# Lesson 4: Persistence (continued)

Created by Adriaan Klue, last modified by Jacques Marais on Sep 13, 2018

> *As a System Administrator I want to delete other System Administrators.*

## Lesson Outcomes

By the end of this lesson you will have added user deletion to the functionality completed in Lesson 2 and Lesson 3, and thereby have added a full user management component to your app. The key concepts here are:

- "hard" deletion of objects
- best practice "soft" deletion of role objects
- removal of user roles, a.k.a. "uninviting" users

You will also be introduced to in-app alerts (pop-ups).

## New & Modified App Files

`./model/SystemAdmin.mez`

`./services/UserInvite.mez`

`./web-app/lang/en.lang`

`./web-app/presenters/SystemAdminUserMgmt.mez`

`./web-app/views/SystemAdminUserMgmt.vxml`

## Deleting Objects

Assuming your new `<rowAction>` looks like this:

```
<rowAction label="button.remove" action="removeUser">
   <binding variable="selectedSystemAdmin" />
</rowAction>
```

The `delete` built-in function is used like this:

```
void removeUser() {
   SystemAdmin:delete(selectedSystemAdmin);
}
```

Note that the `delete` built-in function is called from the object namespace (and not on a variable holding an instance of the object) and is passed the object instance as parameter. Having executed `delete`, the object instance is now gone forever.

## A "Soft Deletion" Best Practice

Changing an app's data model, like adding an attribute/field to an object, will require the developer to recreate the sandbox app.

When removing an object instance there are various reasons why you might want to keep the record in the database. Common reasons for user role objects are reporting on e.g. how many stock updates were done by a certain user, for which we'll need the reference to the user to still be valid, or simplifying reactivation when a user has been removed by mistake. Instead of deleting an object in such cases, it is recommended to set a deleted/archived/inactive flag, which we'll do for the users in our tutorial's code as well.

```
persistent object SystemAdmin {
   .
   .
   bool deleted;
}
```

`removeUser()` then becomes this:

```
DSL_VIEWS removeUser() {
   selectedSystemAdmin.deleted = true;
   init();
   return null;
}
```

This also means we have to set this flag when first creating the object instance, and check this flag when reading from the database.

System administrators are created in either of two ways in our app, so we need to set the `deleted` flag to **`false`** in both cases:

```
unit SystemAdminUserMgmt;

[...]

DSL_VIEWS saveUser() {
   formSystemAdmin.deleted = false;
   formSystemAdmin.save();
   [...]
```

```
unit UserInvite;
@InviteUser
SystemAdmin inviteSystemAdmin() {
   [...]
   systemAdmin.deleted = false;
   return systemAdmin;
}
```

In our `init()` where we populate the collection bound to the view table, we replace the `:all()` selector with `:equals(deleted, false)`:

```
void init() {
   systemAdmins = SystemAdmin:equals(deleted, false);
   formSystemAdmin = SystemAdmin:new();
   editing = false;
}
```

## Removing User Roles

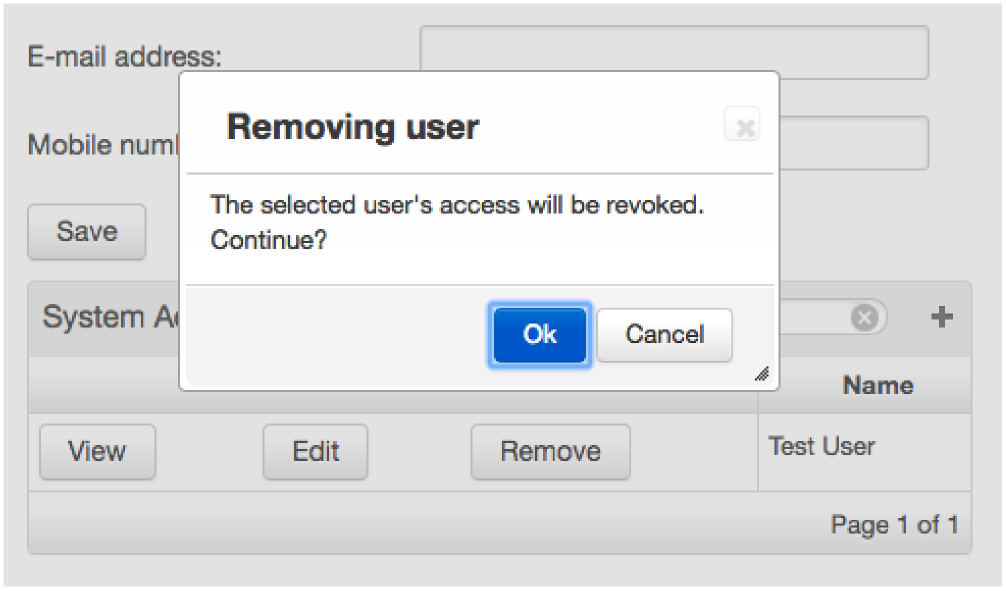
In our case the object instance we deleted also represents an invited app user. Deleting the record usually means we should also revoke that user's access. We "uninvite" the user using the `removeRole` BIF, invoked through an object instance.

```
DSL_VIEWS removeUser() {
   selectedSystemAdmin.deleted = true;
   selectedSystemAdmin.removeRole();
   init();
   return null;
}
```

## Prompting for Confirmation

Another table widget feature allows for a confirmation prompt to appear when the button is clicked. The `<rowAction>` child element itself has an optional child element named `<confirm>` that takes a `subject` and a `body` attribute, both with translation keys from your `.lang` file for values.

```
<rowAction label="button.remove" action="removeUser">
   <binding variable="selectedSystemAdmin" />
   <confirm subject="confirm_subject.removing_user" boc
</rowAction>
```



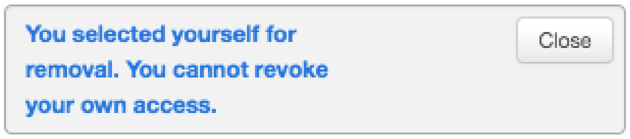
Removing user

The selected user's access will be revoked. Continue?

## Showing an Pop-up Message When You Try to Remove Yourself

Already in Lesson 1 it was demonstrated how to assign a reference to the currently logged in user to a variable. To keep system administrators from revoking their own access, we can check this against `selectedSystemAdmin`. Not to look like a silent failure, we'll also add a pop-up, called "alerts" in Helium: `alert`, `alertError`, and `alertWarn`. These BIFs are available on the `Mez` namespace, and the difference between them is only the appearance of the alert pop-ups.

```
DSL_VIEWS removeUser() {
   if (selectedSystemAdmin == SystemAdmin:user()) {
      Mez:alert("alert.currently_logged_in");
   } else {
      selectedSystemAdmin.deleted = true;
      selectedSystemAdmin.removeRole();
   }
   init();
   return null;
}
```


You selected yourself for removal. You cannot revoke your own access.

The `alertWarn` and `alertError` BIFs display orange and red text, respectively.

## Lesson Source Code

Lesson 4.zip

remove role helium user delete app

## 2 Comments

Alexander Thompson

Is there a way to get a popup confirmation screen that returns a variable but still continues to the function if you click "OK" or "CANCEL" ? - Meaning based on the variable being returned on the confirmation screen you can do 2 separate functionalities based on the result, instead of not proceeding to the function when user press cancel.

Adriaan Klue

@ Alexander Thompson No, currently the cancel action can't be made to do anything other than cancel. Will need a feature request for what you describe.