilhar com o Cliente as roles dos usuários, a informação adicional, etc.
O importante a ser mostrado na demo, são as URL de login local e callback. A primeira deve corresponder ao Web Panel GAMRemoteLogin da KB, que será a que realize o processo de login no IDP. A segunda deve ser o path da aplicação cliente de onde será invocado o IDP, que será chamada após o término do processo de Login. Este último parâmetro pode ser composto por mais de uma URL, que devem ser separadas por ponto e vírgula.

Obviamente é neste GAM onde deve ter definidos os usuários que serão utilizados para efetuar login no IDP.

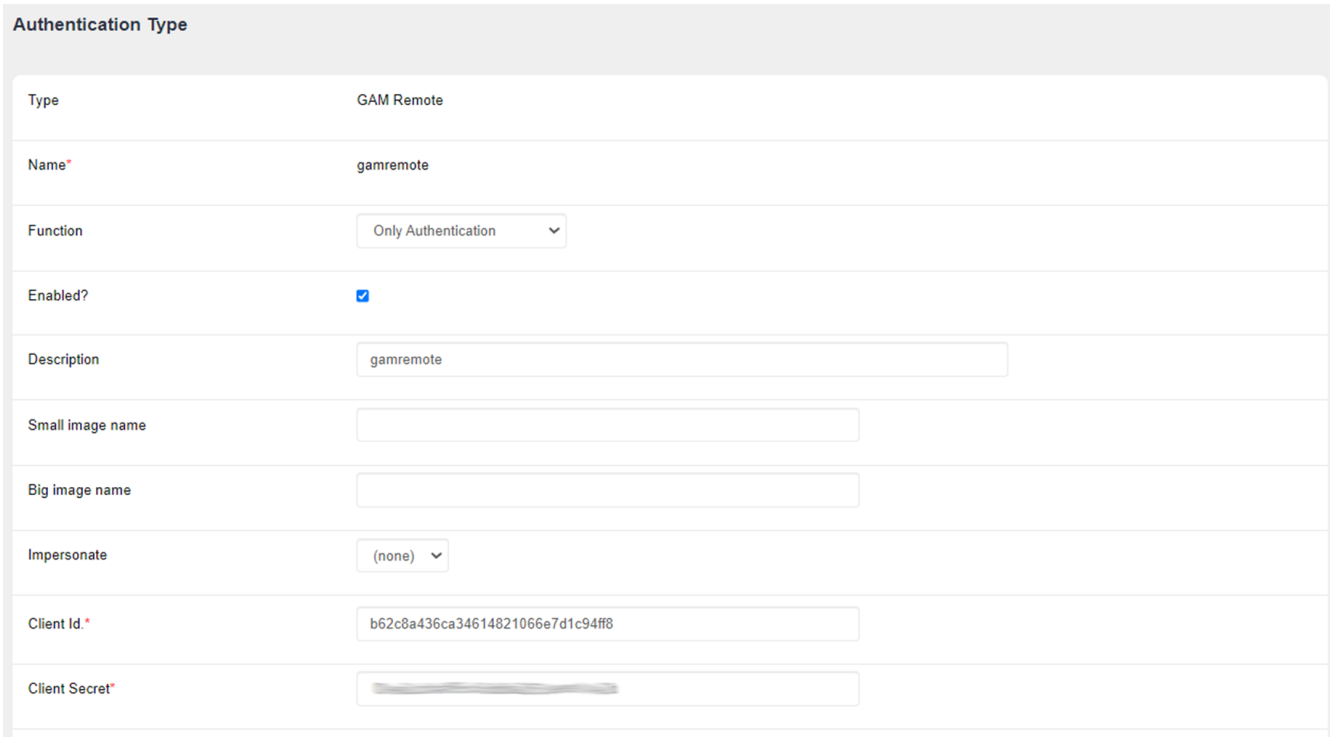
Tendo esta configuração e um usuário criado, já finalizamos o processo do lado do IDP.





Vejamos o lado do Cliente.

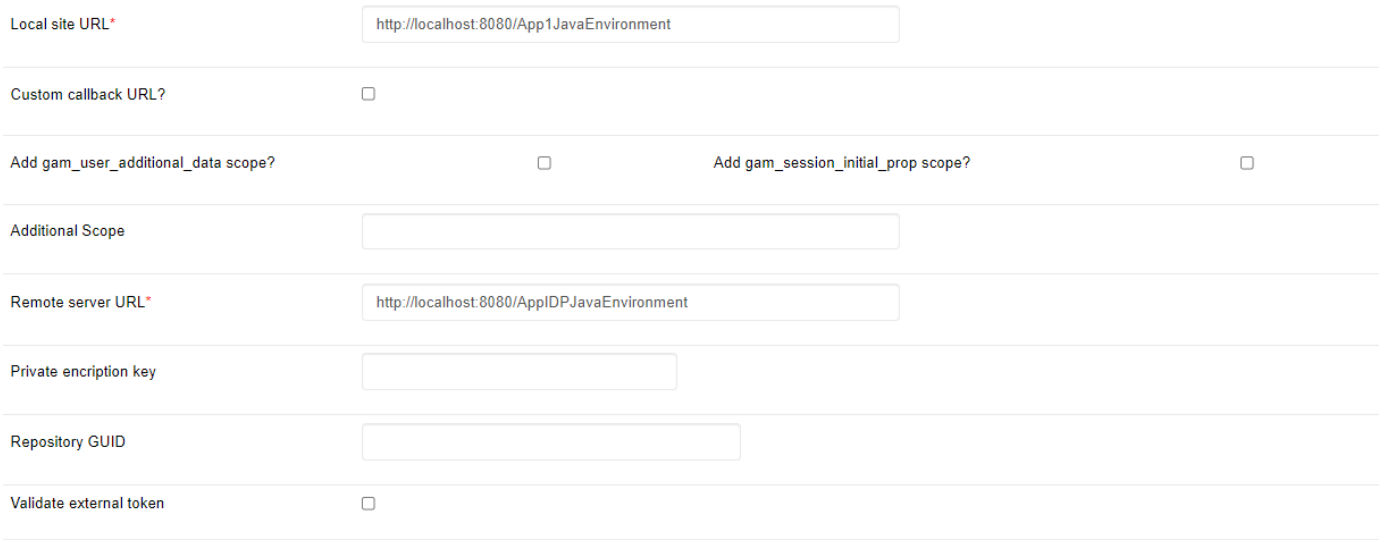
O primeiro passo é ir para Tipos de Autenticação a partir do menu do backend de GAM e criar um do tipo GAM Remote.



O importante a configurar aqui é o seguinte:

A propriedade Function, definiremos como Only Authentication, pois no lado do servidor de IDP não indicamos que sejam compartilhadas as roles de usuário. Caso coloquemos a outra opção, Autenticação e roles, no momento de logarmos retornará um erro.

O seguinte a configurar são o Client Id e Client Secret que salvamos do IDP.



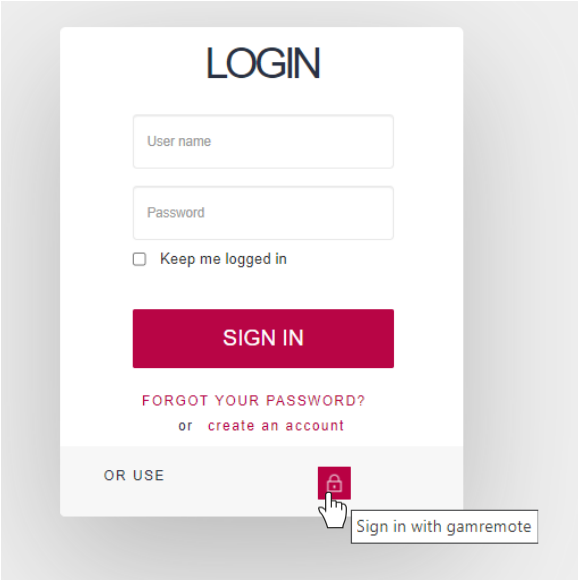
$Server/<Base_URL>

Posteriormente configuraremos a propriedade “Local site URL” com o endereço de nossa aplicação cliente, a mesma que já especificamos na Callback URL no servidor; também a propriedade “Remote server URL” com o endereço do IDP, seguindo o formato que vemos em vermelho.

Como comentários adicionais:
A propriedade “Add gam_user_additional_data scope?” deve ser ativada quando queremos passar dados adicionais do usuário. No lado do servidor, deve ser selecionada a propriedade Permitir Autenticação, na seção Web (Provedor de Identidade, SSO).

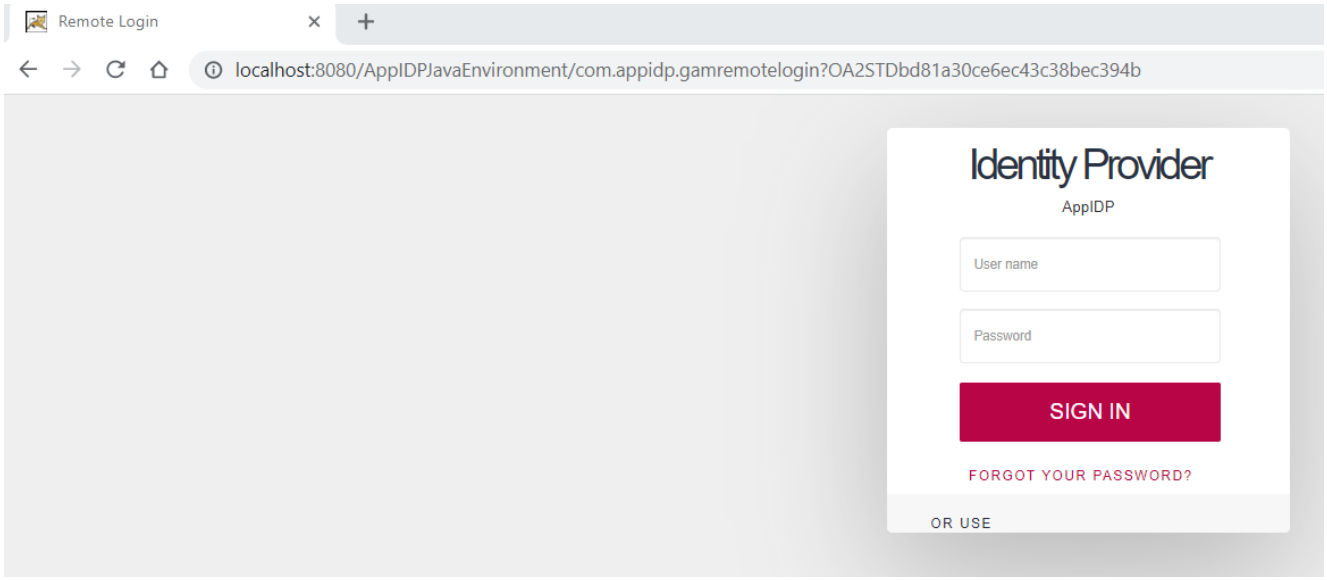
A propriedade “Add gam_session_initial_prop scope?” consiste em pedir ao IDP que devolva ao cliente as propriedades iniciais estabelecidas dinamicamente no início de sessão. Obviamente, no IDP também deve ser configurado que seja enviada esta informação.

Finalmente, a propriedade “Validate External Token” valida o vencimento da sessão de acordo com a expiração do token e o renova automaticamente sem que nós tenhamos que realizá-lo manualmente.

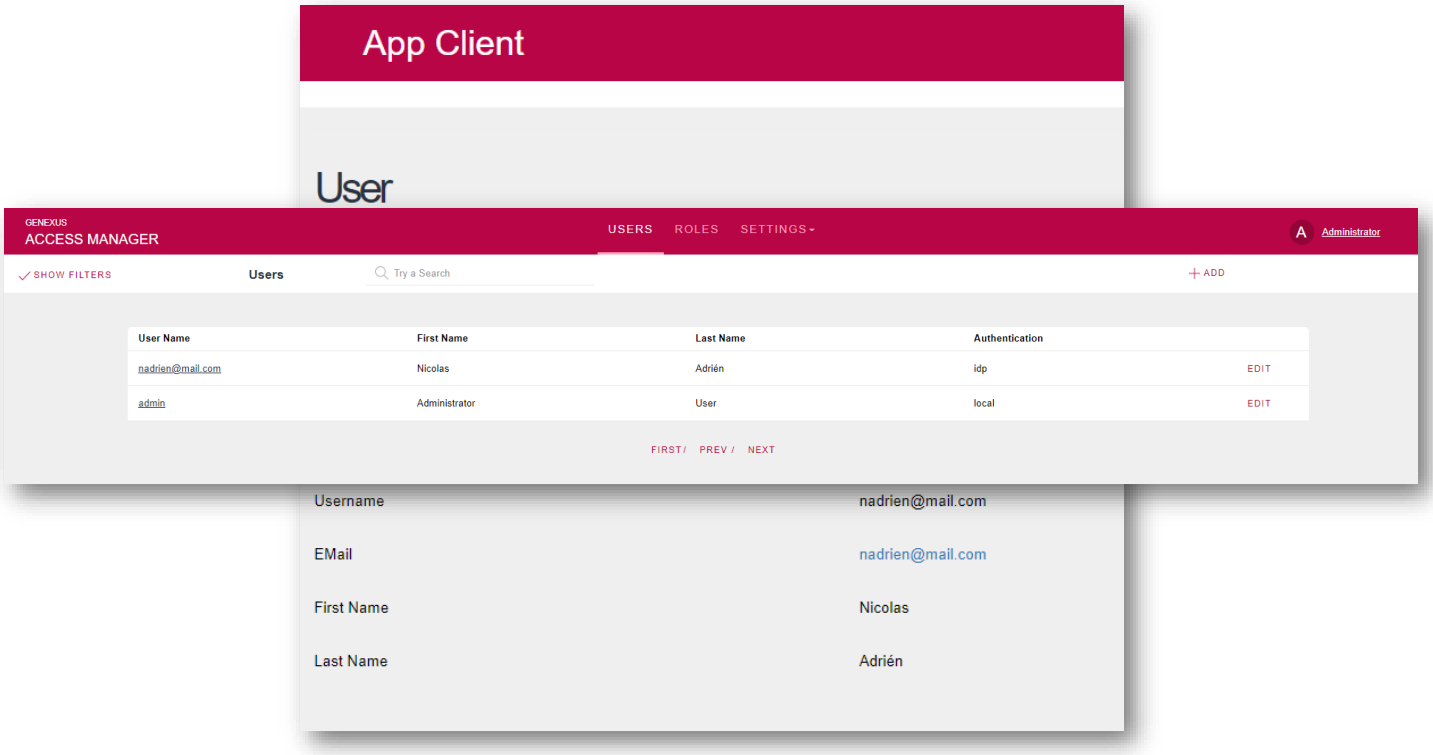


Para fins da demo, criamos um Web Panel na aplicação Cliente, onde nos mostra os dados do usuário logado. Obviamente, este objeto tem ativada a segurança integrada com valor Autenticação.

Quando queremos acessá-lo, dado que não estamos logados, nos redireciona para o login.
Vejamos que agora que definimos o tipo de autenticação Oauth, a partir do login temos a opção de acessar através dele.



Ao clicar nessa opção, vemos que nos redireciona para o IDP e seu login remoto. Procedemos para efetuar login com o usuário que havíamos definido no IDP.



De fato, vemos como agora nos redirecionou para nosso Web Panel com a informação do usuário logado.

Se formos ao Backend da aplicação cliente, podemos ver como foi criado o usuário que criamos no IDP com sua informação.

DEMO: Custom Authentication

Terceira demo: Autenticação Custom.

```
Parm(in:&StrInput, out:&StrOutput); //&StrInput and &StrOutput are varchar(256)


&Key = '03E1E1AAA5BCA19FBA8C42058B4ABF28'
&GAMWSLoginIn.FromJson(&StrInput) // &GAMWSLoginIn is &GAMWSLoginInSDT data type

//Decrypt parameters
&UserLogin = Decrypt64( &GAMWSLoginIn.GAMUsrLogin, &Key )
&UserPassword = Decrypt64( &GAMWSLoginIn.GAMUsrPwd, &Key )
&GAMWSLoginOut = New GAMWSLoginOutSDT() //&GAMWSLoginOut is &GAMWSLoginOutSDT data type
&GAMWSLoginOut.WSVersion = GAMAutExtWebServiceVersions.GAM10
&GAMWSLoginOut.User = New GAMWSLoginOutUserSDT()

Do 'ValidUser'

&StrOutput = &GAMWSLoginOut.ToJson()

Sub 'ValidUser'
    If &UserLogin = !"user"
       If &UserPassword = !"password"
          &GAMWSLoginOut.WSStatus = 1
          &GAMWSLoginOut.User.Code = !"code"
          &GAMWSLoginOut.User.FirstName = !"FirstName"
          &GAMWSLoginOut.User.LastName = !"LastName"
          &GAMWSLoginOut.User.EMail = !"name2@domain.com"
          Do 'GetRoles' //optional
       Else
          &GAMWSLoginOut.WSStatus = 3
       EndIf
    Else
       &GAMWSLoginOut.WSStatus = 2
    EndIf
EndSub

Sub 'GetRoles'
    &GAMWSLoginOutUserRol = New()
    &GAMWSLoginOutUserRol.RoleCode = "role_1"
    &GAMWSLoginOut.User.Roles.Add(&GAMWSLoginOutUserRol)
    &GAMWSLoginOutUserRol = New()
    &GAMWSLoginOutUserRol.RoleCode = "role_2"
    &GAMWSLoginOut.User.Roles.Add(&GAMWSLoginOutUserRol)
EndSub
```

Para realizar uma autenticação Custom, devemos criar um procedimento.
Na Wiki de GeneXus pode encontrar o exemplo que vemos em tela, com uma lógica muito simples já definida. Fica a cargo do desenvolvedor modificá-lo sob suas condições.

Em primeiro lugar, vemos que como regras são definidos dois Varchar: um de entrada e outro de saída, os quais trarão os dados inseridos pelo usuário e retornarão o resultado do login com determinada informação do usuário (este último em caso de sucesso, claro).

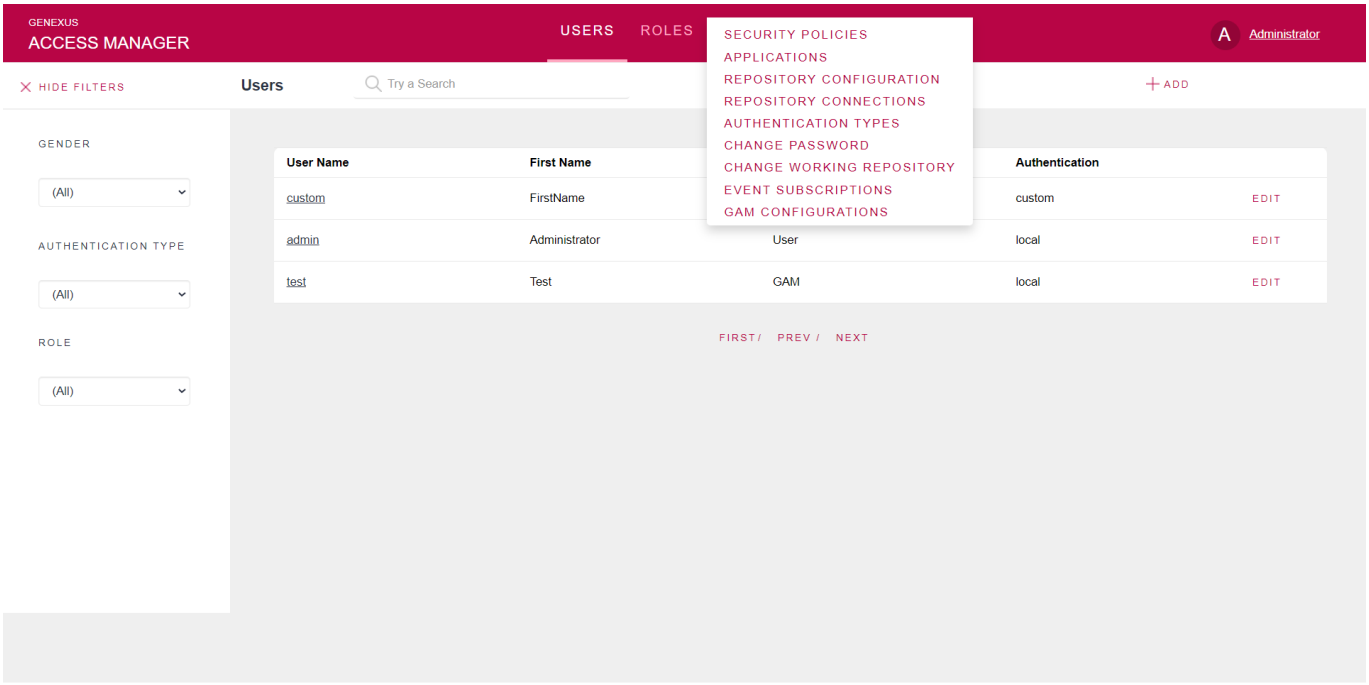
Em seguida, é definida uma chave que mais adiante detalharemos, e são descriptografados os parâmetros desse parâmetro de entrada procedente das regras, além de ser criado um data type que será carregado no parâmetro de saída ao finalizar.

Posteriormente, no método **ValidUser**, é realizada a validação do nome de usuário e senha, o que está feito a título de exemplo verificando se o nome de usuário é “user” e a senha é “password”. Caso contrário, são retornados erros diferentes dependendo da falha.
Este método deve ser alterado para uma lógica de login mais segura e que não faça diferenciação entre os erros com base em usuário ou senha.

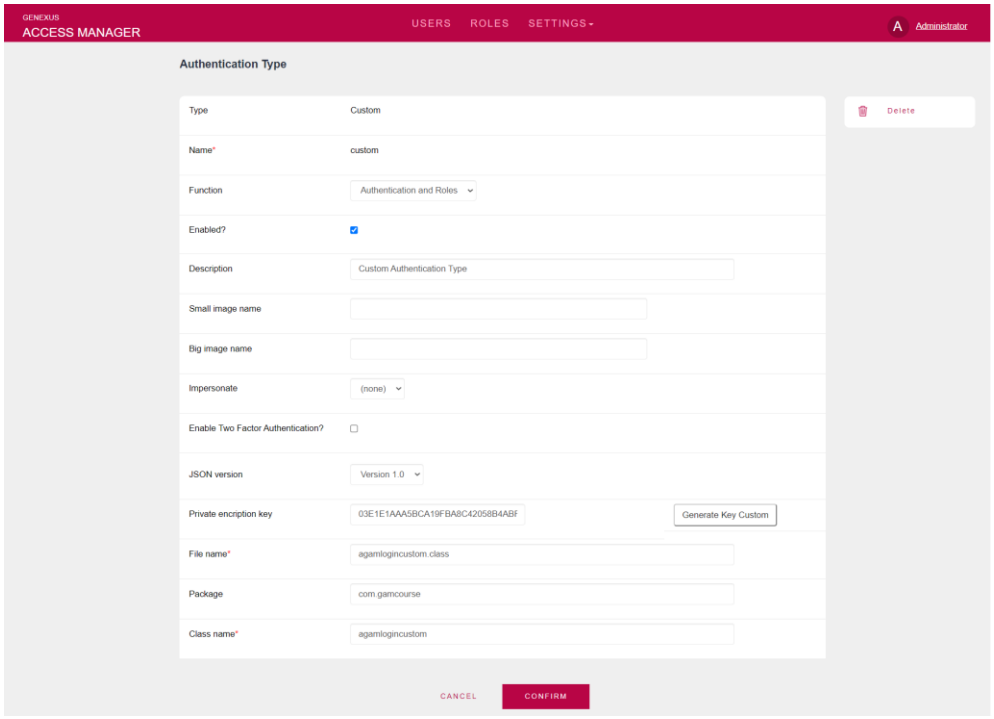
Opcionalmente, pode ser utilizado o método **GetRoles** para definir determinadas roles para o usuário logado.

Este método é de utilidade quando nós queremos programar como validamos a

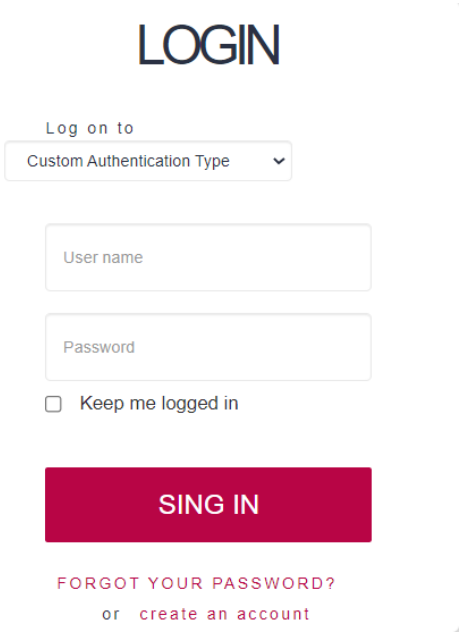
senha de um usuário, seja para validá-la em uma base de dados local, em um LDAP ou em outro local onde se encontram armazenadas as credenciais dos usuários.



Agora que já temos um procedimento personalizado para a autenticação, devemos configurá-lo no GAM.
O primeiro passo é ir para Authentication Types e criar um novo do tipo Custom.



Aqui, as configurações a serem destacadas são as seguintes:
Function: Como já dissemos, permite especificar se queremos que o tipo de autenticação seja Autenticação e roles, ou apenas Autenticação. No nosso caso deixamos a primeira opção.
Private encription key, aqui deve ser configurada a chave de criptografia utilizada no procedimento para descriptografar o usuário e senha recebidos. Se você se lembra, no slide do procedimento GeneXus que mostrei anteriormente, foi definida uma chave, que é a que inserimos nesta propriedade. Isto é útil porque a função de criptografia do GeneXus a utiliza para criptografar o nome de usuário e senha quando são passados ao programa.
Nome de arquivo: aqui é especificado o nome do arquivo que corresponde ao procedimento externo. No caso de Java é opcional.
Pacote: aqui é especificado no caso de modelos Java o mesmo valor de propriedade de nome de pacote de Java, e no caso de modelos NET o valor da propriedade de espaço de nomes da aplicação. Esta propriedade é opcional e depende se o procedimento ou programa utilizado tem pacote ou não.
Finalmente Nome da classe, uma propriedade obrigatória que especifica o nome da classe do procedimento.

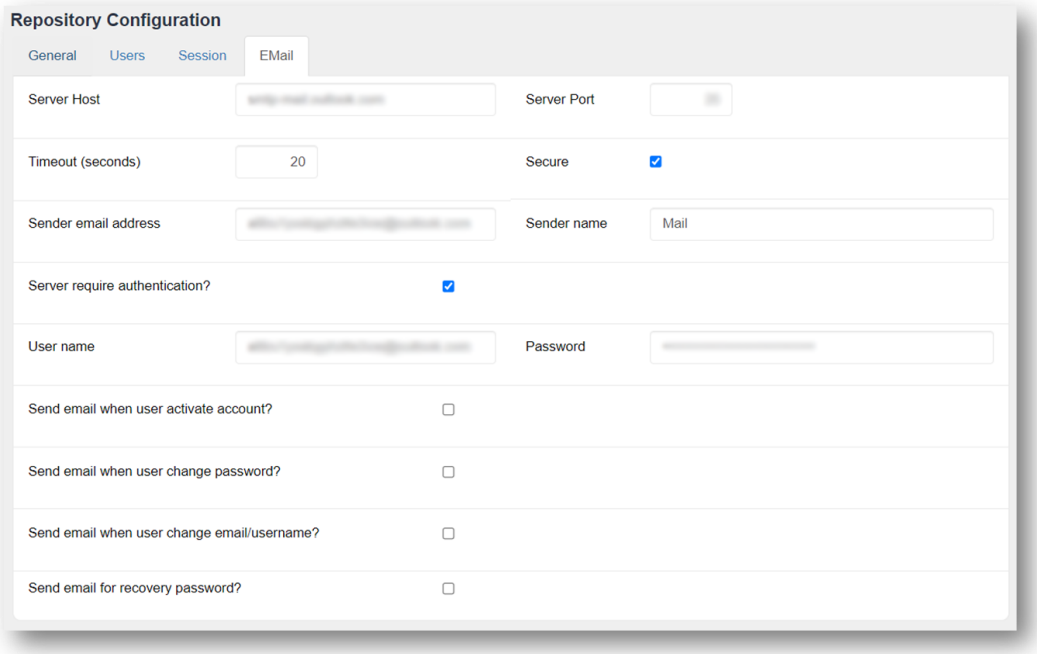


Depois de ter tudo configurado, simplesmente em nosso login é definido o tipo de autenticação custom, e é realizado o login.

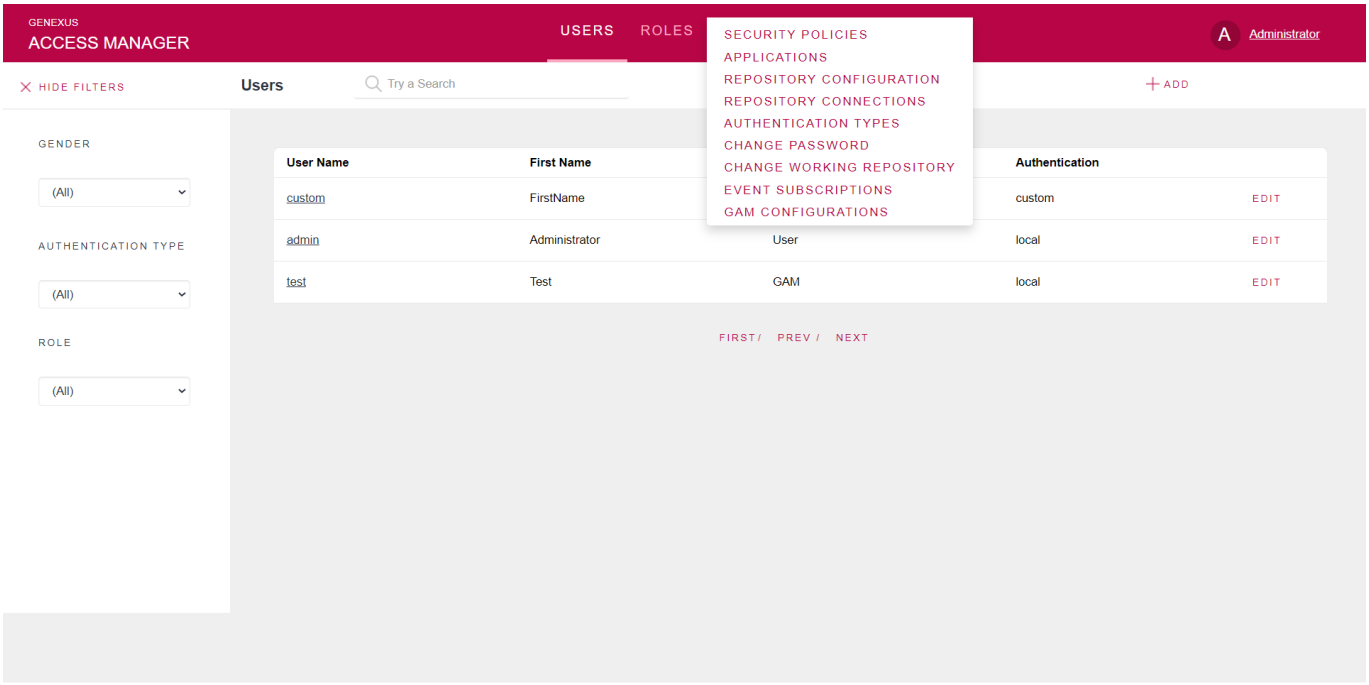
Um detalhe a mencionar é que neste caso o tipo de autenticação é selecionado através do combo destacado porque indicamos que não se redirecione para o IDP. Caso contrário, o tipo de autenticação é mostrado como um ícone na parte inferior do login, como vimos na Demo do IDP.

DEMO: OTP

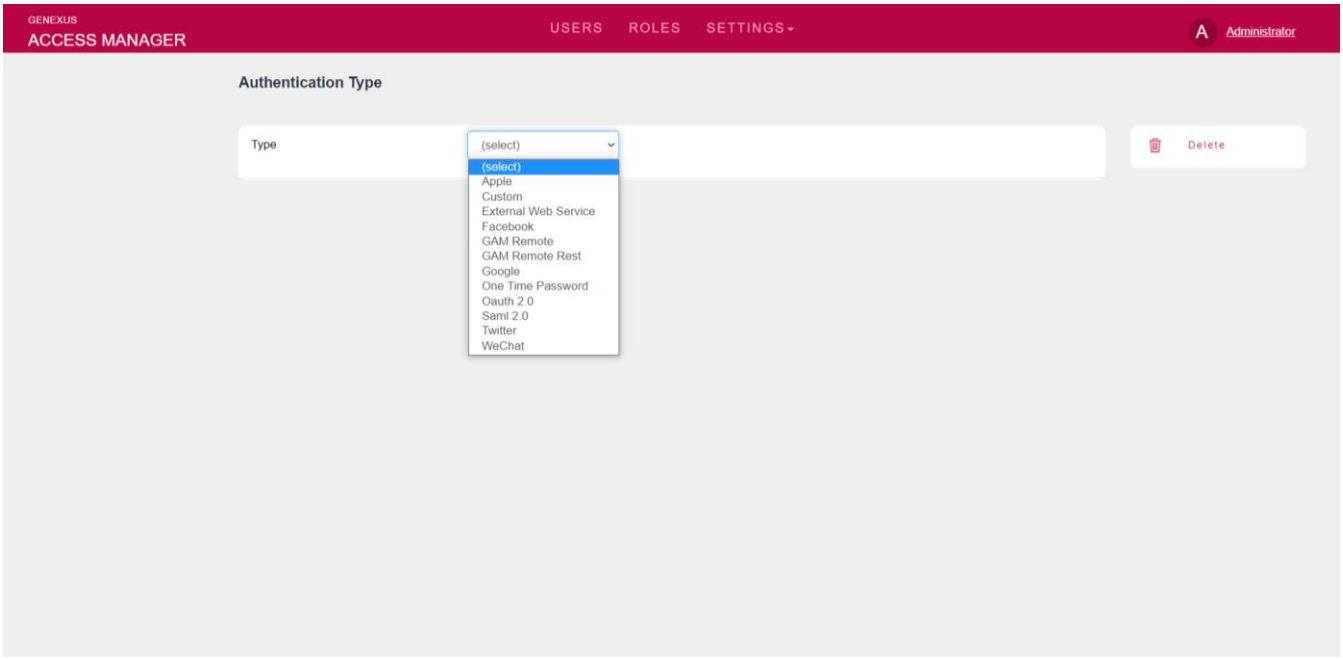
OTP.



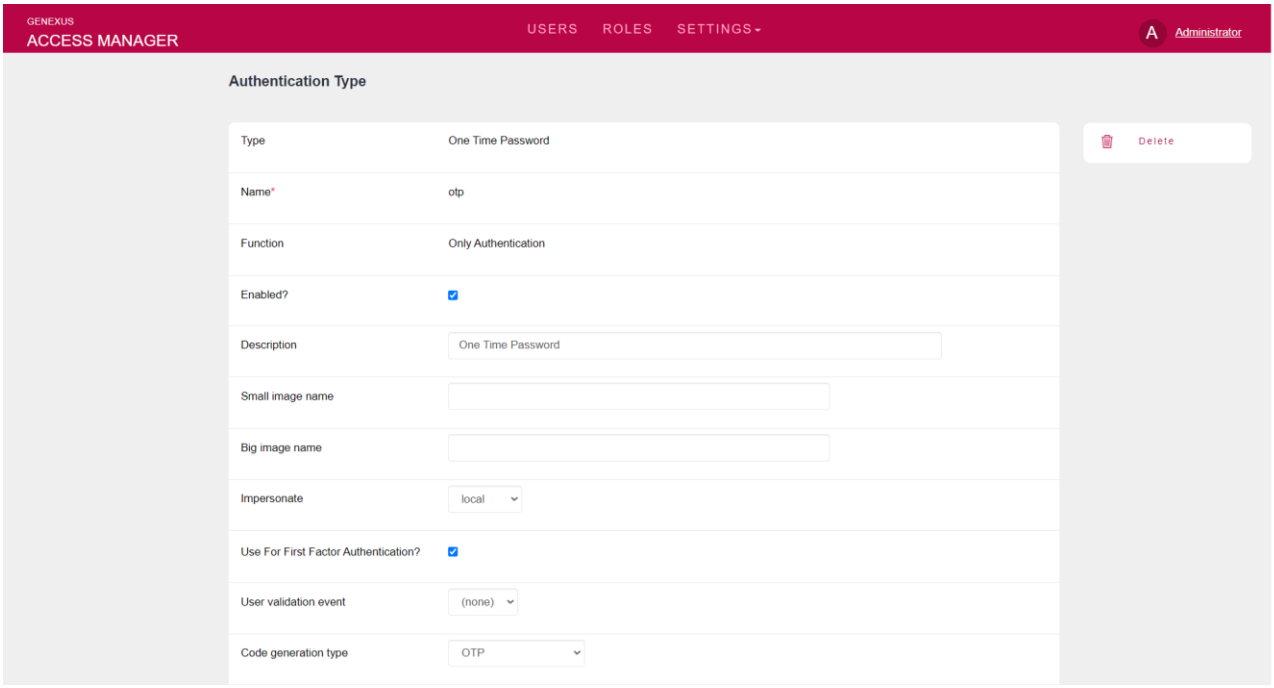
Um pré-requisito para fazer funcionar OTP é que o repositório deve ter configurado o serviço de e-mail para enviar os códigos.
Isto é configurado na opção “Configuração do Repositório” do backend de GAM.



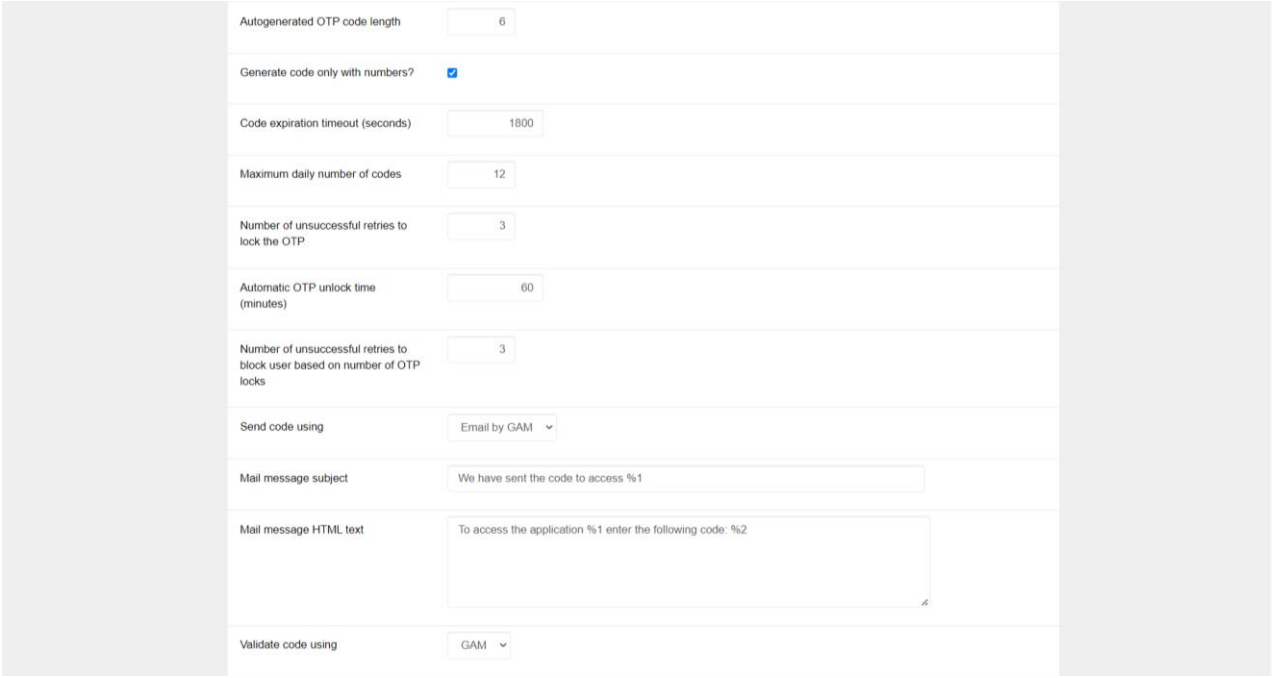
Agora sim, para definir este tipo de autenticação, tudo é realizado e configurado novamente através do backend de GAM.
Como na Demo anterior, vamos para Authentication Types e registramos um novo tipo.



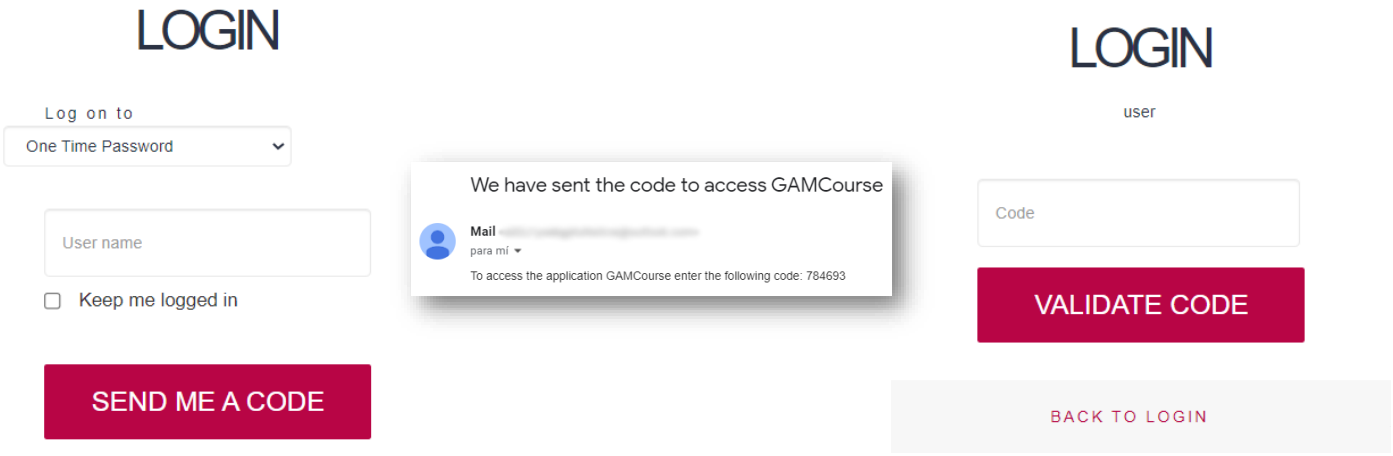
Neste caso, selecionamos o tipo One Time Password.



Passemos a descrever as propriedades mais importantes:
Impersonate: Aqui é especificado o tipo de autenticação em que os usuários serão validados ao usar o OTP. Como mencionei anteriormente no teórico sobre isto, os usuários já devem existir. Este é o único tipo de autenticação em que é necessário configurar esta propriedade, pois os usuários devem existir na base de dados de GAM.
Usar como autenticação de primeiro fator: Se não configurar esta propriedade, OTP só poderá ser usado como um segundo fator. No nosso caso, o habilitamos.



As demais propriedades são utilizadas para definir propriedades do código a ser enviado e o formato do e-mail.
No nosso caso usaremos o padrão de GAM que é o e-mail, mas lembre-se que existe a possibilidade do envio do código através de um SMS. Caso o desenvolvedor decida optar por esta segunda forma, deve implementar e configurar o evento de GAM, que então deve selecionar na propriedade "Send code using“.

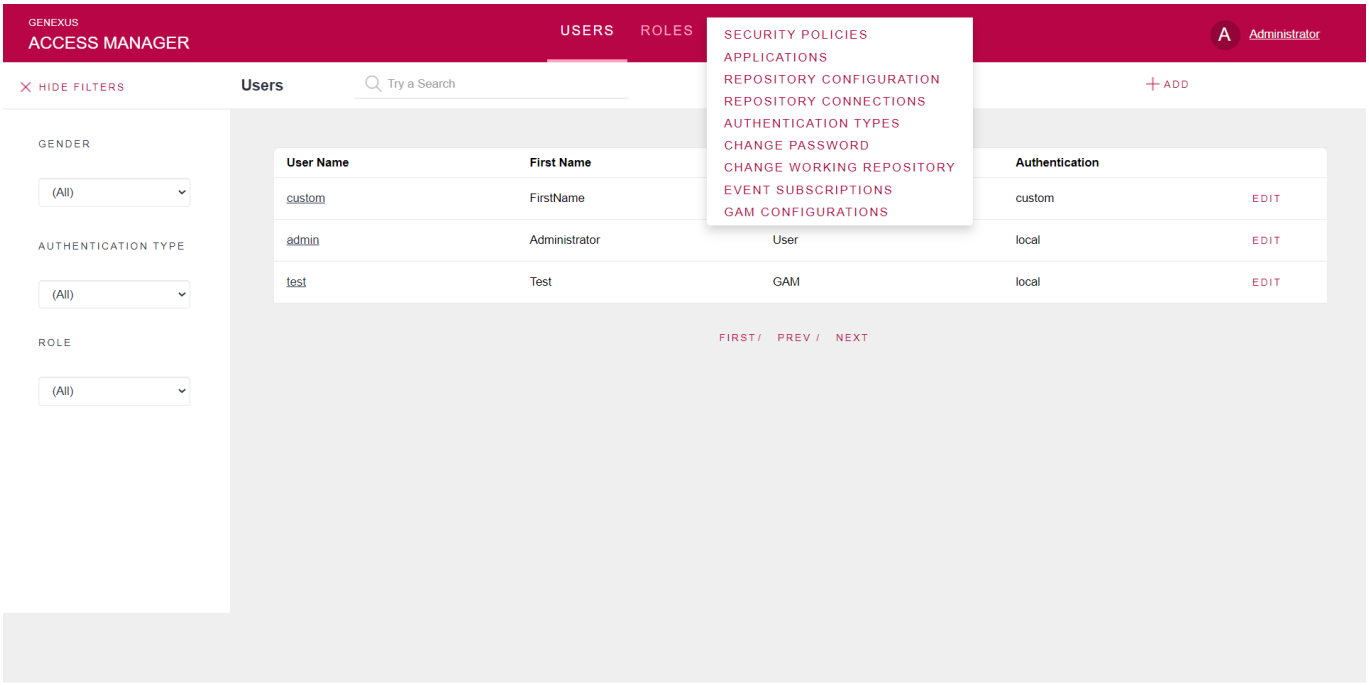


Finalmente, indo ao Login, selecionamos o tipo OTP, onde vemos que solicitará apenas o nome de usuário para o envio de código.
Após ter chegado o código através do e-mail, basta você digitar no login para se autenticar no sistema.

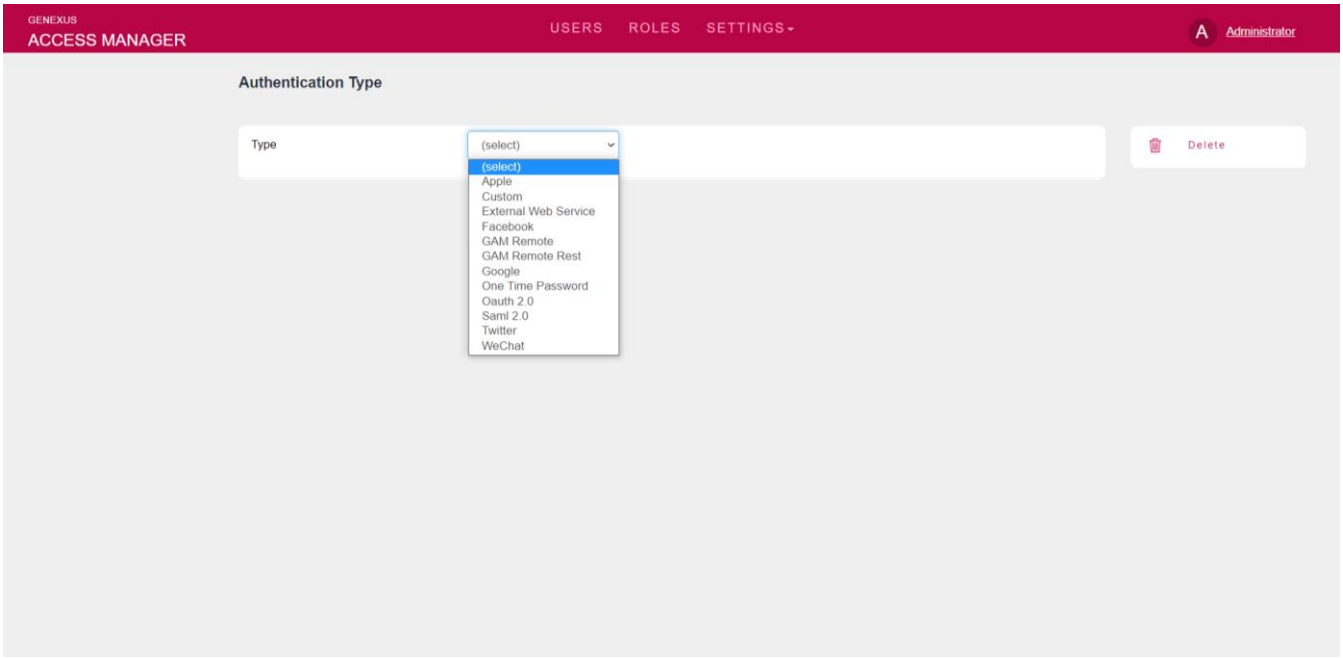
DEMO: TOTP

Última Demo: TOTP

Nesta demo, as etapas para configurar um novo tipo de autenticação TOTP são praticamente as mesmas do OTP, exceto por uma ressalva.

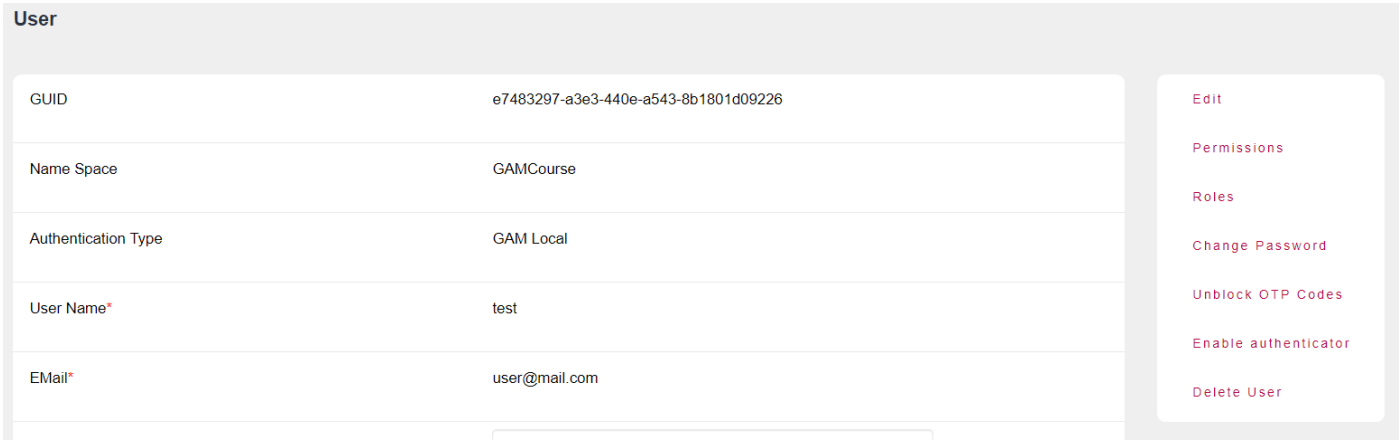


Para definir este tipo de autenticação, novamente vamos para Authentication Types e registramos um novo tipo.



Neste caso, também selecionamos o tipo One Time Password.

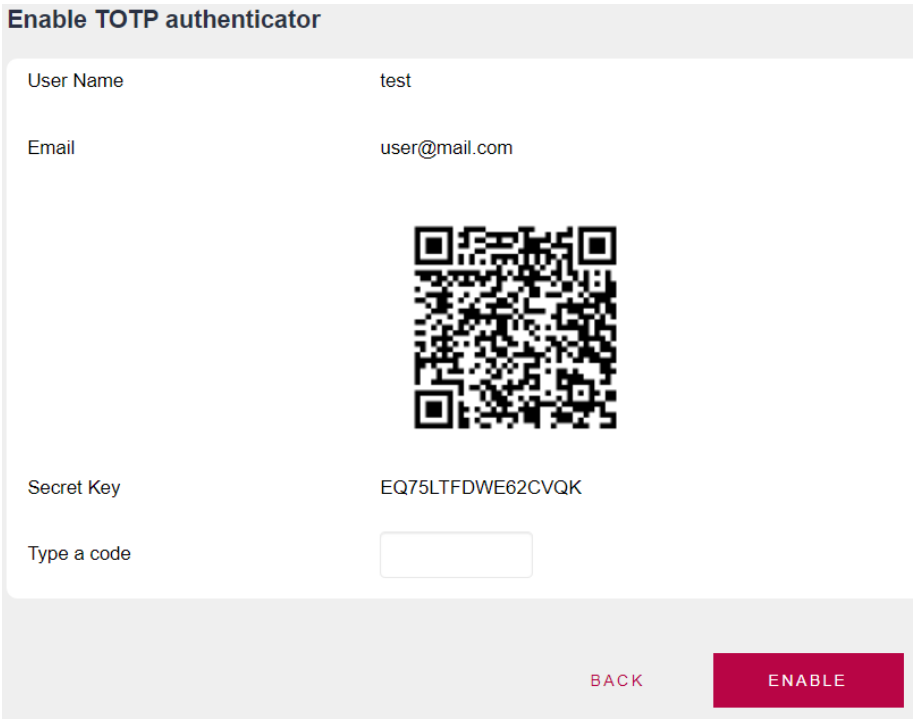
A diferença com OTP é a propriedade mostrada em tela, onde neste caso escolhemos TOTP Authenticator.
As demais propriedades são para configurações do código que não vêm ao caso.



Vejamos a ressalva mais importante que tem com OTP.

Cada usuário deve ativar a autenticação através de suas configurações. A diferença mais importante é que este algoritmo do código está baseado no tempo e os códigos são gerados pelas diferentes aplicações autenticadoras.

Para fins da demo, foi criado utilizando o usuário administrador do backend de GAM para um usuário “test” de uma aplicação de exemplo.
Os passos a serem realizados para esta maneira baseiam-se em ir até o usuário em questão e clicar em Enable authenticator.



Uma vez lá, será fornecido o código QR para ser configurado em um software ou aplicação móvel baseada em autenticação com senha de uso único, que após ter sido lido, retornará o código para inserir no campo “Type a code”.

Finalmente, o login é o mesmo de todos os tipos anteriores, onde neste caso existe uma aplicação intermediária que nos fornece o código de acesso.

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