CST8218 Lab4: User Database

Submit your final results to Brightspace and demonstrate to your lab professor for 1+2=3 marks (3% of course grade)

See Brightspace for due date.

Objective: Implement user accounts and an administration area for performing Create, Read, Update, and Delete operations on accounts to represent users for an arbitrary JEE Web Application

Instructions:

You are preparing to add security to your Sprite application (or any other application) by making yet another application called **AppUser** that manages an appusers database table called **appuser**. This is a simple adaptation of what you did in Lab 1/Lab 2 where you made a JEE application that can do the CRUD operations on Contacts. Now you need to do basically the same thing except you'll adapt the Entity for your current purposes. For authentication, userid, password, and groupname fields are needed, but phone numbers and birthdays, for example, are not needed.

Create the project for the new application, and create your Entity file (AppUser.java). You can create that file however you like, and one way is to create a Contact entity as in Lab 1, then just after "Create Entity from Database", refactor->rename the Contact entity to AppUser.java, and change the fields of the entity to be fields appropriate to an appuser Entity: userid, password, and groupname (all String data type in the Java class). Do a search for CONTACT or Contact in your AppUser.java file to make sure no "table=CONTACT" string is present and your database table will be named correctly. Then create your JSF pages from Entity, etc, as you did in Lab 1.

Instructions (about that password…)

When creating a new user account, the user will type in text for the account password, and when they submit the request, your application needs to set the hashed/salted value of the password before the new AppUser is stored in the database (do not use a converter for the password).

When we are processing hashed passwords, our approach will be for the hashing/salting procedure to occur in the setter for the password field. Because hashed/salted passwords cannot be reversed, it makes sense for the getter of the password field to return an empty string (not the password itself, which should be impossible). This way, when an appuser is edited on the Edit.xhtml page, the password field will display as empty, and if the editing user does not want to change the password, they leave the password empty. When the password setter is passed an empty password entry (zero-length string), it means that the password should not be changed. With this convention, it is impossible to change a password to the empty string.

The following java code is provided for you to read and understand (for example, look up PasswordHash in the Java EE 8 API Documentation). You will need to adapt this code to your purposes of handling any password besides "mySecretPassword". The code as it is will take a plaintext password (String) "mySecretPassword" and create a password entry that is hashed and salted with the default PasswordHash parameters:

```
// initialize a PasswordHash object which will generate password hashes
```

```
Instance<? extends PasswordHash> instance = CDI.current().select(Pbkdf2PasswordHash.class);
PasswordHash passwordHash = instance.get();
passwordHash.initialize(new HashMap<String,String>());  // todo: are the defaults good enough?
// now we can generate a password entry for a given password
String passwordEntry = "mySecretPassword"; //pretend the user has chosen a password mySecretPassword
passwordEntry = passwordHash.generate(passwordEntry.toCharArray());
//at this point, passwordEntry refers to a salted/hashed password entry corresponding to mySecretPassword
```

Next Steps

After implementing the AppUser application, you are ready to proceed with securing your Sprite application when the time comes (or any JEE application) against this table of appusers.
Preview: here is an annotation that goes with our appuser table. This can be added to any JEE application to give it an identity store to authenticate against our database of users:

```
@DatabaseIdentityStoreDefinition(

  dataSourceLookup = "${'java:comp/DefaultDataSource'}",

  callerQuery = "#{'select password from app.appuser where userid = ?'}",

  groupsQuery = "select groupname from app.appuser where userid = ?",

  hashAlgorithm = PasswordHash.class,

  priority = 10

)
```