Using the GAM API

Let's remember that APIs are properties and methods provided by GAM for applications to interact with it, allowing for communication with the GAM database, which is the one containing users, roles, and so on.

We will consider two new specific requirements for this application:

1. The event's attendees must be able to register at any time, by creating a username that allows them, among other things, to mark their favorite sessions and access these sessions' details from any device or computer with an Internet connection (through the SD or Web application).

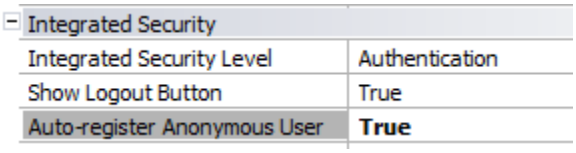
   If sessions are marked as favorites before the user registers, they will be associated with the device used for selecting them. But if the user decides to register at that same moment, the favorite sessions stored in the device will be associated with the new user, with no need for him/her to do anything else.

2. Only authorized users can view and run CRUD operations from the device itself. For example, for Speakers. In sum, it must be possible to associate special roles with special users.

In addition, we want the application to be secure: that is to say, business components exposed as REST services (and the other services) can only be run through the application.

To achieve this, we will enable the authentication feature at the main object level (and in this way it will be inherited by all the objects referenced by it). In addition, to make it possible for the user to access it without logging in, we will enable self-registration as anonymous user.

To do so, we open the EventDay dashboard and set the **Auto-register Anonymous User** property to **True**.



This will create a user in the GAM database, and its identifier will be the device identifier. It will be transparent for the user, who will think he isn't logged in when in fact he is logged in as anonymous user. Therefore, all favorites marked in this way will be associated with the user's device. To achieve this functionality, as well as to allow non-anonymous users to mark favorites, the FavoriteSessions transaction must be updated.

Before applying the GAM, since we didn't have information about users, favorite sessions had to be associated with the users' devices.

When we open the FavoriteSessions transaction we see that its primary key is a compound key made up by the DeviceId and SessionId attributes. That is to say, the list of favorites will depend on each device, and it will be different for each device.

Name	Type	Description
FavoriteSessions	FavoriteSessions	Favorite Sessions
DeviceId	VarChar(128)	Device Id
SessionId	Id	Session Id
SessionName	Name	Session Name
SessionInitialTime	DateTime	Session Initial Time
SessionSpeakers	Character(200)	Session Speakers
RoomImage	Image	Room Image

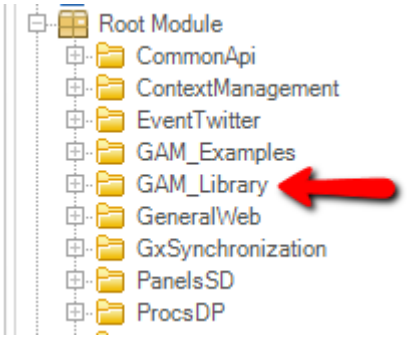
But now we don't want favorites to depend on the device used, but on the user. In this way, if a user has marked certain sessions as favorites in his device, he should be able to view them from the Event's web application.

To do so, first we change the structure of the FavoriteSessions transaction by replacing the DeviceId attribute with a UserId attribute, of GAMGUID type, in order to be able to use the GAM user identifier.

Name	Type	Description
FavoriteSessions	FavoriteSessions	Favorite Sessions
UserId	GAMGUID	User Id
SessionId	Id	Session Id
SessionName	Name	Session Name
SessionInitialTime	DateTime	Session Initial Time
SessionSpeakers	Character(200)	Session Speakers
RoomImage	Image	Room Image

When a session is marked as favorite, the user's ID must be used as part of the key that identifies it. To achieve this, we will use the GAM API to retrieve the logged-in user identifier, and then we will use this value to store favorite sessions.

Remember that APIs are located in the GAMLibrary folder.



The GAMUser external object will allow us to use the methods available and obtain, among other things, the associated user's roles.

Next, we open the Patterns tab and select Work With for Smart Devices. Also, we open the List node and select Grid1. In its conditions we enter: UserId = GAMUser.GetId(); so that only the favorite sessions corresponding to the logged-in user are displayed, and save.

Now we open WorkWithDevicesSession and select Section(General). In the events, we look for the “SessionFavorite” event. To know if a session is a favorite or to set it as a favorite, the

necessary interaction with the database is performed by the methods IsFavoriteSession and SetFavoriteSession, respectively.

```
Event 'SessionFavorite'
    Composite

        &IsFavoriteSession = IsFavoriteSession(SessionId)
        If &IsFavoriteSession
            ButtonFavorite.Class = 'Button.FavoriteDisabled'
        else
            ButtonFavorite.Class = 'Button.FavoriteEnabled'
        endif

        SetFavoriteSession( SessionId )

        Refresh
    endcomposite

Endevent
```

First, we open the SetFavoriteSession session.

```
&ClientInformationID = ClientInformation.Id
&FavoriteSessions.Load(&ClientInformationID, &SessionID)

if &FavoriteSessions.Fail()
    &FavoriteSessions = new()
    &FavoriteSessions.DeviceId = &ClientInformationID
    &FavoriteSessions.SessionId = &SessionID
    &FavoriteSessions.Save()
else
    &FavoriteSessions.Delete()
endif

commit
```

In the first line, the ClientInformation API is used to retrieve the Smart Device ID.

We will replace this line with a call to the GAM API to retrieve the logged-in user ID. We type: `&UserId = GAMUser.GetId()`. Next, we right-click on the &UserId variable and select Add Variable.

We replace ClientInformationId with UserId… and save.

```
&UserId = GAMUser.GetId()
&FavoriteSessions.Load(&UserId, &SessionID)

if &FavoriteSessions.Fail()
    &FavoriteSessions = new()
    &FavoriteSessions.UserId = &UserId
    &FavoriteSessions.SessionId = &SessionID
    &FavoriteSessions.Save()
else
    &FavoriteSessions.Delete()
endif

commit
```

We open the IsFavoriteSession method.

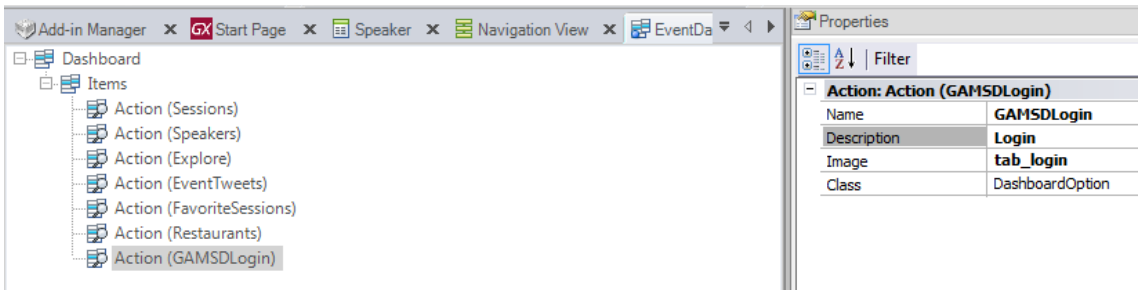
```
&IsFavorite = false
&ClientInformationID = ClientInformation.Id
for each FavoriteSessions
    where DeviceId = &ClientInformationID.Trim()
    &IsFavorite = True
endfor
```

And edit the line where ClientInformationId is loaded and the Where line, entering UserId instead of ClientInformationId. We save...

```
&IsFavorite = false
&UserId = GAMUser.GetId()
for each FavoriteSessions
    where UserId = &UserId
    &IsFavorite = True
endfor
```

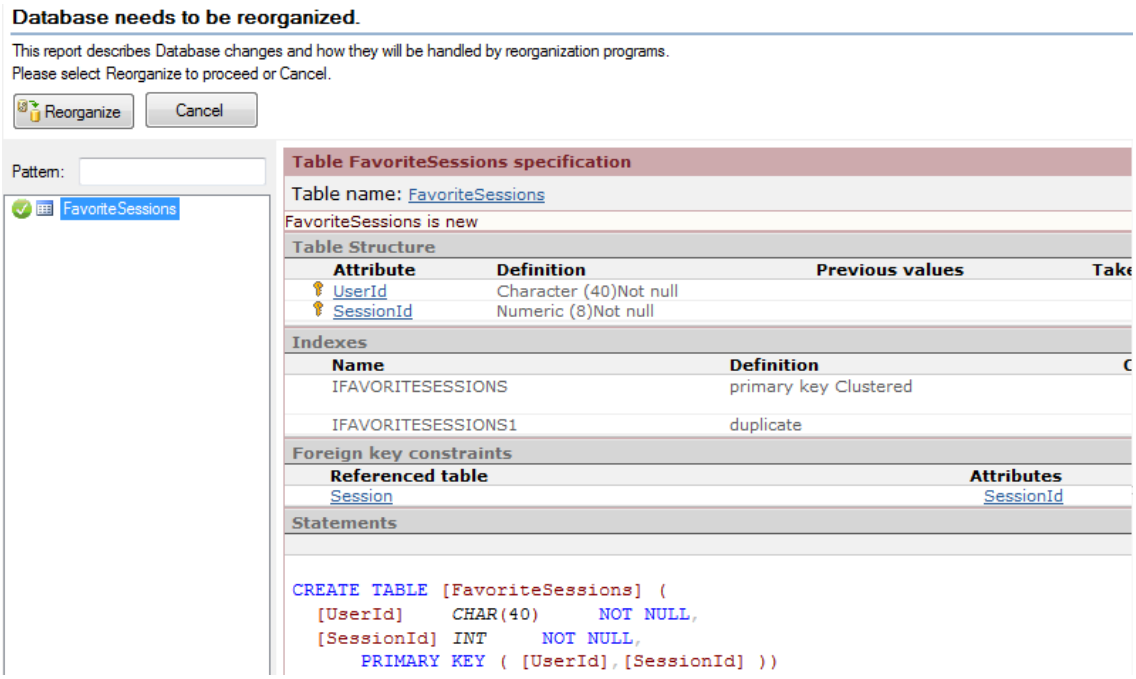
Now we add a button to the dashboard to invoke the login SD panel that was built by the GAM and contains an option to register.

We open the EventDay dashboard, click on Items, right-click and select Add/Action. Next, we select the GAMSDLogin SD panel and in its properties we select the tab_login image.



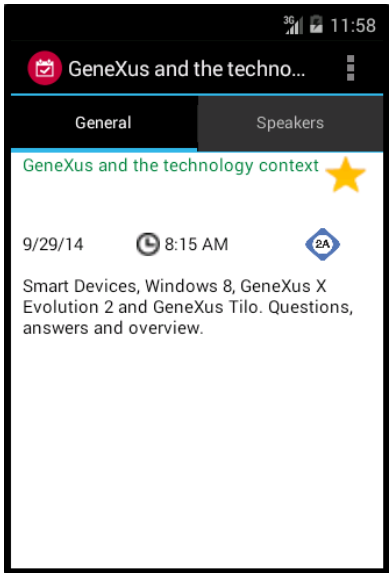
We press F5 to see how it works.

The FavoriteSessions table will be reorganized. We click on Reorganize.

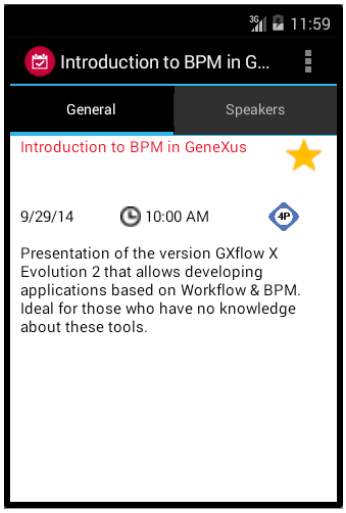


We open the Android emulator and run the application. Because we haven't logged in, we access the application as an anonymous user.

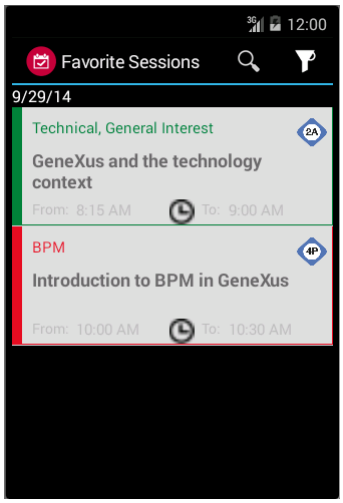
Now we open Sessions, select the first session, and mark it as favorite.



We open the list of sessions again, select the second one and also mark it as favorite.



We open the dashboard and select Favorites. As we can see, the two sessions we marked are now displayed as favorites.



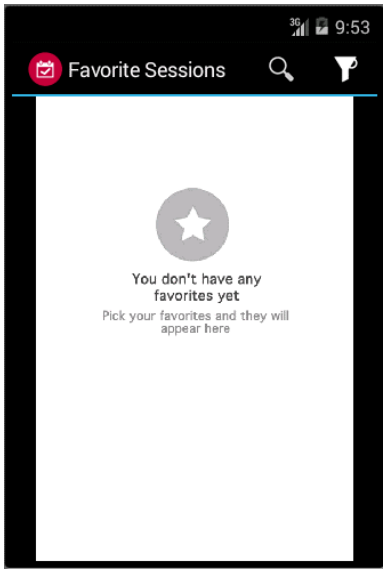
We return to the dashboard, select Login, click on the three dots, choose Register and log in. We enter the user “jsmith”, name “John” and last name “Smith”; his email is jsmith@example.com. The password is “jsmith123”, reenter “jsmith123” and confirm.

If we return to the Favorites, we see that the same sessions are selected, because they were automatically assigned from the anonymous user to user “jsmith”.

We return to the dashboard, press the Menu button and select Logout.

Now we tap on Login and enter the user “admin”, password “admin123”.

If we go to Favorites, we see that the admin user doesn't have any favorites, because from now on the favorite sessions will depend on the user logged into the application, not on the device being used to run it.



Now we want to implement the second requirement.

We've mentioned that we wanted to separate the operations that users could perform in the EventDay application according to their roles, in order to identify those operations corresponding to the mobile backend.

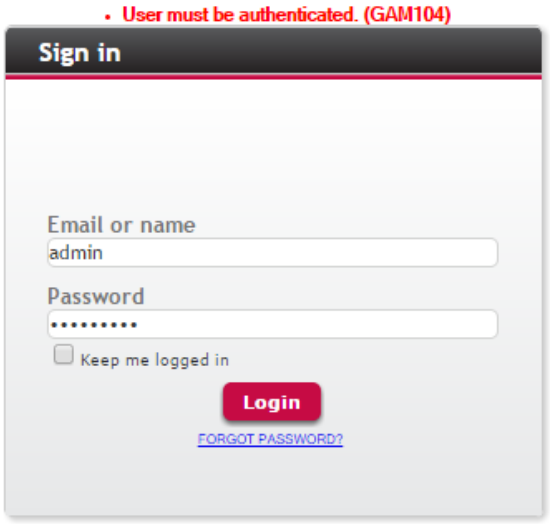
Suppose that we want the information about speakers to be changed only by the event's organizers. To do so, we will create a special role for the organizers, so that users who don't have that role will not be able to perform CRUD operations (Create, Update, Delete) in the Speakers screen.

Any user can enter the system, but not everyone will be able to access the options to insert, update or delete speakers. In order to perform these operations, users must log in with an event organizer profile.

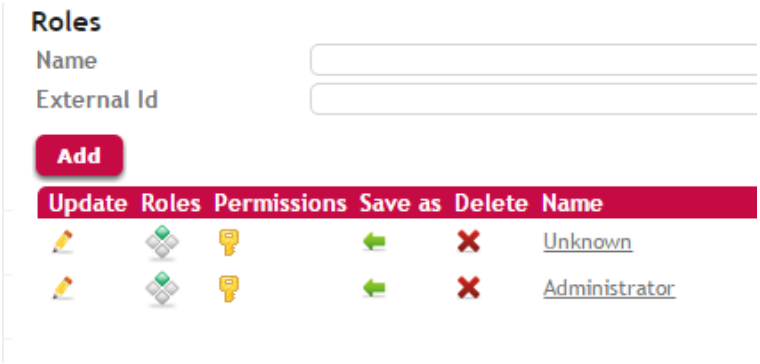
We will begin by creating the organizers role and a user with that role.

To do so, we run the backend provided by the GAM.

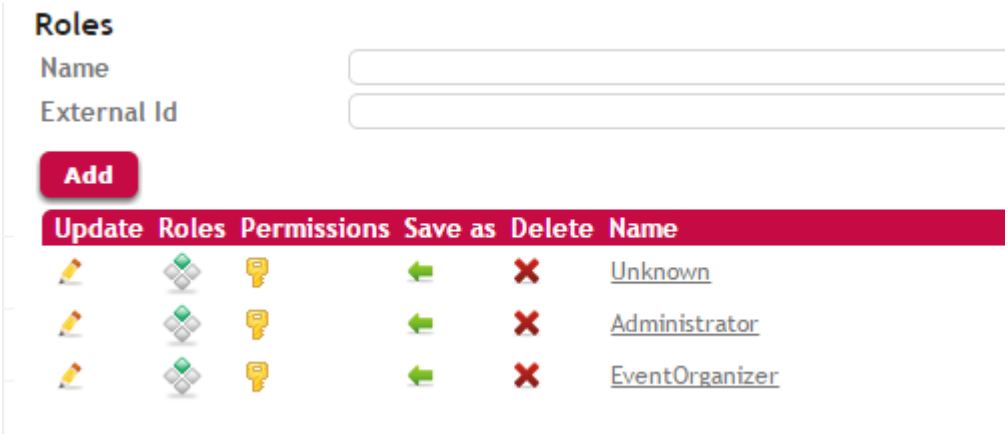
In GeneXus, we open View and select Show QR Codes. We expand the links section and select GAMHome. We log in with username “admin” and password “admin123”.



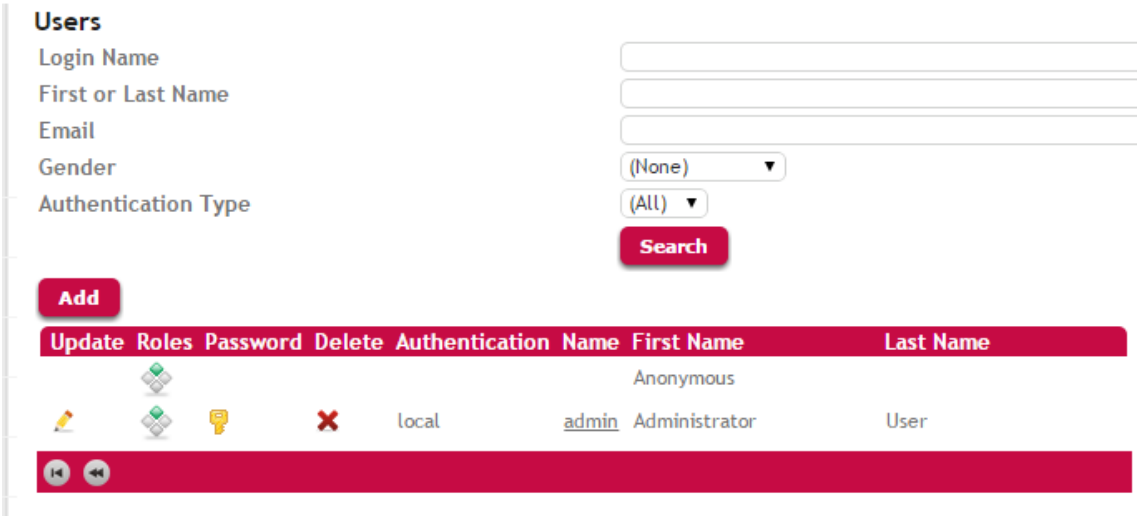
We select the "Roles" option and see that the "Unknown" and "Administrator" roles are available. These roles were automatically created when we enabled the GAM:



We will create a new role called EventOrganizer, so we click on Add, enter the name EventOrganizer and the description "Event Organizer". We Confirm.



Now we will edit the users. To do so, we open the Users option.

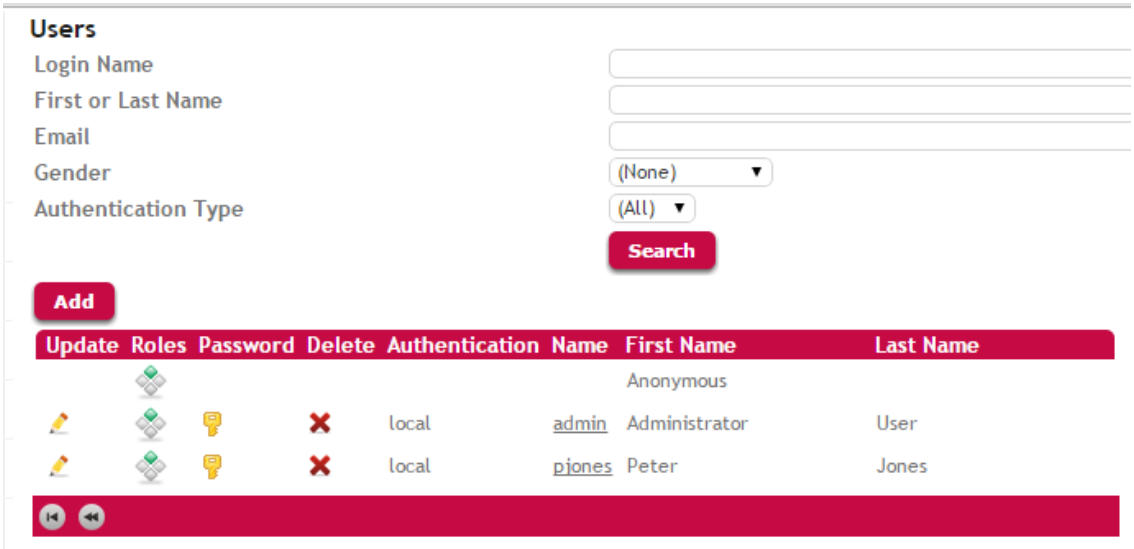


The anonymous and admin users were automatically created. The admin user was based on the administrator role. Now we will create user pjones with the role EventOrganizer.

We select “Add” and define some details of our new user. The user is pjones, his email is pjones@example.com, his name is Peter and his last name is Jones.

The password will be “pjones123” and we confirm it: “pjones123”.

Let's now associate it with the new role. We click on the Role option and select EventOrganizer.

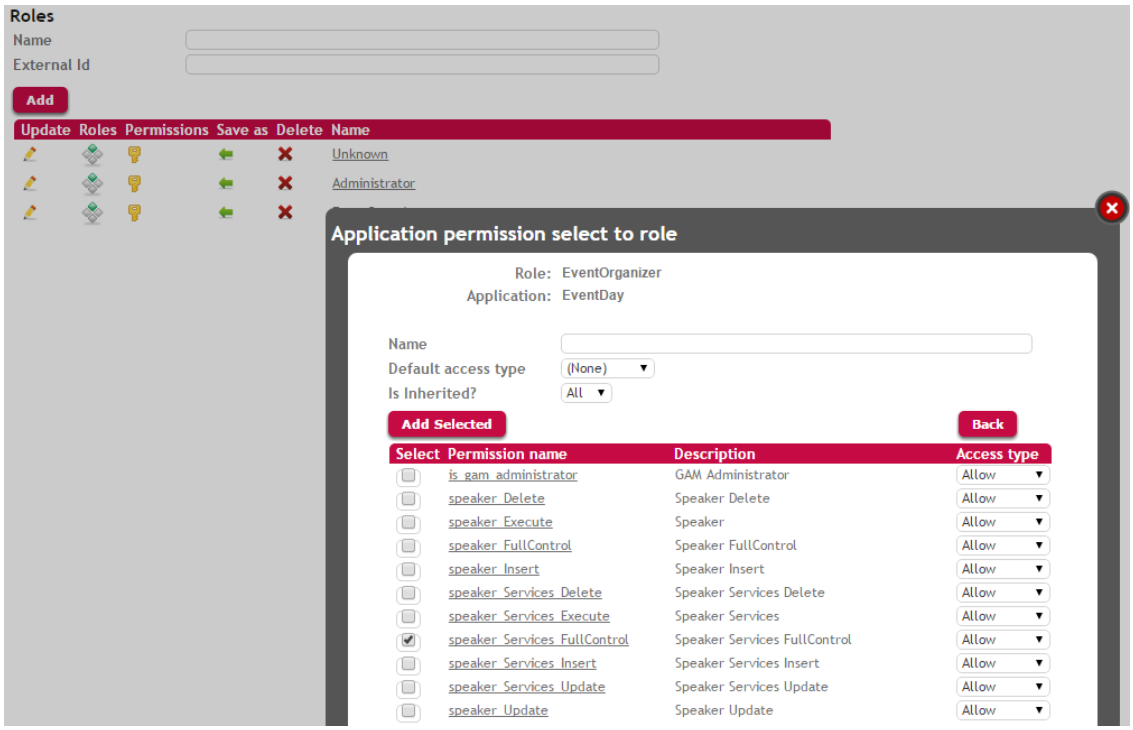


To assign permissions to certain operations in Speakers, we have to increase the security level of the Speaker Business Component so that it not only requires authentication, but also authorization. So, we open the Speaker transaction and assign the **Authorization** value to the **Integrated Security Level** property .

We Rebuild All and press F5…

We click on the GAMHome link and log in with username “admin” and password “admin123”.

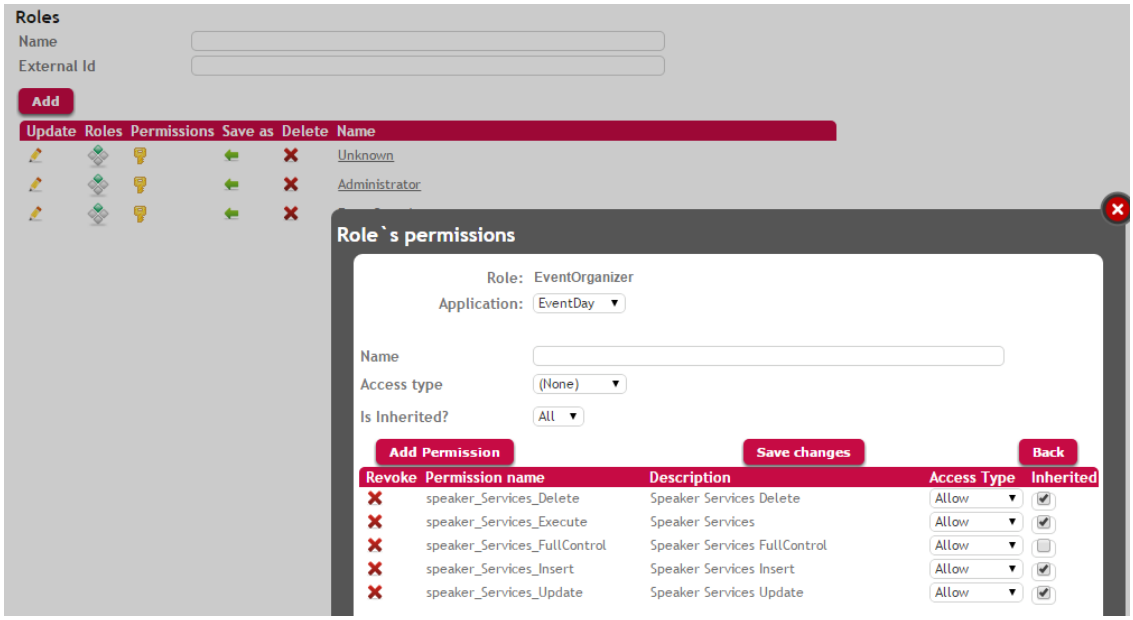
We select Roles and in the EventOrganizer role we click on Permissions. We select the EventDay application (note that the SD application has the same name as the dashboard) and then click on Add Permission.



Note that permissions are only related to the Speakers object, because it was the only object whose security level was set to Authorization.

We select **speaker_Services_FullControl** and click on Save changes.

In this way, this role will now have permissions to run, insert, update and delete objects in the application.



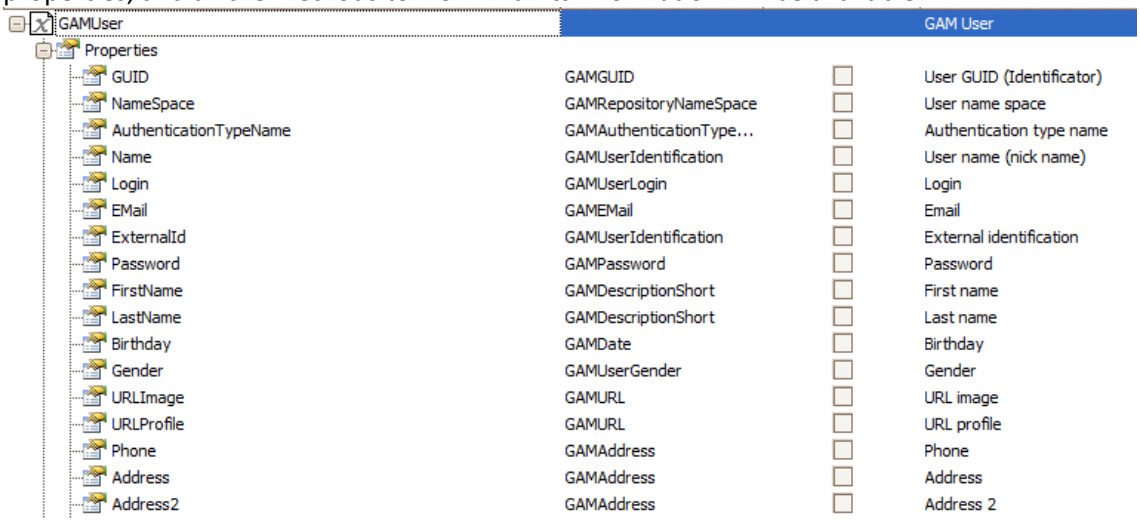
If we want this role to have the same permissions in the web application, we need to do the same for it.

We open Work With Speakers.

The idea is to change this object so that it takes the logged in user role into account. Therefore, if the role is “EventOrganizer” the Insert, Update and Delete buttons must be displayed; otherwise, these buttons will be hidden.

We select the List node and create the Start event. To identify the role we will use the GAM's **GAMuser** API. Among the methods of the GAMUser external object we will use one that returns the associated user's roles.

We will create a variable called &GAMUser in the Work With Speakers object, whose GAMUser data type matches this external object. In this way, its structure will match the external object properties, and all the methods to work with its information will be available.



In the Start event we start this &GAMUser variable with the data returned by the Get() method of the API. In this way, we obtain all of the logged-in user's details (GUID, Name, Password, etc.)

When creating a variable of external object data type, we will be creating a variable with a certain structure. Also, we will have to use the Get method in order to retrieve the user instance in memory and thus access the properties and methods of the user created.

```
Event Start
    &GAMUser = GAMUser.Get()
```

More specifically, we will be able to obtain all the user's roles with the GetAllRoles method, which will return a collection of elements of GAMRole type that is also an external object. The GAMRole type has a property called Name, corresponding to the role's name.

Even though in our case each user has only one associated role, in the future they could have more roles. So, we will have to run through this collection of roles to look for the role called “EventOrganizer” that we're interested in.

We will create two new variables:

- &GAMRole: of GAMRole data type, to run through the roles in that collection.
- A Boolean variable called &isEventOrganizer to find out if the role we're interested in exists.

We type: For the &GAMRole variable in &Gamuser.GetAllRoles…. between brackets we need to indicate a variable to save the potential errors that could happen. We type &Errors, right-click and select Add Variable…

We create the &Errors variable of GAMError data type, and set it as a collection.

Next, we type what we want to do with each role of that collection. In sum, we want to know if their name is “EventOrganizer”, and in that case we will not want to iterate any more.

We close with Endfor.

```
Event Start
    &GAMUser = GAMUser.Get()
    for &GAMRole in &GAMUser.GetAllRoles(&GAmError)
        &isEventOrganizer = False
        If &GAMRole.Name = 'EventOrganizer'
            &isEventOrganizer = True
            Exit
        endif
    endfor

Endevent
```

If the user has the “EventOrganizer” role, we show the Insert button; otherwise, we hide it.

```
Event Start
    &GAMUser = GAMUser.Get()
    for &GAMRole in &GAMUser.GetAllRoles(&GAmError)
        &isEventOrganizer = False
        If &GAMRole.Name = 'EventOrganizer'
            &isEventOrganizer = True
            Exit
        endif
    endfor
    If &isEventOrganizer
        ButtonInsert.Visible = 1
    else
        ButtonInsert.Visible = 0
    endif
Endevent
```

Now we select the Start event code, copy it, go to Section(General) and paste it in the event window. We also copy the variables of the List node to Section(General).

Next, we change the part related to showing and hiding buttons in order to make reference to the Update and Delete buttons.

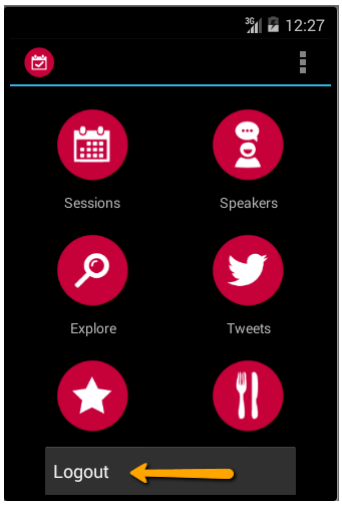
```
Event Start
    &GAMUser = GAMUser.Get()
    for &GAMRole in &GAMUser.GetAllRoles(&GAmError)
        &isEventOrganizer = False
        If &GAMRole.Name = 'EventOrganizer'
            &isEventOrganizer = True
            Exit
        endif
    endfor
    If &isEventOrganizer
        ButtonUpdate.Visible = 1
        ButtonDelete.Visible = 1
    else
        ButtonUpdate.Visible = 0
        ButtonDelete.Visible = 0
    endif
Endevent
```

Let's see what we have done at runtime.

We press F5...

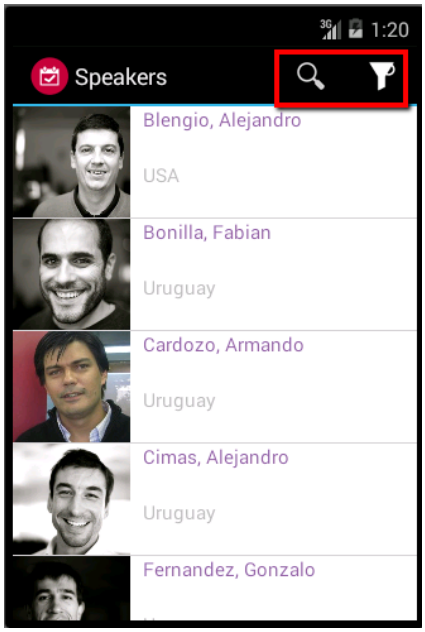
Open the Android emulator and run the application.

When we return to the dashboard, we press the Menu button and select Logout.

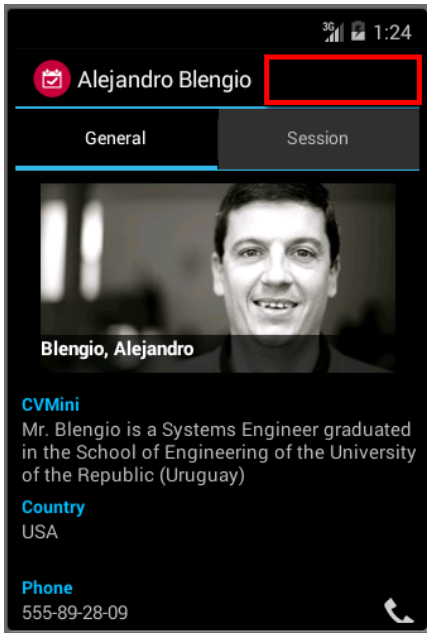


Now we log in with user “jsmith”. Remember that “jsmith” is a regular user, not an event organizer. In the dashboard, we tap on the Speakers icon.

We see that the list of speakers doesn't show the Insert option.

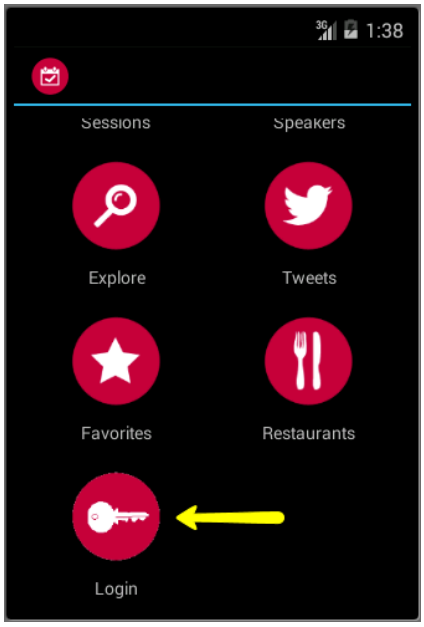


We select a speaker and see that the buttons to update or delete the speaker aren't displayed either.

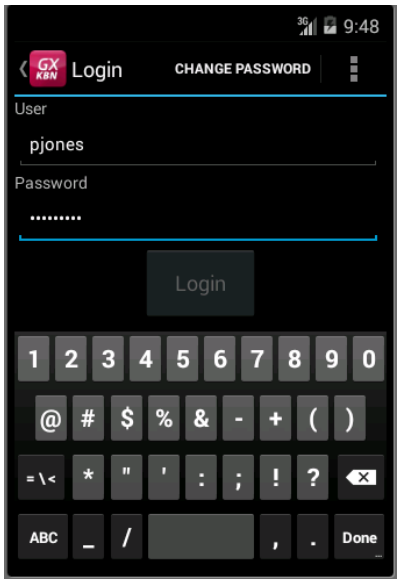


Now we will enter the system with a user who has EventOrganizer permissions. We open the dashboard and log out.

We tap on the Login icon.

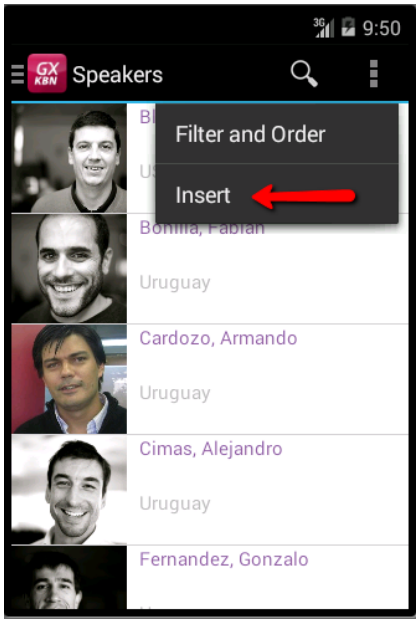


We log in with username “pjones” and password “pjones123”. Remember that user “pjones” was assigned the “EventOrganizer” role.

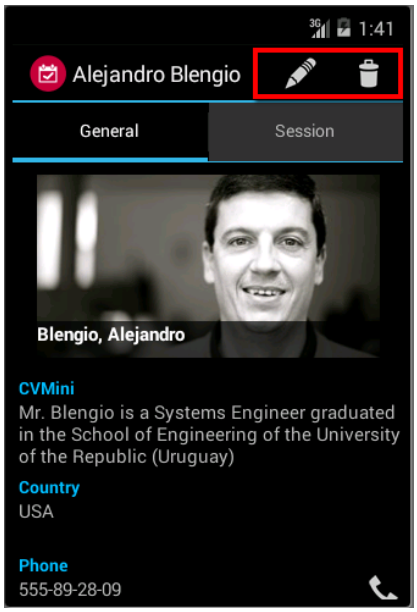


In the dashboard we select Speakers.

In the Work With Speakers screen, the three dots are displayed in the top right corner; we tap and the option to insert a new speaker is displayed.



If we select a speaker, we see that we can also update it or delete it.



In this video we have talked about the possibility of accessing the GAM APIs at runtime in order to access the profile data of users logged into the system and restrict their access to certain operations, depending on their roles.

In this case, we have coded changes to the interface by hiding or showing buttons depending on the user's permissions, which we obtained thanks to the GAM APIs. In addition, by applying the GAM we have ensured that no one can access the REST services in the web server without authentication, even if they have their URL.

You can find more information about the methods and properties of the GAM API in this link:

http://wiki.genexus.com/commwiki/servlet/hwikibypageid?16535