Simple Authorisation Tutorial.
Andrew Stratton

This tutorial aims to show how to set up a simple login using Grails. This is not meant to be a first tutorial; you should have already run through the quick start grails tutorial.

A more complex, and complete, authorisation example can be seen in the CMS example within the grails download.

Firstly we start with a user model containing email (used as login) and password. This is the model used for login and is also stored in the session. The sample below only shows a few constraints in place:

```
class User
{
	Long id
	Long version

	String email
	String password

	String toString()
		{ "$email" }

	def constraints =
	{
		email(email:true)
		password(blank:false, password:true)
	}
}
```

Next we add a sample user in the bootstrap in grails-app/conf. This just creates an initial user for testing login and saves writing registration code at this stage:

```
class ApplicationBootStrap {

    def init = { servletContext ->
        new User(email:"eg@eg.com",password:"password").save()
    }
    def destroy = {
    }
}
```

Next create a simple Plant model, as a sample for viewing. This model is our test model being used for demonstration purposes:

```
class Plant {
	Long id
	Long version

	String description
	Boolean validated
	String hardiness
	Boolean evergreen
	String annual
	String genus
	String genusHybrid
	String species
	String speciesHybrid
	String variety
	String subSpecies
```

```
        String cultivar
        String forma

        def constraints = {
              hardiness(inList:["Hardy", "Half Hardy", "Tender"])
              annual(inList:["Annual", "Perennial", "Biennial"])
        }

        String toString() { "${this.class.name} :  $id" }

        boolean equals(other) {
              if(other?.is(this))return true
              if(!(other instanceof Plant)) return false

              if(!id || !other?.id || id!=other?.id) return false

              return true
        }

        int hashCode() {
              int hashCode = 0
              hashCode = 29 * (hashCode + ( !id ? 0 : id ^ (id >>> 32) ) )
        }
}
```

Please pardon the length of this class - this is taken straight from my current application.

Next we need to create a PlantController:

```
        class PlantController {
              def beforeInterceptor = [action:this.&checkUser,except:
        ['index','list','show']]
              def scaffold = true

              def checkUser() {
                    if(!session.user) {
                          // i.e. user not logged in
                          redirect(controller:'user',action:'login')
                          return false
                    }
              }
        }
```

This controller has some extra features. Firstly, it adds in a beforeInterceptor that calls a method before any of the controller's methods (this concept should be familiar to aspect oriented developers. In this case, checkUser is called - the & just acts as a reference/pointer to the method. There is also an except list, which disables the interceptor for the index, list and show methods.

> *The effect of this is to call checkUser before the standard scaffold methods create, edit, delete, save and update. i.e we check for login when creating, updating or deleting (CUD), but not when reading or showing lists of plants.*

The checkUser method simply checks to see if there is a user object in the session object and redirects the output to user/login if the user object is missing.

N.B. Note how the beforeInterceptor returns false when invalid - which is if the session does not contain a user entry. Since we haven't yet added a user entry, we expect to be redirected for create/update/delete plant methods (i.e. except list, index and show).

> *If you are confused by what is going - as I was - you might need to check out the quick start tutorial again - but in brief, the 'def scaffold = true' in a Controller gives us CRUD, in the*

form of index, list, show, create, edit, delete, save and update methods, which are url mapped to the PlantController methods - so http://localhost:8080/plantalert/plant/create will allow you to create a new plant - assuming you're logged in. If you're not logged in, you will be taken to the login screen. The except list means that list and show will avoid any login checking, so http://localhost:8080/plantalert/plant/list ignores whether you have logged in or not (n.b. you still stay logged in - if you were already).

Note - if you have looked at the CMS example - the example above is much simpler and doesn't use a 'BaseController' to make the example simpler.

Now run the application and try out the current situation with a call to http://localhost:8080/plantalert/plant (*plantalert* is the name of my application). You should see the following (showing that the list method is not calling checkUser):

If nothing happens - make sure you've done a 'grails run-app' and there are no errors (e.g. port clashing with tomcat can be a problem). If it's still not working - please try following the quickstart tutorial again.

Once you see the above, try and follow the New Plant link. You should get:

HTTP ERROR: 404
/WEB-INF/grails-app/views/user/login.jsp

RequestURI=/plantalert/user/login

Powered by Jetty://

We've been redirected to user/login view - but we haven't created it yet. So let's create login.gsp:

```
<html>
    <head>
         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
                     <meta name="layout" content="main" />
         <title>User Login</title>
    </head>
    <body>
        <div class="body">
           <g:form action="doLogin" method="post">
               <div class="dialog">
                              <p>Enter your login details below:</p>
                <table  class="userForm">
                  <tr class='prop'>
                      <td valign='top' style='text-align:left;' width='20%'>
                          <label for='email'>Email:</label>
                      </td>
                      <td valign='top' style='text-align:left;' width='80%'>
                          <input id="email" type='text' name='email' value='${user?.email}' />
```

```
                    </td>
                </tr>

                <tr class='prop'>
                    <td valign='top' style='text-align:left;' width='20%'>
                        <label for='password'>Password:</label>
                    </td>
                    <td valign='top' style='text-align:left;' width='80%'>
                        <input id="password" type='password' name='password'
                               value='${user?.password}' />
                    </td>
                </tr>

            </table>
            </div>
            <div class="buttons">
                  <span class="formButton">
                     <input type="submit" value="Login"></input>
                  </span>
            </div>
        </g:form>
    </div>
</body>
</html>
```

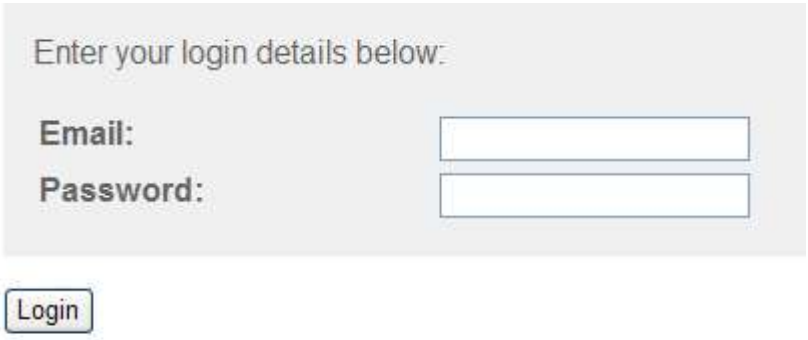
N.B. This is a chopped down version of the CMS example.

We also need a UserController with a login method:

```
class UserController {

    def login = {
      }
}
```

This makes sure that http://localhost/plantalert/user/login actually maps somewhere. Now restart the app and go to http://localhost:8080/plantalert/plant, then click on the new plant button to get:



So, now we get redirected, to /user/login, when we're not logged in, except for list, index and show which ignore login. If you try to login, you will get:

HTTP ERROR: 404

/WEB-INF/grails-app/views/user/doLogin.jsp

RequestURI=/plantalert/user/doLogin

Powered by Jetty://

So, next we need to add the doLogin to the UserController. Here is the whole code for the controller:

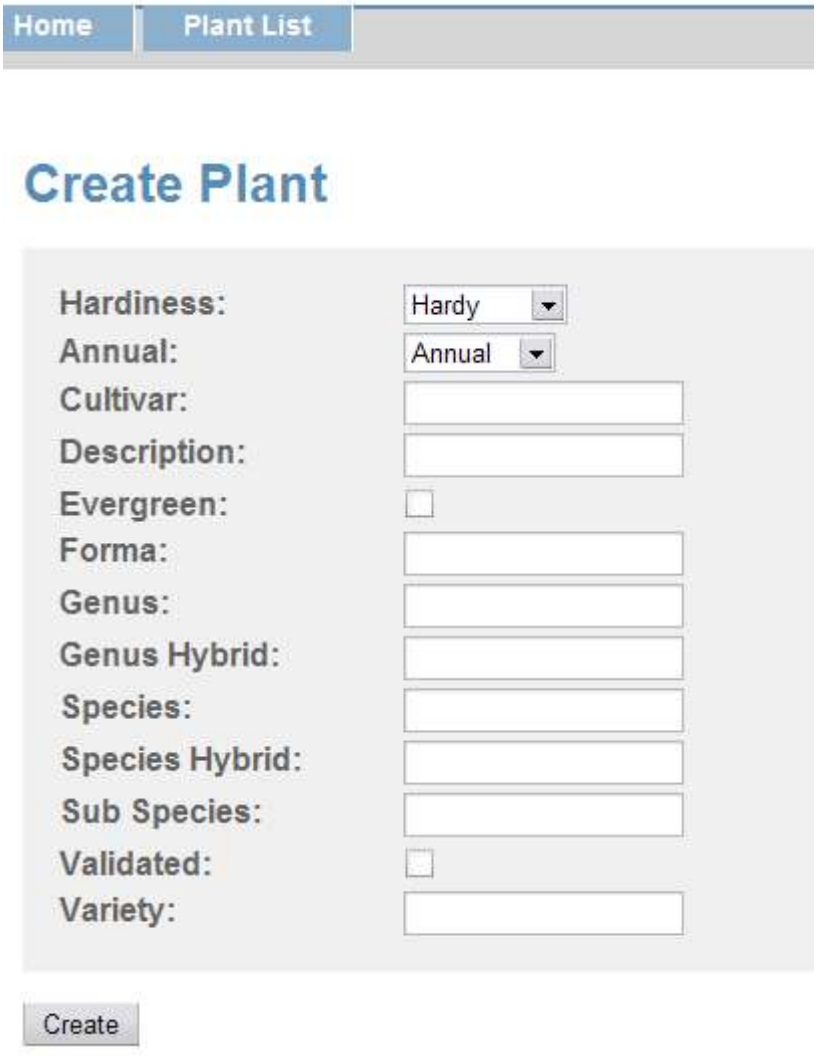
```
class UserController {

    def login = {
      }

    def doLogin = {
      def user = User.findWhere(email:params['email'],
                  password:params['password'])
      session.user = user
      if (user)
                  redirect(controller:'plant',action:'list')
            else
                  redirect(controller:'user',action:'login')
      }
}
```

The User.findWhere is a nice easy way of doing a “where email=’email’ and password=’password’”. We store the resulting object in the session - which will replace any existing copy. Then we redirect to plant/list (if successful) or user/login (again) if not. Currently there is no message, so if you have a nice fast machine, you may not notice the delay before the login page reloads (but the password is blanked). On my development machine I don’t have such a problem :(.

Now try logging in with a valid user (eg@eg.com) and password (password), after clicking login, you should get the plant list (which didn’t require you to be logged in anyway). Click on new plant and you should get:



You can also check that you get redirected back to the login if you try logging in with the wrong password.

Other things we need to do - which may follow in a subsequent tutorial are:

1. Add a logout option. This would simply set session.user to null.
2. Add registration. The supplied email would need to be unique.
3. Add an AuthorisationController that contains the standard code, so our controllers can sub class it.
4. Add other details for user. The model could be extended with extra information, e.g. in my case, the plant hardiness zone the user lives in.
5. We need a facility to email lost passwords.

I hope this helps. Best of luck with Grails.

Andy Stratton.
email andyATstrattonenglishDOTcoDOTuk.